HIGH TECH

Х. МАРМАНИС, Д. БАБЕНКО

АЛГОРИТМЫ ИНТЕЛЛЕКТУАЛЬНОГО ИНТЕРНЕТА

ПЕРЕДОВЫЕ МЕТОДИКИ СБОРА, АНАЛИЗА И ОБРАБОТКИ ДАННЫХ

СИМВОЛ®

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-186-8, название «Алгоритмы интеллектуального Интернета» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, атакже сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

Algorithms of the Intelligent Web

Haralambos Marmanis
Dmitry Babenko

MANNING

Алгоритмы интеллектуального Интернета

Хараламбос Марманис,
Дмитрий Бабенко

Санкт-Петербург — Москва
2011

Хараламбос Марманис,
Дмитрий Бабенко

Алгоритмы интеллектуального Интернета

Передовые методики сбора, анализа и обработки данных

Перевод М. Низовец

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Научный редактор	*С. Щербак*
Редактор	*Т. Темкина*
Корректор	*С. Минин*
Верстка	*К. Чубаров*

Марманис Х., Бабенко Д.

Алгоритмы интеллектуального Интернета. Передовые методики сбора, анализа и обработки данных. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 480 с., ил.

ISBN 978-5-93286-186-8

Важный аспект современных коммерчески успешных приложений – применение методик, позволяющих осуществлять обработку информации и добавлять средства интеллектуальной поддержки. Многочисленные примеры успешных проектов, основанных на применении таких методик, включают такие широко известные бренды, как Google, Netflix и Amazon. Эта книга о том, как построить алгоритмы, формирующие интеллектуальное ядро таких веб-приложений.

В книге рассматриваются пять важных категорий алгоритмов: поиск, выработка рекомендаций, создание групп, классификация и ансамбли классификаторов. Исходный код написан на языке Java, тем не менее программистам, знающим другой объектно-ориентированный язык, вполне по силам разобраться в этом коде и использовать общие принципы с учетом своей специфики. Материал в равной степени применим к различным приложениям – от утилит мобильной связи до традиционных настольных приложений.

Издание в первую очередь адресовано программистам и веб-разработчикам, однако множество примеров и новых идей будут полезны и руководителям разного уровня, желающим лучше разобраться в соответствующих технологиях и предлагаемых возможностях с точки зрения бизнеса.

ISBN 978-5-93286-186-8
ISBN 978-1-933988-66-5 (англ)

© Издательство Символ-Плюс, 2011
Authorized translation of the English edition © 2009 Manning Publications Co. This translation is published and sold by permission of Manning Publications Co., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 29.04.2011. Формат 70×100 1/16. Печать офсетная.
Объем 30 печ. л. Тираж 1200 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие 11

Благодарности 14

Об этой книге 16

Глава 1. Что такое интеллектуальный Интернет? 23

1.1. Примеры интеллектуальных веб-приложений 25
1.2. Базовые элементы интеллектуальных приложений 26
1.3. Что могут выиграть приложения от интеллектуальности? 29
1.3.1. Сайты социальных сетей 29
1.3.2. Гибридные веб-приложения (мэшапы) 31
1.3.3. Порталы 32
1.3.4. Вики-сайты 33
1.3.5. Сайты общего доступа к медиафайлам 34
1.3.6. Онлайн-игры 35
1.4. Как встроить интеллект в мое приложение? 36
1.4.1. Анализ функциональности и данных 36
1.4.2. Получение дополнительных данных из Интернета 37
1.5. Машинное обучение, интеллектуальный анализ данных и так далее 42
1.6. Восемь заблуждений насчет интеллектуальных приложений 44
1.6.1. Заблуждение 1: данные достоверны 45
1.6.2. Заблуждение 2: логический вывод осуществляется мгновенно 46
1.6.3. Заблуждение 3: размер данных не имеет значения 46
1.6.4. Заблуждение 4: масштабируемость решения – не проблема 46
1.6.5. Заблуждение 5: одна хорошая библиотека годится на все случаи 47
1.6.6. Заблуждение 6: время вычислений известно 47
1.6.7. Заблуждение 7: чем сложнее модель, тем лучше 47
1.6.8. Заблуждение 8: существуют модели без систематической ошибки 48

1.7. Заключение 48
1.8. Ссылки 49

Глава 2. Поиск 51

2.1. Поиск с применением библиотеки Lucene 52
2.1.1. Программный код библиотеки Lucene 54
2.1.2. Анализ основных этапов поиска 61
2.2. Зачем нужен поиск вне индексов? 65
2.3. Уточнение результатов поиска на основе анализа ссылок 67
2.3.1. Алгоритм PageRank 67
2.3.2. Вычисление вектора PageRank 69
2.3.3. alpha: эффект телепортации между веб-страницами 71
2.3.4. Основные сведения о степенном методе 73
2.3.5. Объединение оценок индексирования и оценок PageRank 78
2.4. Уточнение результатов поиска на основе анализа экранных данных 82
2.4.1. Первое знакомство с анализом экранных данных 82
2.4.2. Применение наивного байесовского классификатора 85
2.4.3. Объединение оценок индексирования Lucene, вектора PageRank и данных о переходах пользователя по ссылкам 90
2.5. Ранжирование документов Word, PDF и других документов без ссылок 94
2.5.1. Введение в алгоритм DocRank 95
2.5.2. Внутренние механизмы алгоритма DocRank 96
2.6. Проблемы масштабной реализации 102
2.7. Получили ли вы то, что искали? Точность и выборка 105
2.8. Заключение 108
2.9. Сделать 109
2.10. Ссылки 112

Глава 3. Выработка предложений и рекомендаций 113

3.1. Музыкальный интернет-магазин: основные понятия 114
3.1.1. Понятия расстояния и сходства 115
3.1.2. Подробнее о вычислении сходства 120
3.1.3. Какую из формул вычисления сходства предпочесть? 125
3.2. Как работают системы выработки рекомендаций? 126
3.2.1. Рекомендации на основе сходства пользователей 127
3.2.2. Рекомендации на основе сходства предметов 138
3.2.3. Рекомендации на основе контента 142
3.3. Выработка рекомендаций по друзьям, статьям и новостным сообщениям 150

3.3.1. Знакомство с сайтом MyDiggSpace.com 151
3.3.2. Нахождение друзей 152
3.3.3. Внутренние механизмы класса DiggDelphi 155
3.4. Рекомендации фильмов на сайте, подобном сайту Netflix.com 161
3.4.1. Введение в наборы данных о кинофильмах и рекомендателях 161
3.4.2. Нормализация данных и коэффициенты корреляции 165
3.5. Масштабная реализация и вопросы оценки 171
3.6. Заключение 173
3.7. Сделать 174
3.8. Ссылки 177

Глава 4. Кластеризация: объединение в группы 179

4.1. Необходимость кластеризации 180
4.1.1. Группы пользователей на веб-сайте (конкретный случай) 181
4.1.2. Нахождение групп с помощью SQL-предложения order by 183
4.1.3. Нахождение групп путем сортировки массива 185
4.2. Обзор алгоритмов кластеризации 188
4.2.1. Классификация алгоритмов кластеризации по структуре кластеров 189
4.2.2. Классификация алгоритмов кластеризации по типу и структуре данных 190
4.2.3. Классификация алгоритмов кластеризации по размеру обрабатываемых данных 192
4.3. Алгоритмы связей 193
4.3.1. Дендрограмма: базовая структура данных кластера 193
4.3.2. Знакомство с алгоритмами связей 196
4.3.3. Алгоритм одной связи 198
4.3.4. Алгоритм средней связи 200
4.3.5. Алгоритм минимального остовного дерева 203
4.4. Алгоритм k-средних 206
4.4.1. Первое знакомство с алгоритмом k-средних 207
4.4.2. Внутренние механизмы работы алгоритма k-средних 208
4.5. Устойчивая кластеризация, использующая связи (ROCK) 211
4.5.1. Введение в алгоритм ROCK 212
4.5.2. Почему ROCK – это надежно? 213
4.6. DBSCAN 218
4.6.1. Первое знакомство с алгоритмами на основе плотности 218
4.6.2. Внутренние механизмы алгоритма DBSCAN 221

4.7. Вопросы кластеризации очень больших наборов данных 226
4.7.1. Вычислительная сложность 226
4.7.2. Большая размерность 228
4.8. Заключение 229
4.9. Сделать 231
4.10. Ссылки 233

Глава 5. Классификация: размещение по принадлежности 235

5.1. Необходимость классификации 237
5.2. Обзор классификаторов 241
5.2.1. Алгоритмы структурной классификации 243
5.2.2. Статистические алгоритмы классификации 245
5.2.3. Жизненный цикл классификатора 246
5.3. Автоматическая категоризация почтовых сообщений и фильтрация спама 248
5.3.1. Наивная байесовская классификация 250
5.3.2. Классификация по правилам 266
5.4. Обнаружение мошенничества с помощью нейронных сетей 280
5.4.1. Сценарий выявления мошенничества в транзакционных данных 281
5.4.2. Обзор нейронных сетей 283
5.4.3. Детектор мошенничества на основе нейронной сети в действии 285
5.4.4. Анатомия нейронной сети детектора мошенничества 291
5.4.5. Базовый класс для создания универсальных нейронных сетей 299
5.5. Можно ли доверять полученным результатам? 305
5.6. Классификация очень больших наборов данных 310
5.7. Заключение 313
5.8. Сделать 315
5.9. Ссылки 320

Глава 6. Объединение классификаторов 323

6.1. Кредитоспособность: анализ примера объединения классификаторов 325
6.1.1. Краткое описание данных 327
6.1.2. Создание искусственных данных для реальных задач 332
6.2. Оценка кредитоспособности с помощью единственного классификатора 337
6.2.1. Основы применения наивного байесовского классификатора 337

6.2.2. Основы применения дерева решений 340
6.2.3. Основы применения нейронных сетей 343
6.3. Сравнение классификаторов в применении к одним и тем же данным 346
6.3.1. Тест Макнемара 347
6.3.2. Тест на разность пропорций 350
6.3.3. Q-тест Кохрана и F-тест 352
6.4. Bagging – самонастраиваемое объединение 355
6.4.1. Bagging-классификатор в действии 357
6.4.2. Заглянем внутрь bagging-классификатора 359
6.4.3. Ансамбли классификаторов 362
6.5. Boosting – итеративный подход к улучшению 365
6.5.1. Boosting-классификатор в действии 366
6.5.2. Заглянем внутрь boosting-классификатора 368
6.6. Заключение 373
6.7. Сделать 375
6.8. Ссылки 380

Глава 7. Все вместе: интеллектуальный новостной портал 381

7.1. Обзор функциональности 383
7.2. Получение и обработка контента 385
7.2.1. На старт, внимание, краулинг! 385
7.2.2. Обзор предварительных условий поиска 386
7.2.3. Используемый по умолчанию набор извлеченных и обработанных новостных сообщений 389
7.3. Поиск новостных сообщений 391
7.4. Распределение по новостным категориям 394
7.4.1. Порядок имеет значение! 395
7.4.2. Классификация с помощью класса NewsProcessor 400
7.4.3. Знакомьтесь: классификатор 402
7.4.4. Стратегия классификации: выход за пределы низкоуровневой категоризации 405
7.5. Формирование групп новостей с помощью класса NewsProcessor 408
7.5.1. Кластеризация обычных новостных сообщений 409
7.5.2. Кластеризация новостных сообщений в категории новостей 414
7.6. Динамический контент на базе пользовательских оценок 418
7.7. Заключение 421
7.8. Сделать 422
7.9. Ссылки 428

Приложение A. Введение в BeanShell 429
A.1. Что такое BeanShell? 429
A.2. Зачем нужен язык BeanShell? 430
A.3. Выполнение BeanShell 430
A.4. Ссылки 431

Приложение B. Краулинг 432
B.1. Обзор компонентов поискового робота 432
B.1.1. Этапы краулинга 433
B.1.2. Наш простой поисковый робот 434
B.1.3. Поисковые роботы с открытым исходным кодом 435
B.2. Ссылки 436

Приложение C. Памятка по математике 437
C.1. Векторы и матрицы 437
C.2. Измерение расстояний 438
C.3. Развитые матричные методы 440
C.4. Ссылки 441

Приложение D. Обработка естественных языков 442

Приложение E. Нейронные сети 445

Алфавитный указатель 448

Предисловие

Во время обучения в аспирантуре я познакомился с проблемами машинного обучения и, в частности, с распознаванием образов. Центральное место в моей работе занимало математическое и численное моделирование, но возможность распознавать образы в большом объеме данных нашла очевидные применения во многих областях. Последующие годы подвели меня к теме машинного обучения даже ближе, чем я себе это представлял.

В 1999 году я оставил академические занятия и приступил к работе в промышленном секторе. В одном из моих консалтинговых проектов мы пытались определять риск сердечной недостаточности на основании (главным образом) данных ЭКГ пациентов. В подобных задачах точная математическая формулировка либо невозможна, либо ее нельзя реализовать. Моделирование (наше программное обеспечение) должно было опираться на методы, способные повысить точность прогнозирования за счет имеющегося набора историй болезни пациентов, для которых риск сердечной недостаточности уже был диагностирован кардиологом. Другими словами, мы искали методы, которые могли бы «обучаться» на своих входных данных.

Тогда, в 1990-х, благодаря стечению обстоятельств новая индустрия развивалась быстро. Интернет стал вездесущим! По закону Мура, ЦП продолжали наращивать быстродействие и дешеветь. Модули ОЗУ, жесткие диски и другие компьютерные компоненты следовали тем же тенденциям совершенствования возможностей и снижения стоимости. Не отставая от них, продолжала возрастать пропускная способность типичного сетевого подключения, которое одновременно становилось все более доступным по цене. Более того, были востребованы и появились надежные технологии разработки веб-приложений, а быстрое распространение проектов с открытым исходным кодом, охватывающих все направления разработки программного обеспечения, ускоряло развитие соответствующих методов. Все эти факторы внесли свой вклад в построение всеобъемлющей цифровой экосистемы, которую мы сегодня называем Интернетом.

Естественно, первостепенной задачей, вставшей перед представителями нашей профессии – разработчиками программного обеспечения и веб-разработчиками всего мира, – стала разработка таких технологий, которые позволили бы создавать надежные, масштабируемые и эстетически

привлекательные веб-приложения. Таким образом, за последние десять лет было приложено много сил, чтобы достичь этих целей, и налицо значительный прогресс. Разумеется, совершенство – это конечная цель, а не состояние, так что нам еще есть куда расти. Но что касается надежности, масштабируемости и эстетической привлекательности, то здесь, по-видимому, наша эффективность уже не повышается. Эра «обустройства» интернет-приложений более или менее закончилась. Простое агрегирование данных и несложные модели типа «запрос пользователя/ответ системы», основанные на заданной логике, достигли состояния зрелости.

Сегодня в некоторых приложениях можно обнаружить другую волну инноваций, и эта волна довольно быстро распространяется благодаря повышению уровня просвещения. Именно такие приложения мы называем в этой книге *интеллектуальными приложениями*. В отличие от традиционных, интеллектуальные приложения настраивают свое функциональное поведение в соответствии с полученными ими входными данными, во многом подобно тому, как моя программа моделирования должна была диагностировать риск сердечной недостаточности по данным ЭКГ.

За последние пять лет мне стало ясно, что многие методики, применяемые в интеллектуальных приложениях, труднодоступны для подавляющего большинства профессионалов в области программного обеспечения. Думаю, главные причины здесь две. Во-первых, коммерческий потенциал инноваций в этих областях способен обеспечить гигантскую финансовую прибыль. Имеет смысл (с финансовой точки зрения) защищать право собственности на компоненты конкретных приложений и скрывать важнейшие детали реализаций. Вторая причина, по которой базовые методики так долго остаются в тени, заключается в том, что почти все они зародились как научные исследования и, следовательно, в значительной мере опираются на язык математики. С первой причиной ничего не поделаешь. Но объем общедоступных знаний так велик, что возникает вопрос: является ли непреодолимой вторая причина? Мой краткий ответ – громкое и решительное «Нет!». Чтобы получить развернутый ответ, вам придется прочесть эту книгу!

Я решил написать эту книгу с целью продемонстрировать, что некоторые из упомянутых методик можно представить в виде алгоритмов, не предполагая при этом наличия у читателя серьезной математической подготовки. Задача этой книги – дать вам методики, помогающие встроить в приложение интеллектуальное поведение с минимальным по возможности обращением к математическому аппарату. Все необходимые математические выкладки содержатся в программном коде – в виде алгоритма.

Сначала я хотел использовать для представления этих методик библиотеки с открытым исходным кодом. Но большинство таких библиотек разрабатывались конъюнктурно и нередко без малейшего намерения обучать положенным в их основу методикам. То есть, скорее всего, это

туманный код, который даже читать нелегко, не говоря уже о том, чтобы понимать! Я понял, что мой предполагаемый читатель больше выиграл бы от ясной, хорошо документированной программной базы. На этом распутье ко мне присоединился Дмитрий, и это он написал большую часть программного кода, который вы встретите в этой книге.

Количество книг, посвященных этой новой и захватывающей теме, будет медленно, но верно расти. Настоящая книга – только введение в область, которая уже обширна и продолжает быстро развиваться. Естественно, число рассматриваемых алгоритмов должно было быть ограниченным, а объяснения краткими. Моя задача сводилась к тому, чтобы отобрать несколько вопросов и дать им хорошее толкование, вместо того чтобы пытаться охватить как можно больше с риском запутать вас или просто создать рецептурный справочник.

Надеюсь, мы справились с поставленной задачей, придерживаясь следующих четыре правил:

- Работать с понятными примерами, сосредоточившись именно на них
- Использовать высокоуровневые сценарии, которые отражают такое применение алгоритмов, как если бы вы внедрили их в собственное приложение
- Помогать вам экспериментировать с программным кодом и осваивать его с помощью многочисленных заданий из разделов «сделать»
- Писать первоклассный удобочитаемый программный код

Итак, захватите свой любимый горячий напиток, сядьте поудобнее и опробуйте несколько умных приложений – они здесь, чтобы остаться!

Хараламбос Марманис

Благодарности

Мы хотели бы выразить признательность сотрудникам издательства Manning, предоставившим возможность опубликовать эту работу. Они не только помогли привести рукопись к ее окончательному виду, но и терпеливо дождались завершения работы над ней, для чего потребовалось много больше времени, чем мы планировали изначально. В частности, нам хотелось бы поблагодарить Марджана Бейса (Marjan Bace), Джеффа Блейела (Jeff Bleiel), Карен Тетмайер (Karen Tegtmeyer), Меган Йоки (Megan Yockey), Мери Пирджис (Mary Piergies), Морин Спенсер (Maureen Spencer), Стивена Хонга (Steven Hong), Рона Томича (Ron Tomich), Бенджамина Берга (Benjamin Berg), Элизабет Мартин (Elizabeth Martin), а также всех тех членов коллектива издательства Manning, кто работал над этой книгой, но чьих имен мы не знаем. Спасибо за ваш напряженный труд.

Нам также хотелось бы признать своевременными, действенными и ценными замечания рецензентов и посетителей нашего авторского форума Author Online. Ваши отклики помогли во многом улучшить эту книгу. Мы понимаем, как ограниченно и ценно «свободное» время каждого профессионала, поэтому очень благодарны за ваш вклад.

Мы выражаем особую благодарность за многократное прочтение нашей рукописи на разных стадиях ее написания и за предоставленные комментарии следующим рецензентам: Роберту Хэнсону (Robert Hanson), Самиту Пэлу (Sumit Pal), Карлтону Гибсону (Carlton Gibson), Дэвиду Хэнсону (David Hanson), Эрику Свенсону (Eric Swanson), Франку Вонгу (Frank Wang), Бобу Хатчисону (Bob Hutchison), Крейгу Уоллсу (Craig Walls), Николасу Хейнли (Nicholas C. Heinle), Владу Горски (Vlad Gorsky), Алессандро Галло (Alessandro Gallo), Крейгу Ланкастеру (Craig Lancaster), Джейсону Колтеру (Jason Kolter), Мартину Флетчеру (Martyn Fletcher) и Скотту Доусону (Scott Dawson). Последняя, но ничуть не меньшая благодарность в адрес Аджая Бхандари (Ajay Bhandari), который был корректором и последним вычитал главы и проверил программный код перед сдачей книги в печать.

Х. Марманис

Мне хотелось бы поблагодарить своих родителей, Еву и Александра. Это от них у меня неуемное любопытство и страсть к знанию, которые заставляют меня писать и заниматься исследованиями ночи напролет. Этот долг слишком велик, чтобы его можно было оплатить за одну жизнь.

Я от всего сердца благодарю мою дорогую жену Аврору и трех наших сыновей: Никоса, Лукаса и Альберта – величайшую гордость и радость моей жизни. Я всегда буду признателен за их любовь, терпение и понимание. Ежеминутное любопытство моих детей всегда вдохновляло меня на изыскания. Бесконечная благодарность родителям моей жены, Кучи и Хосе, моим сестрам Марии и Катерине, а также моим лучшим друзьям Майклу и Антонио за их постоянное ободрение и безоговорочную поддержку.

Не могу не отдать должное всесторонней поддержке докторов наук Амилькара Авенданьо (Amilcar Avendaco) и Марии Балерди (Maria Balerdi), которые многому научили меня в области кардиологии и на первых порах субсидировали мою исследовательскую работу. Я также выражаю свою благодарность профессору Леона Купера (Leon Cooper) и многим другим замечательным сотрудникам университета Брауна (Brown University), чей энтузиазм в изучении человеческого мозга вдохновил таких, как я, заняться интеллектуальными приложениями.

Моим коллегам в прошлом и в настоящем Аджаю Бхандари (Ajay Bhandari), Кавите Канеткар (Kavita Kanetkar), Александру Петрову (Alexander Petrov), Кишоре Кирдату (Kishore Kirdat) и многим другим, кто поощрял и поддерживал все мои начинания, связанные с искусственным интеллектом: эти скупые строки не могут выразить всю глубину моей благодарности вам.

Д. Бабенко

Прежде всего, хочу поблагодарить свою любимую жену Елену. На завершение работы над этой книгой ушло больше года, и моей жене пришлось смириться с тем, что муж все свое время отдавал службе или работе над книгой. Ее поддержка и ободрение создали идеальную среду для того, чтобы я мог написать эту книгу.

Мне хотелось бы поблагодарить всех моих бывших и нынешних коллег, которые повлияли на мою профессиональную жизнь, став ее вдохновителями: Константина Бобовича (Konstantin Bobovich), Пола А. Денниса (Paul A. Dennis), Кита Лоулесса (Keith Lawless) и Кевина Беделла (Kevin Bedell).

В заключение я также хотел бы поблагодарить своего соавтора доктора Марманиса за приглашение участвовать в этом проекте.

Об этой книге

Впечатление от современных веб-приложений в основном складывается за счет насыщенного возможностями пользовательского интерфейса (user interface, UI). Менее известный аспект современных приложений – применение методик, позволяющих осуществить интеллектуальную обработку информации и добавить полезные свойства, которые нельзя обеспечить другими средствами. Многочисленные примеры успешных проектов, основанных на применении таких методик, включают такие широко известные бренды, как Google, Netflix и Amazon. Эта книга о том, как построить алгоритмы, формирующие интеллектуальное ядро этих веб-приложений.

В книге рассматриваются пять важных категорий алгоритмов: поиск, выработка рекомендаций, создание групп, классификация и ансамбли классификаторов. По каждому из этих тематических разделов можно было бы написать отдельную книгу, и понятно, что целью данной книги не является исчерпывающее их рассмотрение. Эта книга – введение в основы пяти названных тематических разделов. Это попытка познакомить вас с базовыми алгоритмами интеллектуальных приложений, а не охватить сразу все алгоритмы вычислительного интеллекта. Книга адресована самой широкой аудитории и требует от читателя минимум предварительных знаний.

Отличительная особенность этой книги – специальный раздел в конце каждой главы. Мы назвали его «Сделать» (To Do), предназначив не только для дополнительного материала. Каждый из этих разделов позволит вам глубже вникнуть в тему соответствующей главы. Кроме того, разделы «Сделать» призваны посеять зерно любопытства, заставляющее вас задуматься о новых возможностях, равно как о связанных с ними проблемах, всплывающих в реальных приложениях.

В этой книге широко применяется библиотека сценариев BeanShell. Этот выбор служит достижению двух целей. Первая цель – обеспечить знакомство с алгоритмами на более легком для восприятия уровне, прежде чем углубляться в дебри подробностей. Вторая цель – наметить шаги, которые вам следует предпринять, чтобы включить эти алгоритмы в свое приложение. В большинстве случаев можно воспользоваться данной библиотекой, написав при этом лишь несколько строк кода! Более того, чтобы гарантировать сохранность и упростить сопровождение

исходного кода, мы создали на специальном сайте Google проект *http://code.google.com/p/yooreeka/*.

Структура книги

Книга состоит из семи глав. Глава 1 – введение. В главах 2–6 рассматриваются поиск, выработка рекомендаций, создание групп, классификация и ансамбли классификаторов соответственно. Глава 7 обобщает материал предыдущих глав, но в ней рассматривается новая тема в контексте одного приложения.

Хотя в главах встречаются ссылки на другие главы, материал организован так, что главы 1–5 можно прочесть независимо друг от друга. Глава 6 опирается на материал главы 5, так что ее трудно будет читать отдельно. Глава 7 тоже не является самостоятельной, поскольку в ней затрагивается материал всей книги.

В главе 1 приведен обзор интеллектуальных приложений и рассмотрены некоторые примеры их полезности. Дано практическое определение интеллектуальных веб-приложений, описан набор принципов проектирования. Перечислены шесть крупных классов веб-приложений, в которых можно задействовать интеллектуальные алгоритмы, рассматриваемые в этой книге. Также приводятся сведения о происхождении этих алгоритмов и рассматривается их связь с такими областями, как искусственный интеллект, машинное обучение, анализ данных и мягкие вычисления. В конце главы приведены восемь ошибок проектирования, часто встречающихся на практике.

Глава 2 начинается с описания процедуры поиска, основанного на традиционных методиках извлечения информации. Обобщив традиционный подход, мы перейдем к поиску вне индексов, в том числе к самому известному алгоритму оценки ссылок – PageRank. Один из разделов главы посвящен уточнению результатов поиска путем анализа экранных данных. Эта методика изучает предпочтения пользователя в отношении конкретного сайта или тематического раздела, и ее можно в значительной степени развить и дополнить, чтобы обеспечить новые возможности.

Также в этой главе описан поиск документов, не являющихся веб-страницами, с помощью нового алгоритма, который мы называем DocRank. Это довольно перспективный алгоритм, но важнее то, что он показал: математические принципы оценки ссылок несложно дополнить и, внеся необходимые изменения, применять в других контекстах. Также в этой главе говорится о некоторых проблемах, возникающих при работе в очень крупных сетях. В конце главы затрагивается проблема правдоподобия и контроля достоверности результатов поиска.

В главе 3 вводятся важнейшие понятия расстояния и сходства. В ней представлены две обширные категории методик выработки рекомен-

даций: коллаборативная фильтрация и подход на основе контента. В качестве контекста для выработки рекомендаций взят вымышленный онлайн-магазин музыкальных записей. Также здесь рассматриваются два примера более общего характера. Первый – гипотетический веб-сайт, который с помощью интерфейса Digg API извлекает контент пользователей, чтобы рекомендовать им статьи, которые они еще не видели. Второй пример, связанный с рекомендациями по кинофильмам, вводит понятие нормализации данных. В этой главе мы также оцениваем точность вырабатываемых рекомендаций исходя из среднеквадратической погрешности.

В главе 4 представлены алгоритмы кластеризации. Есть много прикладных областей, где можно применить кластеризацию. Теоретически, для кластеризации пригоден любой набор данных, объекты которого можно определить в терминах значений атрибутов. В этой главе мы рассматриваем создание групп сообщений на форуме и выявление пользователей веб-сайта, между которыми есть сходство. Здесь также предлагаются общий обзор типов кластеризации и полные реализации шести алгоритмов: одной связи, средней связи, минимального остовного дерева с одной связью, k-средних, ROCK и DBSCAN.

В главе 5 представлены алгоритмы классификации, которые являются важнейшими компонентами интеллектуальных приложений. В начале главы описаны онтологии, основанные на трех фундаментальных блоках – концептах, образцах и атрибутах. Классификация представлена как задача назначения «наилучшего» концепта данному образцу. Классификаторы отличаются друг от друга способом представления и способом оценки этого оптимального назначения. В этой главе дается обзор категорий классификации, который включает *двоичную* и *многоклассовую* классификации, *статистические* и *структурные* алгоритмы. Здесь также представлены три этапа жизненного цикла классификатора: обучение, тестирование и эксплуатация.

Изложение материала в главе 5 продолжает представлять на высоком уровне регрессионные алгоритмы, байесовские алгоритмы, алгоритмы на основе правил, функциональные алгоритмы, алгоритмы ближайших соседей и нейронные сети. Подробно обсуждаются три методики классификации. В основе первой методики – наивный байесовский алгоритм, применяемый к единственному строковому атрибуту. Вторая методика основана на процессоре правил Drools, который является объектно-ориентированной реализацией алгоритма Rete и позволяет объявлять и применять правила в целях классификации. Третья методика представляет и применяет *вычислительные нейронные сети*; в этой главе предлагается элементарная, но надежная реализация для построения нейронных сетей общего назначения. Глава 5 также предупреждает вас о проблемах, связанных с достоверностью и вычислительными требованиями классификации, которые необходимо решить, прежде чем внедрять возможности классификации в приложения.

В главе 6 рассматриваются ансамбли классификаторов – модифицированные методики, способные повысить точность классификации, обеспечиваемую одиночным классификатором. Основной пример, который рассматривается в этой главе, – оценка кредитоспособности для приложения, обслуживающего ипотечное кредитование. Подробно представлены метод формирования ансамблей классификаторов (bagging) и метод улучшения слабых моделей (boosting). Также в этой главе представлена реализация алгоритма улучшения Бреймана arc-x4.

В главе 7 демонстрируется применение интеллектуальных алгоритмов в контексте новостного портала. Мы обсуждаем технические проблемы наряду с новыми, ценными с точки зрения бизнеса возможностями, которые интеллектуальные алгоритмы могут добавить в приложение. Например, алгоритм кластеризации можно использовать для объединения в группы похожих новостных сообщений, а также для расширения области видимости релевантных новостных сообщений с помощью *перекрестных ссылок*. В этой главе мы в общих чертах описываем адаптацию различных интеллектуальных алгоритмов и их объединение для достижения конкретной цели.

Специальный раздел «Сделать»

В каждой главе, начиная со второй, есть раздел с несколькими заданиями для выполнения, которые послужат вам руководством при изучении различных тем. Нам как разработчикам программного обеспечения нравится название «Сделать» («ToDo»): подразумевая обязательность выполнения, оно менее формально, чем такие названия, как «Упражнения» («Exercises»).

Одни задания разделов «Сделать» нацелены на более глубокое освоение темы данной главы, другие предоставляют отправную точку для изучения вопросов, расширяющих эту тему. Выполнив эти задания полностью, вы получите более глубокое и полное представление об интеллектуальных алгоритмах.

Везде, где это уместно, наш программный код снабжен тегами `TODO`, которые должны быть видны в интегрированных средах разработки (Integrated Development Environment, IDE), например в среде Eclipse, если щелкнуть мышью на панели Tasks (Задачи). Если щелкнуть мышью на любом из этих заданий, соответствующая гиперссылка позволит увидеть фрагмент программного кода, связанный с этим заданием.

Кому адресована эта книга

Книга «Алгоритмы интеллектуального Интернета» написана для разработчиков программного обеспечения и веб-разработчиков, которым хотелось бы подробнее ознакомиться с новой разновидностью алгоритмов, обеспечивающих интеллектуальную поддержку многих коммерчески успешных приложений. Поскольку исходный программный код

написан на языке Java, работающим с этим языком данная книга может показаться более привлекательной, чем тем, кто использует другие языки программирования. Тем не менее пишущим на других языках вполне по силам изучить приведенный в этой книге программный код и, возможно, перевести его на предпочитаемый ими язык.

Книга полна примеров и идей, которые могут найти широкое применение, следовательно, она также может быть полезна техническим директорам, менеджерам по продукции и другим руководителям, желающим лучше разобраться в соответствующих технологиях и предлагаемых возможностях с точки зрения бизнеса.

Наконец, несмотря на слово *Интернет* в заглавии книги, ее материал в равной степени применим к самым разным программным приложениям, от утилит мобильной связи до традиционных настольных приложений, таких как текстовые редакторы и электронные таблицы.

Соглашения о примерах кода

Весь программный код в книге набран моноширинным шрифтом, выделяющим код из окружающего текста. Почти каждый пример программного кода снабжен комментариями, выделяющими ключевые понятия, а иногда и нумерованным списком в тексте, содержащим дополнительную информацию по этим коментариям. Те строки, которые не поместились по ширине, включают символ переноса на следующую строку.

Исходный программный код для этой книги доступен по адресу *http://code.google.com/p/yooreeka/downloads/list* или на сайте издательства *http://www.manning.com/marmanis/*.

Файл дистрибутива следует распаковать прямо на диск C:\. Мы предполагаем, что вы используете операционную систему Microsoft Windows, в противном случае вам придется модифицировать наши сценарии, чтобы заставить их работать в вашей системе. Каталог верхнего уровня архивного файла называется *iWeb2*: все ссылки на каталоги даются в книге с учетом этой корневой папки. Например, ссылка на каталог *data/ch02*, согласно нашей договоренности, означает абсолютный путь к каталогу *C:\iWeb2\data\ch02*.

Если указанный файл распакован, вы готовы выполнить сценарий создания сборки Ant. Просто перейдите в каталог сборки и выполните команду `ant`. Обратите внимание: сценарий Ant будет работать независимо от того, куда вы распаковали файл. Теперь вы готовы к тому, чтобы выполнить сценарий BeanShell, как описано в приложении A.

Авторы в Интернете

Издательством Manning Publications организован веб-форум Author Online, где можно оставить свой комментарий об этой книге, задать технические вопросы и получить помощь авторов и других пользователей. Чтобы подписаться на участие в нем, перейдите по ссылке *http://*

www.manning.com/marmanis/. На этой странице есть информация о том, какого рода помощь доступна, как зарегистрироваться, а также о правилах поведения на форуме. Кроме того, на этой странице представлены ссылки на исходный код рассматриваемых в книге примеров, список замеченных опечаток и другие материалы для скачивания.

Издательство Manning предоставляет читателям место для встреч, где может иметь место целевой диалог между отдельными читателями, а также между читателями и авторами. При этом оно не обязано обеспечивать определенный объем участия со стороны авторов, чей вклад в работу форума Author Online является добровольным (и бесплатным). Чтобы поддерживать заинтересованность авторов, советуем вам стараться задавать им разные стимулирующие вопросы!

Форум Author Online и архивы предыдущих обсуждений будут доступны на веб-сайте издательства до тех пор, пока книга находится в продаже.

1

Что такое интеллектуальный Интернет?

- Преимущества интеллектуальных веб-приложений
- Применение веб-приложений в реальном мире
- Встраивание интеллекта в веб-приложения

Итак, о чем эта книга? Прежде всего, давайте скажем, о чем в ней не говорится. Эта книга не о построении безупречного пользовательского интерфейса и не об использовании формата обмена данными JSON или языка запросов XPath, и даже не об архитектуре служб RESTful. Есть хорошие книги о приложениях, поддерживающих стандарт Web 2.0, где описана реализация проектов на основе технологии AJAX и общая практика применения развитых пользовательских интерфейсов. Немало и книг, посвященных другим эффективным веб-технологиям, таким как язык преобразований XSL Transformations (XSLT) и язык запросов XPath, масштабируемая векторная графика (Scalable Vector Graphics, SVG), технология веб-форм XForms, декларативный язык описания интерфейсов XUL и формат обмена данными JSON (JavaScript Object Notation).

Отправной точкой для этой книги послужил факт «тупости» большинства традиционных веб-приложений в том смысле, что отклик системы не учитывает введенные ранее данные и прежнее поведение пользователя. Мы говорим не о проблемах неудобного пользовательского интерфейса, а о том, что на ввод конкретных данных система всегда реагирует одинаково. Наш главный интерес – это построение веб-приложений,

которые учитывают введенные каждым пользователем данные, его поведение в системе на протяжении некоторого периода времени, а также другую потенциально полезную информацию, которая может быть доступна.

Например, вы начинаете заказывать еду с помощью веб-приложения и каждую среду заказываете рыбу. Вам было бы гораздо приятнее, чтобы по средам это приложение спрашивало: «Не хотите ли сегодня рыбу?», а не интересовалось: «Что бы вы хотели заказать сегодня?» В первом случае приложение *каким-то образом смогло понять*, что по средам вы предпочитаете рыбу. Во втором случае приложение пребывает в неведении относительно этого факта. То есть данные, созданные в результате вашего взаимодействия с сайтом, не влияют на то, как приложение выбирает контент страницы, или на то, как этот контент отображается. Обращение с вопросом, сформулированным на основе ранее выбранных пользователем вариантов, представляет собой новый вид взаимодействия между веб-сайтом и его пользователями. И мы могли бы назвать подобное свойство веб-сайтов *способностью к обучению*.

Следующим шагом в этом направлении будет корректировка взаимодействия интеллектуального веб-приложения с пользователем в зависимости от входных данных, полученных от других пользователей, так или иначе связанных друг с другом. Если ваш обычный рацион во многом совпадает с рационом Джона, приложение может рекомендовать вам несколько пунктов меню, обычных для Джона, которые вы никогда не заказывали; выработка рекомендаций рассматривается в главе 3.

Другой пример – сайт социальной сети, такой как Facebook, который предлагал бы чат или электронную конференцию (форум) с проверкой фактов. Под *проверкой фактов* (fact checking) мы подразумеваем следующее: по мере ввода вами текста сообщения то, что вы написали, в фоновом режиме подвергается проверке, чтобы удостовериться в точности утверждаемых вами фактов и даже в том, что они не противоречат вашим предшествующим сообщениям. Такая функциональность сходна с проверкой орфографии, вероятно, уже знакомой вам, но вместо того чтобы проверять правила грамматики, в данном случае проверяется набор фактов, который мог бы включать общеизвестные истины («Япония вторглась в Манчжурию в 1931 году»), ваши собственные представления о конкретном предмете («снижение налогов – благо для экономики») или простые факты частной жизни («посещение врача 11/11/2008»). Веб-сайты с таким функциональным поведением способны *делать логические выводы*; организацию соответствующей функциональности мы описываем в главе 5.

Можно утверждать, что по-настоящему эра интеллектуальных веб-приложений началась с появлением в Интернете поисковых систем, таких как Google. Разумеется, вы можете спросить: почему именно Google? Как решаются задачи, связанные с извлечением информации (поиском), было известно задолго до появления Google на мировой сцене.

Но исключительно важно то, что поисковые системы вроде Google плодотворно используют взаимосвязанность веб-контента. У Google был следующий тезис: на веб-страницах гиперссылки формируют базовую структуру, которую можно анализировать, чтобы определить важность различных страниц. В главе 2 мы подробно рассматриваем алгоритм PageRank, который обеспечивает такую возможность.

Расширяя обсуждаемую тему, можно сказать, что интеллектуальные веб-приложения изначально разрабатываются с прицелом на взаимодействующий и взаимосвязанный мир. Эти приложения проектируются для автоматического обучения, так что могут понимать вводимые пользователем данные или его поведение, либо и то и другое, и соответственно настраивать свой отклик. Совместное использование пользовательских профилей коллегами, друзьями и членами семьи на сайтах социальных сетей, таких как MySpace или Facebook, как и общий доступ к контенту телеконференций и онлайн-форумов наряду с высказываемыми там мнениями, создает новые уровни связанности, которые являются ключевыми для интеллектуальных веб-приложений и выходят за рамки обычных гиперссылок.

1.1. Примеры интеллектуальных веб-приложений

Рассмотрим приложения, в которых на протяжении последнего десятилетия применяется такого рода интеллект. Как уже говорилось, поворотным моментом в истории Интернета стало появление поисковых систем. Все то, что предлагалось в Интернете, в основном не менялось вплоть до 1998 года, когда в контексте поиска появилась и штурмом захватила рынок методика *оценки ссылок* (см. главу 2). Меньше чем за 10 лет компания Google Inc. из начинающего выросла в доминирующего игрока в этом технологическом секторе, благодаря, во-первых, успеху используемой ею технологии поиска на основе ссылок и, во-вторых, набору других предоставляемых ею услуг, таких как Google News (Новости Google) и Google Finance (Финансы Google).

Однако область применения интеллектуальных веб-приложений – это далеко не одни только поисковые системы. Розничный онлайн-торговец Amazon стал одним из первых интернет-магазинов, предлагающих своим пользователям рекомендации на основе образцов сделанных ими покупок. Вам, скорее всего, знакома эта функциональная возможность. Предположим, вы покупаете книгу о JavaServer Faces и книгу о языке программирования Python. Как только товары добавлены в корзину, на сайте Amazon вам порекомендуют дополнительные товары, так или иначе связанные с тем, что вы только что выбрали, например книги, посвященные технологиям AJAX или Ruby on Rails. Кроме того, при следующем вашем посещении веб-сайта Amazon вам могут порекомендовать эти же или другие связанные товары.

Еще одно интеллектуальное веб-приложение – Netflix[1], крупнейшая в мире интернет-служба видеопроката, где подписчикам (а их больше 7 000 000) предлагается 90 000 наименований DVD-дисков и пополняющаяся библиотека, которая предоставляет для онлайн-просмотра на пользовательских компьютерах больше 5000 полнометражных фильмов и телепрограмм. Пять лет подряд, с 2005 по 2007 годы, веб-сайт Netflix возглавлял рейтинг согласно опросу об удовлетворенности клиентов, проводимому раз в полгода исследовательскими компаниями ForeSee Results и FGI Research.

Отчасти успех этого сайта в Интернете – следствие того, что он предлагает пользователям легкий способ отбора фильмов из громадного списка. Ядро этой функциональной возможности – система выработки рекомендаций Cinematch. Ее задача – предсказать, понравится ли данный фильм конкретному пользователю, исходя из того, насколько ему нравятся или нет другие фильмы. Эта система – еще один замечательный пример интеллектуального веб-приложения. Предсказательные способности системы Cinematch настолько важны для Netflix, что в конце концов в октябре 2006 года был объявлен приз[2] в миллион долларов за усовершенствование возможностей данной системы. К октябрю 2007 года насчитывалось 28 845 конкурентов из 165 стран. В главе 3 мы предлагаем всестороннее рассмотрение алгоритмов, необходимых для создания системы рекомендаций, подобной Cinematch.

Интеллектуальные предсказания на основе мнений членов коллектива не ограничиваются рекомендациями по книгам или фильмам. Фирма PredictWallStreet собирает прогнозы своих пользователей относительно конкретных акций или индексов, чтобы выявить тенденции в оценках трейдеров и предсказать объем базовых активов. Мы не советуем вам снять со счета сбережения и заняться торговлей на бирже, полагаясь на их предсказания, но это – еще один пример творчески примененных методик, которые рассматриваются в этой книге, в сценарии из реальной жизни.

1.2. Базовые элементы интеллектуальных приложений

Давайте пристальнее вглядимся в то, какие отличительные особенности делают интеллектуальными приложения, рассмотренные нами в предыдущем разделе, и в частности подчеркнем различие между координацией совместной деятельности и интеллектом. Рассмотрим случай веб-сайта, где пользователи могут коллективно составлять документ. Такой веб-сайт вполне можно было бы квалифицировать как развитое веб-приложение согласно набору определений термина «раз-

---

[1] Источник: Netflix, Inc., *http://www.netflix.com/MediaCenter?id=5379*.

[2] Источник: *http://www.netflixprize.com//rules*.

витой» (advanced). Это приложение, разумеется, способствовало бы совместной деятельности многих пользователей в режиме онлайн, и оно могло бы предложить изобилующий возможностями, не вызывающий затруднений пользовательский интерфейс, гладкий технологический процесс и так далее. Но следует ли рассматривать такое приложение как интеллектуальное веб-приложение?

Документ, создаваемый на рассматриваемом веб-сайте, будет иметь бóльший объем, он будет глубже проработан и, наверное, более точен, чем другие документы, написанные каждым участником в отдельности. В этом отношении такой документ фиксирует не только знания каждого отдельного члена коллектива, но также результаты влияния взаимодействующих пользователей на конечный продукт. Следовательно, созданный таким способом документ фиксирует коллективное знание его создателей.

Идея не нова. Процессом определения стандарта в любой области науки или инженерии обычно управляет технический комитет. Этот комитет создает первый рабочий вариант документа, соединяющий знания экспертов и мнения нескольких заинтересованных групп и отвечающий потребностям коллектива, а не конкретного индивидуума или производителя. Затем первый рабочий вариант становится общедоступным для комментирования. В результате этого процесса окончательный вариант документа должен представить всю совокупность знаний сообщества и сформулировать рекомендации по отдельным требованиям, выявленным в этом сообществе.

Вернемся к нашему приложению. Как было определено до сих пор, это приложение позволяет зафиксировать коллективное знание как результат коллективных усилий, но пока еще не является интеллектуальным. Коллективный интеллект – термин весьма популярный, но часто понимаемый неверно, – требует коллективного знания и создается в результате коллективных воздействий, однако эти условия являются хоть и необходимыми, но не достаточными для того, чтобы характеризовать базовую программную систему как интеллектуальную.

Чтобы понять, из каких обязательных составляющих складывается то, что мы понимаем под *интеллектом* (intelligence), допустим еще, что у нашего воображаемого веб-сайта есть и другие возможности: по мере того как пользователь вводит свою статью документа, система идентифицирует другие документы, которые могут оказаться релевантными вводимому контенту, извлекает из них цитаты и помещает эти цитаты на боковую панель. Это могут быть документы из личной коллекции пользователя, а также совместно используемые участниками текущей работы или просто открытые, свободно доступные документы.

Пользователь может отметить фрагмент разрабатываемого документа и предписать системе уведомлять его, когда она обнаружит в сети документы, имеющие отношение к содержимому данного отрывка или, что, пожалуй, интереснее, когда консенсус сообщества, достигнутый

относительно этого содержимого, изменится в соответствии с некими критериями, заданными пользователем.

Для создания приложения с такими возможностями требуется много больше, чем привлекательный пользовательский интерфейс и платформа для совместных действий. Для этого требуется понимать произвольный текст. Нужна способность, позволяющая распознавать смысл сказанного в некотором контексте. Необходимо иметь возможность автоматически обрабатывать и группировать документы или фрагменты документов, которые содержат произвольный текст на естественном (человеческом) языке, по принципу их «похожести». Требуется некое структурированное знание о мире или, по меньшей мере, о предметной области рассуждения, к которой относится данный документ. Необходима способность сосредоточиться на конкретных документах, удовлетворяющих определенным правилам (пользовательским критериям), и сделать это быстро.

Таким образом, мы приходим к заключению, что приложения вроде Википедии и других общественных порталов отличаются от приложений вроде поисковой системы Google, рекламных программ Google Ads, программы выработки рекомендаций Netflix Cinematch и так далее. Приложения первого рода – это платформы для совместных действий, которые способствуют накоплению и обслуживанию коллективного знания. Приложения второго рода выделяют абстрактные структуры из всей совокупности коллективного знания и, следовательно, генерируют новый уровень возможностей и ценности. Мы завершаем этот раздел, суммируя элементы, необходимые для создания интеллектуального веб-приложения:

- *Агрегированный контент* (*aggregated content*) – другими словами, большой объем данных, относящихся к конкретному приложению. Агрегированный контент является скорее динамическим, чем статическим, и его источники, как и места его хранения, могут быть рассредоточены географически. Каждый фрагмент информации обычно связан или ссылается на множество других фрагментов информации.
- *Ссылочные структуры* (*reference structures*) – эти структуры обеспечивают одну или несколько структурных и семантических интерпретаций контента. Например, это относится к тому, что называют *фолксономией* (*folksonomy*), то есть к использованию тегов для динамического аннотирования контента и постоянного обновления представления коллективного знания для пользователей. Ссылочные структуры о мире или о конкретной предметной области знаний принадлежат одному из трех крупных классов: словари, базы знаний и онтологии (см. ссылки по теме в конце главы).
- *Алгоритмы* (*algorithms*) – этот элемент относится к слою модулей, позволяющих приложению задействовать информацию, скрытую

в данных, и использовать ее в целях абстракции (обобщения), предсказания и (со временем) улучшенного взаимодействия с их пользователями. Алгоритмы применяются к агрегированному контенту и иногда требуют присутствия ссылочных структур.

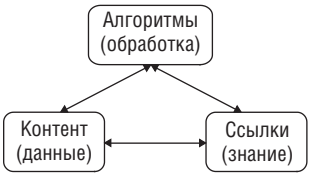

Рис. 1.1. *Треугольник интеллекта: три неотъемлемые составные части интеллектуальных приложений*

Наличие этих составляющих, сведенных вместе на рис. 1.1, обязательно для того, чтобы можно было определить приложение как интеллектуальное веб-приложение, и мы будем ссылаться на них на протяжении всей этой книги как на *треугольник интеллекта* (triangle of intelligence).

Целесообразно не смешивать эти три компонента друг с другом и построить такую модель их взаимодействия, которая в наибольшей степени отвечает вашим потребностям. Более подробно о разработке архитектуры мы поговорим в других главах, особенно в главе 7.

1.3. Что могут выиграть приложения от интеллектуальности?

Составляющие интеллекта, как показано в предыдущем разделе, можно обнаружить в широком диапазоне приложений: от сайтов социальных сетей до специализированных приложений для борьбы с терроризмом. В этом разделе мы рассмотрим примеры приложений из каждой категории. Наш список, разумеется, не полон, но он продемонстрирует, что рассматриваемые в этой книге методики во многих областях могут быть полезными, а в некоторых случаях и незаменимыми.

1.3.1. Сайты социальных сетей

За последние несколько лет наиболее заметное влияние на Интернет оказали веб-сайты социальных сетей. Такие сайты являются веб-приложениями, предоставляющими своим пользователям возможность установить факт своего присутствия в сети, не пользуясь для этого никакими иными средствами, кроме броузера и подключения к Интернету. Пользователи могут обмениваться друг с другом файлами (презентациями, видеофайлами, аудиофайлами), комментировать текущие события или страницы других пользователей, создать собственную со-

циальную сеть или присоединиться к уже имеющейся исходя из своих интересов. Два наиболее посещаемых[1] сайта социальных сетей – это MySpace и Facebook с сотнями миллионов и десятками миллионов зарегистрированных пользователей, соответственно.

По своему устройству эти сайты являются агрегаторами контента, так что первая составляющая для создания интеллекта легко доступна. Вторая составляющая также присутствует на этих сайтах. Например, на сайте MySpace контент распределяется по категориям с помощью основных ярлыков, таких как Books (Книги), Movies (Фильмы), Schools (Школы), Jobs (Вакансии) и так далее, которые отчетливо видны на странице сайта (рис. 1.2).

Рис. 1.2. *Категории, применяемые на веб-сайте MySpace*

Кроме того, эти категории верхнего уровня подвергаются дальнейшей детализации с помощью структур нижнего уровня, которые отделяют контент, относящийся к группе Classifieds (Объявления), от контента, относящегося к группам Polls (Опросы) или Weather (Погода). Наконец, большинство сайтов социальных сетей в состоянии порекомендовать своим пользователям новых друзей и новые сообщения (постинги, postings), которые могли бы представлять для них интерес. В этом они опираются на сложные алгоритмы создания предсказаний и абстракции накопленных данных и, следовательно, содержат все три составные части интеллекта.

[1] На основании данных о трафике, собранных службой Alexa.com в декабре 2007 года.

1.3.2. Гибридные веб-приложения (мэшапы)

На сайте DeveloperWorks фирмы IBM (*http://www.ibm.com/developerworks/spaces/mashups*) есть целый раздел, посвященный *гибридным веб-приложениям, или мэшапам* (mashups), с чрезвычайно подходящим определением: «Гибридные веб-приложения – это впечатляющий класс интерактивных приложений, которые извлекают контент из внешних источников данных, чтобы создать совершенно новые и инновационные службы». Другими словами, вы создаете сайт, используя контент и элементы пользовательского интерфейса, «заимствованные» у других. Еще один интересный сайт в контексте таких веб-приложений – это сайт ProgrammableWeb (*http://www.programmableweb.com*). Подходящее место для того, чтобы начать исследование мира гибридных приложений (рис. 1.3).

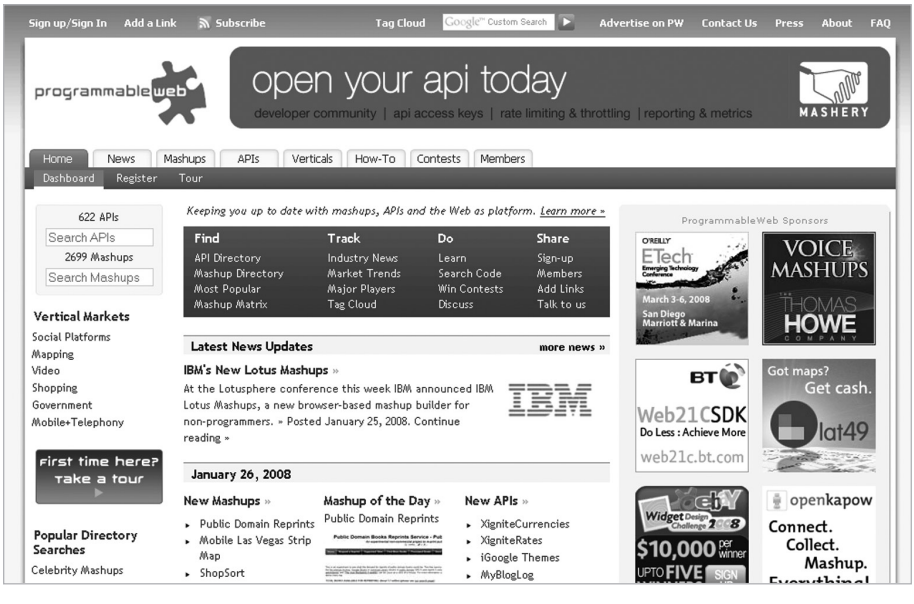

Рис. 1.3. *Чтобы подробнее изучить гибридные веб-приложения, посетите такие сайты, как ProgrammableWeb*

В нашем контексте гибридные веб-приложения важны, потому что основой для них является агрегированный контент; но, в отличие от сайтов социальных сетей, эти приложения не владеют тем контентом, который демонстрируют на экране, – по крайней мере, большая его часть принадлежит не им. Этот контент физически хранится в географически рассредоточенных пунктах и собирается воедино из разнообразных источников для создания уникальной презентации на основе вашего взаимодействия с приложением.

Однако не все гибридные веб-приложения являются интеллектуальными. Чтобы создать интеллектуальные гибридные веб-приложения, необходимо иметь возможность устранить разногласия или выявить сходные элементы контента, который мы пытаемся объединить.

В свою очередь, чтобы согласовать и классифицировать контент, нужно иметь одну или несколько ссылочных структур для интерпретации его смысла, а также ряд алгоритмов, которые позволяют установить, какие элементы этих ссылочных структур содержатся в различных фрагментах, или помогают понять, к какой категории следует отнести извлеченный из разных сайтов контент, чтобы обеспечить возможность его просмотра.

1.3.3. Порталы

Порталы (portal) и, в частности, новостные порталы – это еще один класс веб-приложений, для которых рассматриваемые в этой книге методики могут быть очень важны. По определению, такие приложения являются шлюзами к контенту, который разбросан по Интернету или, если речь идет о корпоративной локальной сети, по интранету. Это еще один случай, когда агрегированный контент рассредоточен, но доступен.

Лучший пример из данной категории – новостной портал Google News (*http://news.google.com*). Этот сайт собирает новостные сообщения из тысяч источников и автоматически группирует похожие сообщения под общим заголовком. Более того, каждая группа новостных сообщений отнесена к одной из новостных категорий, доступных по умолчанию, например Business (Бизнес), Health (Здоровье), World (Мир), Sci/Tech (Наука и техника) и так далее (рис. 1.4).

Вы можете создавать и собственные категории, определив, какого рода новости представляют для вас интерес. Опять-таки, мы видим, что подоплекой является агрегированный контент в связке со ссылочной структурой и рядом алгоритмов, которые могут выполнить требуемые задачи автоматически или, по меньшей мере, в полуавтоматическом режиме.

Многообещающим, с точки зрения встраивания интеллекта в ваш собственный портал, особенно из разряда социальных приложений, является проект OpenSocial (*http://code.go7ogle.com/apis/opensocial/*), а также несколько проектов, развивающихся вокруг него, например проект контейнера Apache Shindig. Исходная предпосылка проекта OpenSocial – создание общей базы интерфейса API, которая позволит разрабатывать приложения, взаимодействующие с большим и постоянно растущим числом веб-сайтов, таких как Engage, Friendster, hi5, Hyves, imeem, LinkedIn, MySpace, Ning, Oracle, orkut, Plaxo, Salesforce, Six Apart, Tianji, Viadeo и XING.

Рис. 1.4. *Веб-сайт Google News – интеллектуальное приложение портала*

1.3.4. Вики-сайты

Википедия (Wikipedia) не требует особого представления: вы, вероятно, уже посещали этот сайт или, по крайней мере, слышали о нем. Это вики-сайт, который не покидает десятку наиболее посещаемых ресурсов. *Вики* (wiki) – это доступное в компьютерной сети хранилище знаний. Вики-сайты используются социальными сообществами в Интернете и внутри корпораций в целях обеспечения общего доступа к знаниям.

Эти сайты, безусловно, являются агрегаторами контента. Кроме того, множество таких ресурсов, благодаря технологическому процессу создания страниц, имеют встроенную структуру, которая аннотирует контент. В Википедии можно отнести статью к какой-то *категории* (category) и связать с ней статьи, относящиеся к этой же тематике. Вики – многообещающая область для применения методик, рассматриваемых в этой книге. Например, вы могли бы создать или модифицировать свой вики-сайт таким образом, чтобы он автоматически распределял по категориям написанные вами страницы. Вики-страницы могли бы иметь панель-врезку (inlet) или другую панель с рекомендованными терминами, к которым вы можете осуществить привязку, – предполагается, что страницы на вики-сайте всегда связаны друг с другом, когда связь предоставляет пояснение или дополнительную информацию для некоторого термина или по какой-то теме. Наконец, естественная связь

страниц обеспечивает плодородную почву для поиска с применением усложненных запросов (глава 2), кластеризации (глава 4) и других аналитических методик.

1.3.5. Сайты общего доступа к медиафайлам

YouTube – фирменная марка веб-сайтов, обеспечивающих общий доступ к медиафайлам, но и другие веб-сайты, такие как RapidShare (*http://www.rapidshare.com*) и MegaUpload (*http://www.megaupload.com/*), не страдают от недостатка посетителей. Уникальной характеристикой этих сайтов является то, что большая часть их контента представлена в двоичном формате, – это видео- или аудиофайлы. В большинстве случаев размер наименьшей информационной единицы на этих сайтах превышает размеры, характерные для сайтов-агрегаторов, обрабатывающих текстовые данные; только лишь объем данных, которые необходимо обработать, составляет одну из самых больших проблем в контексте накопления интеллекта.

Кроме того, две из наиболее трудных (а также наиболее интересных с точки зрения бизнеса) задач интеллектуальных приложений тесно связаны с обработкой двоичных данных. Эти две задачи – распознавание речи и распознавание образов. Такие фирмы, как Clearspring (*http://www.clearspring.com/*) и ScanScout (*http://www.scanscout.com/*), работая совместно, позволяют рекламодателю шире распространить свой бренд и информировать о нем более широкую аудиторию. Фирма ScanScout предоставляет рекламодателям сведения о распределении по сайтам их виджетов и о взаимодействии с ними; список таких сайтов включает MySpace, Facebook, Google и Yahoo, а всего их насчитывается больше 25!

Ту же модель, которую мы описали в предыдущих разделах, можно обнаружить и на этих сайтах. У нас есть агрегированный контент; как правило, мы хотим иметь контент, распределенный по категориям; и нам нужны алгоритмы, которые могут способствовать извлечению смысла из этого контента. Хотелось бы, чтобы наши двоичные файлы были распределены по категориям исходя из определенных нами тем – Autos & Vehicles (Автомобили и транспортные средства), Education (Образование), Entertainment (Развлечения), Politics (Политика) и так далее (рис. 1.5).

Аналогично другим случаям интеллектуальных приложений, эти категории могут иметь иерархическую структуру. Например, категория Autos & Vehicles может быть далее разбита на подкатегории, такие как Sedan (Седан), Trucks (Грузовики), Luxury (Люкс), SUV (Внедорожники) и так далее.

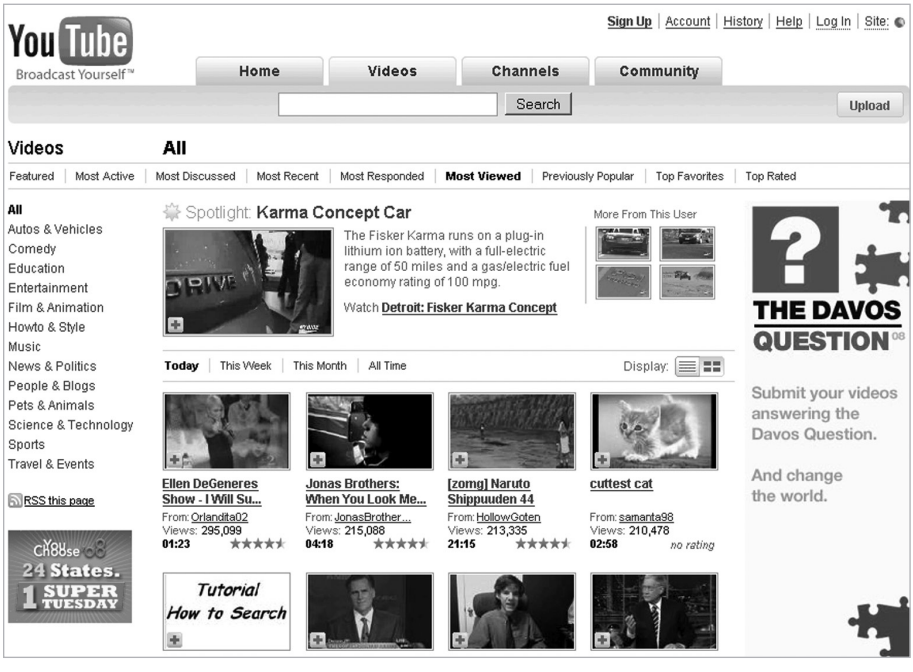

Рис. 1.5. *Категории видеозаписей на веб-сайте YouTube. Структура ссылок для классификации контента показана на панели слева*

1.3.6. Онлайн-игры

Захватывающие онлайн-игры с множеством игроков обладают всеми составляющими, необходимыми для того, чтобы встроить в игру интеллект. У них хватает агрегированного контента и ссылочных структур, которые отражают правила, и они, естественно, могут использовать алгоритмы, рассматриваемые нами в этой книге, чтобы ввести в игру новые уровни сложности. Персонажи, за которых играет компьютер, могут усваивать данные, вводимые игроками-людьми, так что игра станет более занимательной для человека.

Игры в сети – очень интересная область для применения интеллектуальных методик, и она может стать ключевым элементом для дифференциации конкурентов, поскольку с учетом сложности игр и инноваций возрастает уровень доступных для игры вычислительных мощностей и ожиданий игроков-людей. Методики, которые мы рассматриваем в главах 4, 5 и 6, как и большая часть материала приложений, находят прямое применение в онлайн-играх.

1.4. Как встроить интеллект в мое приложение?

Мы привели много причин для встраивания интеллекта в ваше приложение. Мы также рассмотрели ряд областей, где интеллектуальное поведение вашего программного обеспечения может радикально улучшить практические и стоимостные результаты, которые ваше приложение обеспечивает пользователям. На этой стадии естественным становится вопрос: «Как встроить интеллект в мое приложение?»

Вся эта книга – введение в проектирование и реализацию интеллектуальных компонентов, но чтобы использовать ее с наибольшей пользой, вам также следует обеспечить две предпосылки создания интеллектуального приложения.

Первая предпосылка – это ревизия функциональности вашего приложения. Что с ним делают пользователи? Как ваше приложение повышает потребительскую ценность или ценность бизнеса? Мы приводим несколько конкретных вопросов, относящихся в основном к алгоритмам, которые будут прорабатываться в оставшейся части этой книги. Важность вопросов зависит от того, что делает ваше приложение. Тем не менее эти конкретные вопросы должны помочь вам выявить те области, где интеллектуальный компонент добавил бы больше всего ценности в ваше приложение.

Вторая предпосылка касается данных. Для каждого приложения данные являются или внутренними (доступны непосредственно в приложении), или внешними, по отношению к этому приложению. Прежде всего, проанализируйте внутренние данные. Возможно, у вас есть все, что нужно, в таком случае вы готовы к действиям. И наоборот, возможно, вам необходимо организовать технологический процесс или добавить другие средства для получения от пользователей каких-то дополнительных данных. Например, вы хотите добавить на свои страницы элемент пользовательского интерфейса для классификации по принципу «пять звезд», с тем чтобы можно было создать механизм выработки рекомендаций, основываясь на классификации, которую обеспечат пользователи.

Или вам, к примеру, захотелось или понадобилось получать дополнительные данные из внешних источников. Есть много вариантов достижения этой цели. Мы не можем рассмотреть здесь все возможности, но определим четыре крупные категории, достаточно надежные технологически и популярные. Чтобы получить необходимые дополнительные сведения о конкретном выбранном методе, вам придется заглянуть в соответствующую литературу.

1.4.1. Анализ функциональности и данных

Вам следовало бы начать с выявления ряда вариантов использования (use cases), которые выиграли бы от интеллектуального поведения. Эти

варианты, разумеется, отличались бы от приложения к приложению, но вы можете выявить такие случаи, ответив на несколько очень простых вопросов:

- Обслуживает ли приложение контент, собранный из разных источников?
- Есть ли в приложении технологические процессы, реализованные с помощью «мастеров» (wizard)?
- Имеет ли приложение дело с текстом на естественном языке?
- Включает ли приложение создание отчетов какого-либо рода?
- Имеет ли приложение дело с географическими объектами, такими как карты?
- Предоставляет ли приложение функциональные возможности поиска?
- Используют ли ваши пользователи контент совместно?
- Важно ли для приложения обнаружение мошенничества?
- Важна ли для приложения верификация идентичности?
- Принимает ли приложение решения автоматически, основываясь на правилах?

Этот список, разумеется, не полон, но он указывает на имеющиеся возможности. Если на любой из этих вопросов дается ответ «да», ваше приложение может существенно выиграть от методик, которые мы будем рассматривать в оставшейся части этой книги.

Рассмотрим стандартный вариант использования поиска по данным вымышленного приложения. Почти все приложения предоставляют своим пользователям возможность осуществить поиск по сайту. Пусть, скажем, наше воображаемое приложение позволяет своим пользователям приобретать разные товары по каталогу. Пользователи могут выполнять поиск товаров, которые они хотят купить. Обычно такая функциональность реализуется с помощью прямого запроса на языке SQL, который обеспечит извлечение всех товарных позиций, соответствующих описанию товара. Это хорошо, но наш сервер базы данных не учитывает тот факт, что запрос может быть выполнен конкретным пользователем, о котором нам, скорее всего, многое известно в контексте его поиска. Мы наверняка сможем повысить эффективность взаимодействия пользователя с приложением, если реализуем методы ранжирования, описанные в главе 2, или методы выработки рекомендаций, описанные в главе 3.

1.4.2. Получение дополнительных данных из Интернета

Часто вам будет достаточно собственных данных для построения интеллекта, это важно и ценно для вашего приложения. Но в некоторых

случаях обеспечение интеллекта в приложении может потребовать доступа к внешней информации. На рис. 1.6 показана страница гибридного веб-сайта HousingMaps (*http://www.housingmaps.com*), который позволяет пользователям просматривать дома, доступные в некоторой географической точке, получая список домов с сайта объявлений Craigslist (*http://www.craigslist.com*) и карты от службы Google maps (*http://code.google.com/apis/maps/index.html*).

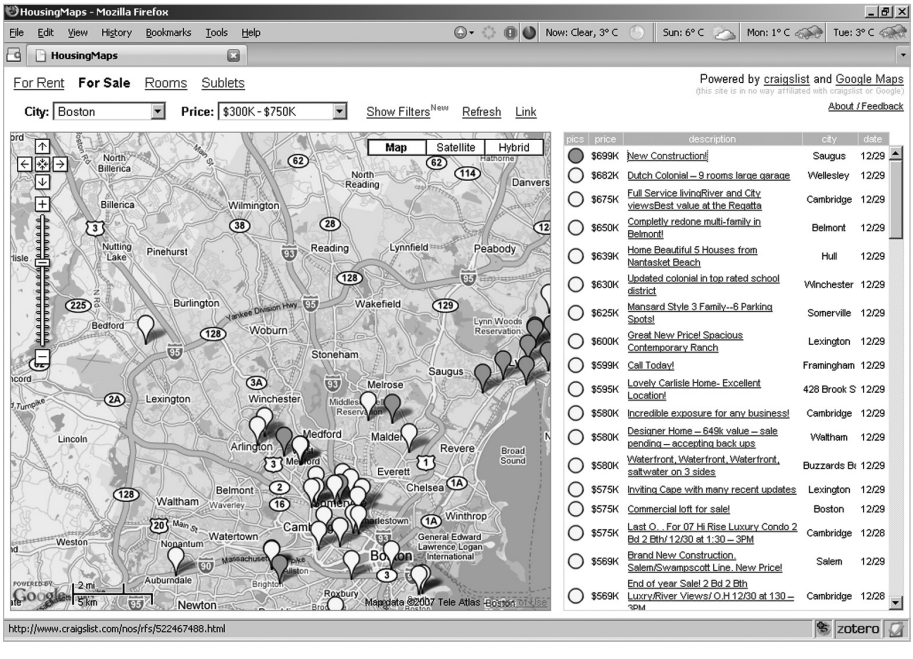

Рис. 1.6. *Список доступных домов, полученный с сайта Craigslist, в сочетании с картами местности, предоставленными службой Google maps (источник: http://www.housingmaps.com)*

Аналогичным образом, новостной сайт мог бы связать новостное сообщение с картой той местности, к которой относится данная новость. Возможность получить карту местности уже является усовершенствованием для любого приложения. Конечно, от этого ваше приложение не станет интеллектуальным, если только вы не сделаете что-то интеллектуальное с той информацией, которую получаете в результате обработки карты.

Географические карты – хороший пример получения внешней информации, но куда больший объем информации доступен в части Интернета, не связанной с картами. Давайте рассмотрим технологии, которые предоставляют такую информацию.

Обход контента и анализ экранных данных

Поисковые роботы (crawlers), их еще называют *пауками* (spiders), – это компьютерные программы, которые могут перемещаться по Интернету и скачивать общедоступный контент. Обычно поисковый робот будет посещать URL-адреса из некоторого списка, пытаясь пройти дальше по ссылкам и дойти до каждого пункта назначения. Этот процесс, который может повторяться многократно, обычно называют *глубиной обхода* (depth of crawling*)*. Посетив страницу, поисковый робот локально сохраняет ее контент для последующей обработки. Вы можете собрать массу данных таким образом, но быстро столкнетесь с проблемами их хранения или с нарушением авторских прав. Будьте осторожны и дисциплинированны, осуществляя обход контента. В главе 2 мы представляем нашу собственную реализацию поискового веб-робота. Мы также включили в эту книгу приложение, где дан общий обзор обхода контента Интернета, кратко описан наш собственный поисковый веб-робот, а также несколько реализаций с открытым исходным кодом.

Анализ экранных данных (screen scraping) относится к извлечению информации, которую содержат HTML-страницы. Это незамысловатое, но трудоемкое занятие. Допустим, вы хотите создать поисковую систему исключительно для поиска места, где можно поесть (такую как *http://www.foodiebytes.com*). Извлечение информации о меню с веб-страницы каждого ресторана – одна из ваших первоочередных задач. Применение рассматриваемых в этой книге методик может оказаться полезным для анализа экранных данных. В случае системы поиска ресторанов вы можете оценить, насколько хорош какой-то ресторан, опираясь на оценки тех, кто там отобедали. В некоторых случаях может быть доступна шкала оценок, но в большинстве случаев эти отзывы – обычный текст на естественном языке. Чтение отзывов, одного за другим, и соответствующая классификация ресторанов, очевидно, не являются масштабируемым бизнес-решением. В процессе анализа экранных данных можно использовать интеллектуальные методики, способные помочь вам автоматически рассортировать отзывы по категориям, а также оценить класс ресторанов. Примером является система поиска ресторанов Boorah (*http://www.boorah.com*).

RSS-каналы

Синдикация (syndication) веб-сайтов – еще один способ получения внешних данных, избавляющий вашего поискового робота от повторного посещения веб-сайтов. Как правило, синдицированный контент в большей степени приспособлен для машинной обработки, нежели обычные веб-страницы, потому что эта информация хорошо структурирована. Есть три распространенных формата таких каналов: RSS 1.0, RSS 2.0 и Atom.

Формат RDF Site Summary, RSS 1.0, как подсказывает его название, является потомком модели представления данных Resource Description Framework[1] (RDF) и основан на том представлении, что информация в Интернете может использоваться как человеком, так и компьютером. Однако если человек, как правило, способен понимать семантику контента (смысл слов или фраз в контексте), то компьютеру сделать это нелегко. Модель RDF была разработана для того, чтобы способствовать семантической интерпретации Интернета. Вы можете воспользоваться ею для извлечения полезных данных и метаданных в ваших целях. Спецификацию RSS 1.0 можно найти по адресу *http://web.resource.org/rss/1.0/*.

Формат Really Simple Syndication, RSS 2.0 базируется на технологии Rich Site Summary 0.91 фирмы Netscape, – здесь, мягко говоря, имеется существенная перегрузка значениями сокращения RSS, – и исходно этот формат предназначался для снижения уровня сложности форматов, основанных на модели RDF. Этот формат использует специальный язык синдикации с представлением его в виде простого XML-формата, который не требует использования пространства имен XML или обращения непосредственно к модели RDF. Сегодня почти все главные сайты обеспечивают каналы RSS 2.0; обычно эти каналы бесплатны для некоммерческого использования – для частных лиц и некоммерческих организаций. На посвященном RSS-каналам сайте фирмы Yahoo (*http://developer.yahoo.com/rss*) есть множество ресурсов, позволяющих постепенно ознакомиться с этой темой. Доступ к спецификации RSS 2.0 и другой сопутствующей информации можно получить по адресу *http://cyber.law.harvard.edu/rss*.

Наконец, вы можете использовать синдикацию на базе формата Atom. Наличие ряда проблем, связанных с применением формата RSS 2.0, привело к разработке стандарта Рабочей группы инженеров Интернета (Internet Engineering Task Force, IETF), изложенного в документе RFC 4287 (*http://tools.ietf.org/html/rfc4287*). Формат Atom не базируется на модели RDF; он не так гибок, как формат RSS 1.0, и не так легок, как формат RSS 2.0. По сути, это был компромисс между возможностями существующих стандартов и ограничением, связанным с необходимостью обеспечить максимальную обратную совместимость с другими форматами синдикации. Тем не менее формат Atom, подобно формату RSS 2.0, получил широкое распространение. Большинство крупных веб-агрегаторов (таких как Yahoo! и Google) предлагают новостные каналы этих двух форматов. Подробнее о формате синдикации Atom можно прочесть на веб-сайте IBM Developer Works (*http://www.ibm.com/developerworks/xml/standards/x-atomspec.html*).

---

[1] *http://www.w3.org/RDF*

Службы RESTful

Технология передачи репрезентативного состояния (Representational State Transfer, REST) была представлена в докторской диссертации Роя Филдинга (Roy T. Fielding)[1]. Это архитектурный стиль программного обеспечения для построения приложений в распределенной, гиперссылочной среде. REST – это клиент-серверная архитектура без состояния, которая каждую службу отображает в адрес URL. Если ваши нефункциональные требования не отличаются сложностью и формальный контракт между вами и поставщиком услуг не обязателен, архитектура REST может оказаться подходящим способом доступа к различным службам Интернета. Дополнительную информацию об этой важной технологии вы можете почерпнуть в книге «RESTful Web Services» Леонарда Ричардсона (Leonard Richardson) и Сэма Руби (Sam Ruby).

Многие веб-сайты предлагают службы RESTful, которые вы можете использовать в своем приложении. Сайт Digg обеспечивает интерфейс API (*http://apidoc.digg.com/*), который получает REST-запросы и предлагает несколько форматов для отклика, например XML, JSON, JavaScript и сериализованный PHP. Функционально, этот интерфейс API позволяет получить список сообщений, составленный с учетом разных критериев, список пользователей, друзей или фанатов пользователей и так далее.

API-интерфейс Facebook также является REST-подобным интерфейсом. Данный интерфейс позволяет взаимодействовать с этой интересной платформой посредством практически любого языка, по вашему выбору. Все, что надо сделать, – это послать HTTP-запрос `GET` или `POST` на REST-сервер, поддерживающий API-интерфейс Facebook. API-интерфейс Facebook хорошо документирован, и позже в этой книге мы воспользуемся им. Подробнее о нем можно прочесть по адресу *http://wiki.developers.facebook.com/index.php/API*.

Веб-службы

Веб-службы – это интерфейсы API, которые способствуют взаимодействию приложений друг с другом. Доступно большое количество фреймворков веб-служб, многие – с открытым исходным кодом. Плагин Apache Axis (*http://ws.apache.org/axis/*) – это реализация с открытым исходным кодом протокола обмена структурированными сообщениями в распределенной вычислительной среде (Simple Access Object Protocol, SOAP), который «может быть использован для обмена структурированной и типизированной информацией между узлами в децентрализованной, распределенной среде»[2]. Фреймворк Apache Axis поль-

---

[1] *http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm*

[2] *http://www.w3.org/TR/soap12-part0/ - L1153*

зуется популярностью и в версии 2 был полностью переработан. Версия Apache Axis2 поддерживает протоколы SOAP 1.1 и SOAP 1.2, а также широко распространенный стиль веб-служб REST, и предлагает ошеломляющее количество возможностей.

Другой заслуживающий упоминания проект Apache – это фреймворк Apache CXF (*http://incubator. apache.org/cxf/*), который появился в результате объединения проектов Celtix фирмы IONA и Codehaus XFire. Фреймворк Apache CXF поддерживает следующие стандарты: JAX-WS 2.0, JAX-WSA, JSR-181, SAAJ, SOAP 1.1, 1.2, WS-I Basic Profile, WS-Security, WS-Addressing, WS-RM, WS-Policy, WSDL 1.1 и 2.0. Он также поддерживает несколько транспортных механизмов, связывания (bindings) и различные форматы. Если вы подумываете о возможности использования веб-служб, следует взглянуть на этот проект.

Помимо множества доступных для веб-служб фреймворков имеются поставщики веб-служб, которых насчитывается еще больше. Почти каждая фирма использует веб-службы для интеграции приложений, совершенно не похожих друг на друга в смысле их функциональности или набора примененных в них технологий. Такая ситуация могла бы сложиться в результате слияния фирм или отсутствия координации разработок, которые ведутся параллельно в одной, обычно крупной фирме. В вертикальном пространстве почти все крупные финансовые и инвестиционные организации используют веб-службы для «бесшовной» интеграции. На сайте Xignite (*http://preview.xignite.com/Default. aspx*) предлагаются разнообразные финансовые веб-службы. Гиганты индустрии ПО (такие как фирмы SAP, Oracle и Microsoft) также предлагают поддержку веб-служб. Подводя итог, интеграция на базе веб-служб является вездесущей, и поскольку веб-службы – один из главных факторов, позволяющих осуществить такую интеграцию, они являются важным элементом инфраструктуры в проектировании интеллектуальных приложений.

На этом этапе вы уже должны представлять, как модернизировать свои готовые приложения, или задуматься о запуске очередного потрясающего проекта! Вы удостоверились в наличии у вас всех необходимых данных или, по крайней мере, в том, что можете получить доступ к таким данным. Теперь давайте посмотрим, какого рода интеллект мы собираемся ввести в свои приложения, и выясним его связь с некоторыми терминами, которые вам, скорее всего, уже знакомы.

1.5. Машинное обучение, интеллектуальный анализ данных и так далее

На протяжении всей этой книги говорится об «интеллекте», но что именно мы под этим понимаем? Говорим ли мы об области искусствен-

ного интеллекта? Как насчет машинного обучения? Это об интеллектуальном анализе данных и мягких вычислениях? Академики в соответствующих областях науки могут годами спорить о точном определении того, что мы собираемся представить. С точки зрения практики, большинство отличий не принципиальны, и это, преимущественно, вопрос контекста, а не сути. Данная книга является квинтэссенцией методик, принадлежащих всем упомянутым областям. Следовательно, давайте рассмотрим эти области.

Искусственный интеллект (artificial intelligence) – широкую известность получил его акроним *ИИ* (AI) – одно из компьютерных направлений, развивающееся с 1950 года. Исходно в области ИИ ставились довольно честолюбивые задачи, нацеленные на разработку компьютеров, способных мыслить подобно человеку [Russell, S. and P. Norvig, 2002; Buchanan, B. G., 2005][1]. Со временем цели стали более практичными и конкретными. Мания величия уступила место прагматизму, а он, в свою очередь, породил многие другие упомянутые нами компьютерные науки, такие как машинное обучение, интеллектуальный анализ данных, мягкие вычисления и так далее.

Сегодня самые развитые системы компьютерного интеллекта не в состоянии осмыслить простой рассказ, который без труда поймет четырехлетний ребенок. Что ж, если нельзя заставить компьютер «думать», – можем ли мы заставить его «обучаться»? Можно ли научить компьютер узнавать животное по его отличительным признакам? Как насчет злоупотребления применением субстандартной ипотеки? Как насчет чего-то более сложного, вроде распознавания голоса и ответа на родном для вас языке, – способен компьютер на это? Ответом на эти вопросы является громкое и решительное «да». Тем не менее вы вправе поинтересоваться: «Из-за чего, собственно, вся эта сумятица?» В конце концов, всегда можно создать громадную таблицу соответствий и получать ответы на свои вопросы, опираясь на содержимое базы данных.

Безусловно, вы можете следовать таблице соответствий, но у этого подхода есть несколько проблем. Во-первых, для любой серьезной задачи в реальной производственной системе такая таблица соответствий имела бы гигантский размер; следовательно, это решение не является оптимальным по соображениям эффективности. Во-вторых, если ваш вопрос сформулирован на основе данных, которых нет в базе данных, вы вообще не получите ответ. Человека за такое поведение вы тут же наградили бы эпитетами, привести которые на этих страницах нам не позволила бы цензура. И последнее, кому-то пришлось бы создать и обслуживать вашу таблицу соответствий, и число таких людей возрастало бы вместе с размером таблицы: деталь, которая вряд ли устроит

[1] Список литературы приводится в конце каждой главы. – *Прим. ред.*

финансовый отдел вашей организации. Поэтому нам требуется нечто получше таблицы соответствий.

Машинное обучение (machine learning) – это способность программной системы, опираясь на прошлый опыт, делать обобщения и использовать их, предоставляя ответы на вопросы, связанные с данными, которые встречались такой системе в прошлом, а также отвечать на вопросы, касающиеся новых данных, с которыми система раньше никогда не сталкивалась. Некоторые обучающие алгоритмы прозрачны для человека – он может проследить ход рассуждений, на которых основано обобщение. Примерами алгоритмов, обеспечивающих прозрачность обучения, являются деревья решений и, обобщая, все методы обучения, основанные на правилах. Есть и другие алгоритмы обучения, не являющиеся прозрачными для человека, – в эту категорию попадают нейронные сети и метод опорных векторов (support vector machines, SVM). Не забывайте, что машинный интеллект, как и человеческий, не является безошибочным. В мире интеллектуальных приложений вы научитесь иметь дело с неопределенностью и нечеткостью; точно так же, как в реальном мире, любой полученный вами ответ будет допустимым с определенной степенью достоверности, но не с уверенностью. В повседневной жизни мы просто допускаем, что определенные вещи произойдут наверняка. По этой причине, используя интеллектуальные приложения, мы будем решать проблемы достоверности, допустимости и стоимости ошибок.

1.6. Восемь заблуждений насчет интеллектуальных приложений

Мы рассмотрели весь вводный материал. К этому моменту у вас должно сложиться довольно прочное, хотя и в общих чертах, представление о том, что такое интеллектуальные приложения и как вы собираетесь их использовать. Скорее всего, вы уже в достаточной степени мотивированы и стремитесь поскорее углубиться в программирование. Мы вас не разочаруем. Каждая глава – кроме этого введения – содержит новый и ценный программный код.

Но прежде чем отправиться в путешествие по яркому и полезному в финансовом отношении (для наиболее прагматичных из нас) миру интеллектуальных приложений, мы приведем ряд ошибок, или заблуждений, характерных для тех проектов, где в функциональность внедряется интеллект. Вам, возможно, знакомы восемь заблуждений распределенных вычислений; если нет, см. профессиональный комментарий [Van den Hoogen]. Это набор часто встречающихся, но неверных допущений, которые делают программисты, впервые приступая к раз-

работке распределенного приложения. По аналогии, мы подготовили свой набор и, не нарушая традиции, перечислим восемь заблуждений.

1.6.1. Заблуждение 1: данные достоверны

Есть много причин, по которым данные могут быть недостоверными. Поэтому всегда следует проводить проверку на предмет того, можно ли доверять данным, с которыми вы работаете, прежде чем переходить к рассмотрению конкретных интеллектуально-алгоритмических решений проблемы. Недостоверные данные способны даже умного человека привести к ошибочным умозаключениям.

Вот наглядный (но не исчерпывающий) перечень того, что может случиться с данными:

- Данные, которыми вы располагали на этапе разработки, возможно, не являются репрезентативной выборкой, представляющей рабочую среду. Допустим, вы хотите распределить пользователей социальной сети по категориям «длинный»/«средний»/«короткий» в зависимости от их роста. Если среди данных, которыми вы располагали на этапе разработки, минимальное значение роста было 180 см, то вы рискуете назвать коротышкой пользователя, рост которого «всего» 180 см.
- Данные могут содержать пропущенные значения. Фактически, если только ваши данные не созданы искусственно, в них почти наверняка будут пропущены значения. Обработка пропущенных значений – дело хитрое. Обычно вы или оставляете эти пропуски «как есть», или заполняете их какими-то значениями, принятыми по умолчанию или вычисленными. Оба эти варианта могут привести к нестабильности реализаций.
- Данные могут меняться. Возможно, изменения вносятся в схему базы данных, или меняется семантика данных в базе данных.
- Данные могут быть ненормализованными. Допустим, нас интересуют данные о массе тела членов какой-то группы. Чтобы сделать какие-либо осмысленные выводы, основываясь на величине массы тела, для всех членов группы должна использоваться одна и та же единица измерения – в фунтах или в килограммах для каждого человека, но не смешанные измерения – и в фунтах, и в килограммах.
- Данные могут не соответствовать тому алгоритмическому подходу, который вы планируете применить. Данные бывают различных видов и форм, известных как *типы данных*. Одни наборы данных являются числовыми, другие – нет. Одни наборы данных можно упорядочить, другие – нельзя. Одни числовые наборы данных являются дискретными (например, количество людей в комнате), другие – непрерывными (температура воздуха).

1.6.2. Заблуждение 2: логический вывод осуществляется мгновенно

Вычисление решения отнимает какое-то время, и быстрота реакции приложения может оказаться ключевым фактором с точки зрения финансового успеха вашего бизнеса. Не следует исходить из предположения, что все алгоритмы на всех наборах данных будут выполняться в пределах ограниченного времени отклика, допустимого для вашего приложения. Вы должны будете протестировать производительность выбранного алгоритма в диапазоне рабочих параметров.

1.6.3. Заблуждение 3: размер данных не имеет значения

Когда мы говорим об интеллектуальных приложениях, размер имеет значение! Размер данных заявляет о себе двумя путями. Первый связан с быстротой отклика приложения, как было упомянуто в заблуждении 2. Второй путь связан с тем, способны ли вы получить осмысленные результаты на большом наборе данных. Не исключено, вы сможете предоставить превосходные рекомендации по фильмам или музыке ряду пользователей, когда их порядка 100 человек, но результатами этого же алгоритма могут стать никуда не годные рекомендации, если пользователей станет порядка 100 000.

И наоборот, в некоторых случаях, чем больше у вас данных, тем интеллектуальнее может быть ваше приложение. Таким образом, размер данных важен в нескольких отношениях, и всегда следует спросить себя: достаточно ли у меня данных? Как это скажется на качестве моего интеллектуального приложения, если придется обрабатывать данные, объем которых в 10 раз больше?

1.6.4. Заблуждение 4: масштабируемость решения – не проблема

Еще одно заблуждение, связанное с заблуждениями 2 и 3, но отличающееся от них, – допускать, что решение для интеллектуального приложения можно масштабировать путем простого увеличения парка компьютеров. Не надейтесь, что ваше решение является масштабируемым. Одни алгоритмы поддаются масштабированию, другие – нет. Допустим, мы пытаемся найти группы схожих экстренных новостных сообщений среди миллиардов заголовков. Не все алгоритмы кластеризации (глава 4) поддерживают распараллеливание. Вы должны учесть возможности масштабирования на стадии проектирования приложения. В некоторых случаях, возможно, удастся разбить данные и распараллелить выбранный интеллектуальный алгоритм, чтобы применить его к наборам данных меньшего размера. Алгоритмы, выбранные вами на стадии проектирования, могут иметь параллельные версии, но это

следует выяснить в самом начале, потому что обычно эти алгоритмы требуют массивной инфраструктуры и бизнес-логики.

1.6.5. Заблуждение 5: одна хорошая библиотека годится на все случаи

Заманчиво каждый раз использовать для решения разнообразных проблем, связанных с интеллектуальным поведением приложения, одну и ту же зарекомендовавшую себя методику. Сопротивляйтесь этому соблазну любой ценой! Мне встречались те, кто пытался решить буквально любую задачу посредством поисковой системы Lucene. Если вы поймали себя на том, что поступаете подобным образом, вспомните выражение: когда в руке молоток, все вокруг кажется гвоздями.

ПО интеллектуальных приложений, как и любое другое ПО, имеет определенную область применения и определенные ограничения. Удостоверьтесь в том, что вы тщательно протестировали свое излюбленное решение в новых областях применения. Кроме того, рекомендуется на каждую задачу смотреть свежим взглядом: возможно, для решения другой задачи эффективнее и целесообразнее использовать другой алгоритм.

1.6.6. Заблуждение 6: время вычислений известно

Классические примеры из этой категории можно отыскать среди задач, задействующих оптимизацию. В некоторых приложениях допускается широкий разброс в показателях времени решения при относительно небольшом расхождении в значениях участвующих параметров. Обычно ожидается, что в случае изменения параметров задачи она может иметь соответствующее решение с учетом времени отклика. Имея метод, возвращающий расстояние между двумя точками земного шара, вы можете рассчитывать на то, что время решения не будет зависеть ни от одной из этих двух конкретных географических координат. Но это истинно не для всех задач. Безобидное внешне изменение в данных может существенно повлиять на время решения, и разница порой выражается не в секундах, а в часах!

1.6.7. Заблуждение 7: чем сложнее модель, тем лучше

Ничто не может быть дальше от истины. Всегда начинайте с самой простой модели, которую только можете себе представить. В дальнейшем старайтесь постепенно улучшать получаемые результаты, комбинируя решение с новыми интеллектуальными элементами. Ваш союзник – девиз KISS[1], незаменимый при разработке ПО.

---

[1] KISS – это мем, аббревиатура для расхожего выражения «keep it simple, stupid» (не усложняй, глупец). – *Прим. перев.*

1.6.8. Заблуждение 8: существуют модели без систематической ошибки

Есть две причины, по которым когда-либо произносятся эти слова, – невежество либо систематическая ошибка! Выбор моделей, которые вы создаете, и данных, которые вы используете, чтобы тренировать свои алгоритмы обучения, вносит погрешность. Мы здесь не станем подробно, по-научному описывать систематическую ошибку в системах приобретения знаний. Но обратим ваше внимание на то, что систематическая ошибка служит противовесом обобщению, в том смысле, что решение будет стремиться к нашему описанию модели и к нашим данным (по структуре). Другими словами, систематическая ошибка вводит наше решение в рамки того, что нам известно о мире (фактов), и иногда этот набор включает то, как мы узнали то, что знаем; тогда как обобщение пытается зафиксировать то, чего мы не знаем (фактически), но находим целесообразным принять за истину, что нам это известно, и рассматривать эту истину как данность.

1.7. Заключение

В этой главе мы предоставили общий обзор интеллектуальных веб-приложений с несколькими конкретными примерами на основе реально существующих веб-сайтов и дали практическое определение интеллектуальных веб-приложений, которое может послужить основой для проектирования. В этом определении объединены три различных компонента: 1) агрегированность данных, 2) ссылочные структуры и 3) алгоритмы, которые демонстрируют способность к обучению и позволяют манипулировать неопределенностью.

Мы предусмотрели проверку в реальных условиях представленным шести основным категориям веб-приложений, к которым можно уверенно применить данное нами определение. Затем мы представили эффективные технологии, позволяющие агрегировать данные или получить доступ к платформам агрегации данных. Мы также заложили фундамент, рассказав об истоках методик, которые будут рассмотрены в последующих главах, и в частности, о связи этих методик с такими областями, как искусственный интеллект, машинное обучение, интеллектуальный анализ данных и мягкие вычисления.

Наконец, мы указали восемь заблуждений проектирования, часто встречающихся на практике. Эти заблуждения приведены не как строго установленные факты, но как общие эмпирические рекомендации. Мы полагаем, что знание этих восьми рекомендаций поможет вам сэкономить много времени и снизить потребление кофеина. Далее в книге мы поочередно проанализируем ряд методик, которые могут добавить интеллектуальное поведение в разрабатываемое вами приложение. Дополнительный материал по каждой из этих методик дается в конце каждой главы в разделе «Сделать».

1.8. Ссылки

Buchanan, B. G. «A (Very) Brief History of Artificial Intelligence». *AI Magazine*. Volume 26, issue 4, 2005.

Gómez-Pérez, A., M. Fernández-López, O. Corcho. «Ontological Engineering: With Examples from the Areas of Knowledge Management, E-commerce and the Semantic Web». Springer, 2005.

Hart, P. E., N. J. Nilsson, and B. Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». *IEEE Transactions on Systems Science and Cybernetics*. Volume 4, issue 2, 1968.

Hart, P. E., N. J. Nilsson, and B. Raphael. Correction to «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». *SIGART Newsletter 37*, 1972.

Richardson, L. and S. Ruby. «RESTful Web Services». O'Reilly Media, 2007.

Russell, S. and P. Norvig. «Artificial Intelligence: A Modern Approach» (2nd Edition). Prentice Hall, 2002.[1]

Van Den Hoogen, I. «Deutsch's fallacies, 10 Years After». *Java Developer's Journal*, 2004. *http://java.sys-con.com/read/38665.htm*

[1] Рассел С., Норвиг П. «Искусственный интеллект: Современный подход», 2-е изд., Вильямс, 2007.

2

Поиск

- Поиск с применением библиотеки Lucene
- Вычисление вектора Pagerank
- Ограничения масштабных вычислений

Предположим, есть список документов, и вам интересно прочесть те из них, которые имеют отношение к фразе «Armageddon is near» (Армагеддон близок) – ну, или к какой-либо менее мрачной. Как бы вы реализовали решение этой задачи? Прямолинейным и наивным было бы решение читать каждый документ и сохранять его, если в нем отыщется фраза «Armageddon is near». Можно было бы даже подсчитать, сколько раз в каждом из документов встретилось каждое слово из искомой фразы, и в соответствии с этими счетчиками отсортировать полученные результаты в порядке убывания. Такая процедура называется *информационным поиском* (information retrieval, IR), или просто *поиском*. Поиск не является новой функциональностью: почти в каждом приложении имеется та или иная реализация этой возможности, но интеллектуальный поиск выходит за рамки простого поиска прежних времен.

Поэкспериментировав, вы убедитесь в том, что упомянутое наивное решение задачи информационного поиска изобилует проблемами. Например, как только вы увеличите количество документов или их размер, производительность такого решения окажется неприемлемой для большинства случаев применения. К счастью, накоплен громадный объем знаний об информационном поиске и доступны довольно сложные и надежные библиотеки, которые предлагают масштабируемость и высокую производительность. Наиболее удачной библиотекой информационного поиска, применяемой при программировании на язы-

ке Java, является библиотека Lucene – проект Дуга Каттинга (Doug Cutting), начатый им почти 10 лет назад. Библиотека Lucene может помочь в решении задач информационного поиска, поскольку обеспечивает индексирование всех документов, позволяя моментально найти в них искомое! Книга «Lucene in Action» Отиса Господнетика (Otis Gospodnetić) и Эрика Хэтчера (Erik Hatcher), вышедшая в издательстве Manning, обязательна к прочтению – особенно если вы хотите знать, как индексировать данные, – и является введением в поиск, сортировку, фильтрацию и представление результатов поиска.

Современный поиск выходит далеко за рамки индексирования. Самая жестокая конкурентная борьба среди фирм, занимающихся поисковыми системами, не охватывает технологические проблемы индексирования, но разворачивается вокруг таких задач, как оценка ссылок, анализ экранных данных, а также обработка естественного языка. Подобные методики повышают функциональные возможности поиска, что иногда оборачивается миллиардами долларов, как это было в случае фирмы Google.

В этой главе вкратце рассмотрены функциональные возможности библиотеки Lucene и продемонстрировано ее применение. Мы представим алгоритм PageRank, который до настоящего времени остается самым удачным алгоритмом оценки ссылок, а также вероятностную методику анализа экранных данных. Мы объединим все эти методики, показав, что их успешное взаимодействие позволяет получать более точные результаты поиска. Материал излагается последовательно, так что вы можете изучить поиск в той степени, какую сочтете нужной, и вернуться к этой теме позже, если сейчас вам на это жалко времени. Давайте без лишних разговоров подберем ряд документов и поищем в них различные выражения с помощью библиотеки Lucene.

2.1. Поиск с применением библиотеки Lucene

Поиск с применением библиотеки Lucene будет основной темой оставшейся части этой главы. Итак, прежде чем браться за современные интеллектуальные алгоритмы, необходимо изучить шаги, из которых складывается традиционный информационный поиск. Попутно мы покажем, как использовать библиотеку Lucene для поиска в наборе найденных документов, представим некоторые внутренние механизмы этой библиотеки и сделаем общий обзор основных этапов построения поисковой системы.

Данные, которые вы хотите найти, могли бы находиться в вашей базе данных, в Интернете или в любой другой сети, доступной для вашего приложения. Данные из Интернета можно собрать с помощью поискового робота. Есть несколько бесплатных поисковых роботов, но мы воспользуемся тем, который написали сами для этой книги. Мы будем работать с набором страниц, которые собрали в Интернете 6 ноября

2006 года, так что эти страницы можно целенаправленно модифицировать и наблюдать влияние внесенных изменений на результаты алгоритмов.

Из собранных страниц мы удалили все лишнее и внесли в них изменения, с тем чтобы этот набор образовал подобие крошечного Интернета. Указанные страницы можно найти в каталоге *data/ch02/*. Важно ознакомиться с контентом этих документов, чтобы вы могли разобраться в том, что делают алгоритмы, и понять, как они работают. Отобранные нами 15 документов (выбор контента носил случайный характер) представляют собой следующее:

- Семь документов относятся к новостям бизнеса: три из них связаны с проникновением фирмы Google в сферу газетной рекламы, еще в трех обсуждаются, главным образом, акции фирмы NVidia, а один документ содержит информацию о стоимости акций и перемещениях индексов.
- Три документа касаются попытки Лэнса Армстронга пробежать марафон в Нью-Йорке.
- Четыре документа связаны с политикой США, в частности с выборами в Конгресс (2006 года).
- Пять документов относятся к международным новостям: четыре сообщения о победе Ортеги на выборах в Никарагуа и одно – о глобальном потеплении.

Библиотека Lucene поможет нам проанализировать, индексировать и найти эти и любые другие документы, которые можно преобразовать в текст, так что наш поиск не ограничивается только веб-страницами. Класс, которым мы воспользуемся, чтобы быстро прочитать сохраненные веб-страницы, называется `FetchAndProcessCrawle`; этот класс может также извлекать данные из Веба. Конструктор этого класса получает три аргумента:

- Базовый каталог для хранения извлеченных данных.
- Глубину, на которую требуется пройти по ссылочной структуре.
- Максимальное число документов, которые следует извлечь.

В листинге 2.1 показано, как можно использовать этот класс в среде BeanShell.

Листинг 2.1. Чтение, индексирование и поиск веб-страниц из заданного по умолчанию списка

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);

crawler.setDefaultUrls();      ← Загрузка файлов

crawler.run();                 ← Сбор и обработка контента
```

```
LuceneIndexer luceneIndexer = new LuceneIndexer(crawler.getRootDir());

luceneIndexer.run();                ← Содержимое индекса помещается в каталог

MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());

oracle.search("armstrong",5);       ← Поиск с применением только что созданного индекса
```

Обход контента (краулинг, crawling) и этап предварительной обработки должны занять лишь несколько секунд, и после их завершения в базовом каталоге должен появиться новый подкаталог. В рассматриваемом нами примере в качестве базового был указан каталог C:/iWeb2/data/ch02. Имя нового каталога будет начинаться строкой crawl-, после которой указывается числовое значение временной метки поискового робота в миллисекундах – например, crawl-1200697910111.

Вы можете изменить контент документов или добавить новые документы и повторить предварительную обработку и индексирование файлов, чтобы наблюдать, как отличаются результаты поиска. На рис. 2.1 представлен снимок экрана, полученный во время выполнения программного кода из листинга 2.1 в среде BeanShell и включающий результаты поиска слова «Armstrong».

Такова механика высокого уровня – загрузка, индексирование, поиск. Нет ничего проще! Но как это работает на самом деле? Какие важнейшие элементы задействованы на каждом этапе?

2.1.1. Программный код библиотеки Lucene

Проанализируем последовательность событий, позволяющих нам выполнить поиск. Задача класса FetchAndProcessCrawler – извлечение данных и выполнение их синтаксического разбора. Результат этой обработки запоминается в подкаталоге processed. Оторвитесь на минуту от чтения, чтобы заглянуть в эту папку. Для каждой группы обрабатываемых документов предусмотрены четыре подкаталога – fetched, knownurls, pagelinks и processed. Обратите внимание, мы «разобрали» веб-страницы, отделив метаданные от контента ядра, и извлекли ссылки, ведущие с одной страницы на другую – так называемые *внешние ссылки* (outlinks.) В классе FetchAndProcessCrawler не используется никакой код из интерфейса Lucene API.

На следующем шаге мы создали экземпляр класса LuceneIndexer и вызвали его метод run(). Именно здесь библиотека Lucene используется для того, чтобы индексировать обрабатываемый контент. Файлы индексов, полученные с помощью библиотеки Lucene, будут храниться в отдельном каталоге с именем lucene-index. Класс LuceneIndexer – это удобная оболочка для вызова класса LuceneIndexBuilder в среде BeanShell. Именно в классе LuceneIndexBuilder используется интерфейс Lucene API. На рис. 2.2 в полном объеме представлена UML-диаграмма основных классов, участвующих в извлечении и индексировании документов.

```
bsh % FetchAndProcessCrawler c =
new FetchAndProcessCrawler("c:/iWeb2/data/ch02",5,200);
bsh % c.setDefaultUrls();
bsh % c.run();
There are no unprocessed urls.
Timer (s): [Crawler fetched data]  --> 5.5
Timer (s): [Crawler processed data]  --> 0.485
bsh %
bsh % LuceneIndexer lidx = new LuceneIndexer(c.getRootDir());
bsh % lidx.run();
Starting the indexing ... Indexing completed!

bsh % MySearcher oracle = new MySearcher(lidx.getLuceneDir());
bsh % oracle.search("armstrong",5);

Search results using Lucene index scores:
Query: armstrong

Document Title: Lance Armstrong meets goal in painful marathon debut
Document URL: file:/c:/iWeb2/data/ch02/sport 01.html -->
Relevance Score: 0.397706508636475
_____________________________________
Document Title: New York 'tour' Lance's toughest
Document URL: file:/c:/iWeb2/data/ch02/sport-03.html -->
Relevance Score: 0.312822639942169
_____________________________________
Document Title: New York City Marathon
Document URL: file:/c:/iWeb2/data/ch02/sport-02.html -->
Relevance Score: 0.226110160350800

_____________________________________
```

Рис. 2.1. *Пример с извлечением данных, синтаксическим разбором, анализом, индексированием и поиском, выполненными применительно к набору веб-страниц с помощью нескольких строк программного кода*

В листинге 2.2 показан весь программный код класса `LuceneIndexBuilder`.

Листинг 2.2. Класс LuceneIndexBuilder обеспечивает создание индекса Lucene

```
public class LuceneIndexBuilder implements CrawlDataProcessor {

  private File indexDir;

  public LuceneIndexBuilder(File indexDir) {

      this.indexDir = indexDir;

       try {
```

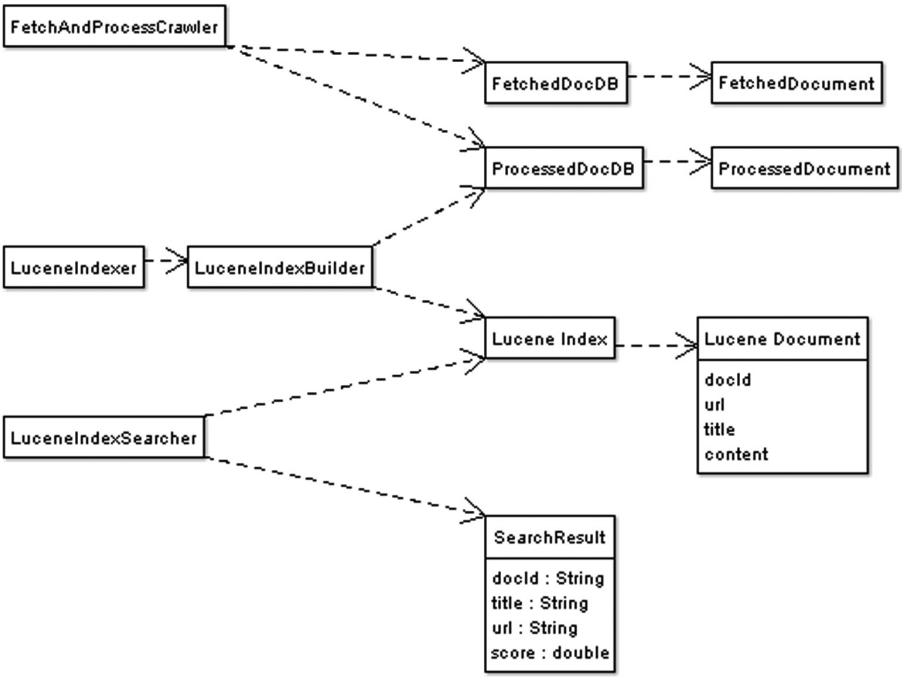

Рис. 2.2. *UML-диаграмма классов, которые мы использовали для обхода веб-страниц из набора, а также для индексирования этих страниц и поиска в них*

```
            IndexWriter indexWriter =                    ← Создание индекса Lucene
➥           new IndexWriter(indexDir, new StandardAnalyzer(), true);

            indexWriter.close();
} catch(IOException ioX) {
             throw new RuntimeException("Error: ", ioX);
        }
    }
    public void run(CrawlData crawlData) {

       List<String> allGroups =                                  Получение
           crawlData.getProcessedDocsDB().getAllGroupIds();      всех групп
                                                                 документов

        for(String groupId : allGroups) {
          buildLuceneIndex(groupId, crawlData.getProcessedDocsDB());
        }
    }

    private void buildLuceneIndex(String groupId,
➥   ProcessedDocsDB parsedDocsService) {

        try {
```

```
            List<String> docIdList =          ← Получение всех документов группы
parsedDocsService.getDocumentIds(groupId);

            IndexWriter indexWriter =
new IndexWriter(indexDir, new StandardAnalyzer(), false);

            for(String docId : docIdList) {   ← Индексирование всех документов

                indexDocument(indexWriter,
➥               parsedDocsService.loadDocument(docId));
            }

            indexWriter.close();

        } catch(IOException ioX) {
            throw new RuntimeException("Error: ", ioX);
        }
    }

    private void indexDocument(IndexWriter iw,
➥   ProcessedDocument parsedDoc) throws IOException {

        org.apache.lucene.document.Document doc =
➥       new org.apache.lucene.document.Document();

        doc.add(new Field("content", parsedDoc.getText(),
➥       Field.Store.NO, Field.Index.TOKENIZED));

        doc.add(new Field("url", parsedDoc.getDocumentURL().
➥       toExternalForm(), Field.Store.YES, Field.Index.NO));

        doc.add(new Field("docid", parsedDoc.getDocumentId(),
➥       Field.Store.YES, Field.Index.NO));

        doc.add(new Field("title", parsedDoc.getDocumentTitle(),
➥       Field.Store.YES, Field.Index.NO));

        doc.add(new Field("doctype", parsedDoc.getDocumentType(),
➥       Field.Store.YES,Field.Index.NO));

        iw.addDocument(doc);
    }
}
```

В библиотеке Lucene для создания индекса используется класс `IndexWriter`. У этого класса очень много конструкторов, с которыми можно ознакомиться в документации `Javadocs`. Конкретный конструктор, который используем мы в своем коде, получает три аргумента:

- Каталог для хранения индекса.
- Анализатор, который мы хотим использовать (об анализаторах поговорим позже в этом разделе).
- Булеву переменную, которая определяет, надо ли перезаписывать существующий каталог.

Как видно из листинга 2.2, мы организуем последовательный перебор групп документов, собранных поисковым роботом. Первая группа соответствует контенту URL-адресов из исходного списка. Вторая группа содержит документы, которые были найдены в процессе чтения контента URL-адресов из исходного списка. Третья группа будет содержать документы, до которых можно добраться из документов второй группы, и так далее. Обратите внимание: структура каталогов меняется, если вы варьируете значение параметра `maxBatchSize` класса `BasicWebCrawler`. Чтобы сохранить описанную структуру в первозданном виде, убедитесь в том, что указанный параметр является достаточно большим числом; соответственно целям этой книги он имеет значение 50.

Такая структура каталогов будет полезна, если вы примените наш поисковый робот для извлечения намного большего набора данных из Интернета. Для простой структуры веб-страниц, которую мы используем в этой книге, вы можете наблюдать эффект создания групп, добавив лишь несколько URL-адресов, – это делается с помощью метода `addUrl` класса `FetchAndProcessCrawler`, – и позволив роботу обнаружить остальные файлы.

Для каждого документа в группе мы индексируем его контент. Это делается в методе `indexDocument`, который показан в конце листинга 2.2. Класс `Document` из библиотеки Lucene инкапсулирует извлеченные документы, так что их можно добавить в индекс; этот же класс можно использовать для инкапсуляции не только веб-страниц, но также сообщений электронной почты, файлов PDF и всего того, что поддается синтаксическому разбору и преобразованию в обычный текст. Каждый экземпляр класса `Document` является виртуальным документом, который представляет собой коллекцию полей. Обратите внимание: мы используем результаты «разбора» извлеченных документов, чтобы создать различные экземпляры объекта класса `Field` для каждого документа:

- Поле `content` соответствует текстовому представлению каждого документа, очищенного от всех тегов форматирования и других аннотаций. Эти документы можно найти в подкаталоге `processed/1/txt`.
- Поле `url` представляет собой URL-адрес, который был использован для извлечения данного документа.
- Поле `docid` уникальным образом идентифицирует каждый документ.
- Поле `title` служит для хранения заголовка каждого документа.
- Поле `doctype` служит для хранения типа каждого документа, например HTML или Microsoft Word.

Поле `content` каждого документа индексируется, но не хранится с файлами индекса; другие поля хранятся вместе с файлами индекса, но не индексируются. Причина в том, что нам нужна возможность обратить-

ся с запросом к контенту, но получать URL-адрес, идентификатор ID и заглавие каждого извлеченного документа мы хотим из файлов индекса.

Это общепринятая практика. Как правило, хранят несколько указателей, позволяющих идентифицировать найденное в индексе, но не включают контент в содержимое файлов индекса, если только для этого нет серьезных оснований (возможно, вам требуется без промедления получить часть контента, а обратиться прямо к первоисточнику нельзя – он недоступен). В этом случае обратите внимание на размер файлов, создаваемых на этапе индексирования.

Для поиска в только что созданном индексе мы используем класс `MySearcher`. В листинге 2.3 показан весь программный код этого класса. Для его создания требуется единственный аргумент – каталог, в котором хранится индекс Lucene, – после чего можно выпонять поиск с помощью метода `search`, получающего два аргумента:

- Строку с запросом, который требуется выполнить применительно к индексу.
- Максимальное число документов, которые мы хотим извлечь.

Листинг 2.3. Класс `MySearcher`*: извлечение результатов поиска на основе индексирования Lucene*

```
public class MySearcher {

  private static final Logger log =
➥ Logger.getLogger(MySearcher.class);

  private String indexDir;

  public MySearcher(String indexDir) {
    this.indexDir = indexDir;
  }

  public SearchResult[] search(String query, int numberOfMatches) {

    SearchResult[] docResults = new SearchResult[0];
    IndexSearcher is = null;

    try {                                                    Открытие
                                                             индекса Lucene
      is = new IndexSearcher(FSDirectory.getDirectory(indexDir));
    } catch (IOException ioX) {
      log.error(ioX.getMessage());
    }

QueryParser qp = new QueryParser("content",          ← Создание парсера запроса
                        new StandardAnalyzer());
    Query q = null;
    try {
```

```
        q = qp.parse(query);          ← Преобразование тестового запроса в запрос Lucene
      } catch (ParseException pX) {
        log.error(pX.getMessage());
      }

      Hits hits = null;
      try {

        hits = is.search(q);   ← Поиск в индексе

        int n = Math.min(hits.length(), numberOfMatches);
        docResults = new SearchResult[n];

        for (int i = 0; i < n; i++) {   ← Сбор N первых результатов поиска

          docResults[i] = new SearchResult(hits.doc(i).get("docid"),
                                           hits.doc(i).get("doctype"),
                                           hits.doc(i).get("title"),
                                           hits.doc(i).get("url"),
                                           hits.score(i));   ← Оценка i-го документа

// Отчет о полученных результатах
          System.out.println(docResults[i].print());
        }
        is.close();

      } catch (IOException ioX) {
         log.error(ioX.getMessage());
      }
      return docResults;
    }
  }
```

Проанализируем шаги, реализованные в листинге 2.3:

1. Мы используем экземпляр класса `IndexSearcher` из библиотеки Lucene, чтобы открыть индекс для поиска в нем.
2. Создаем экземпляр класса `QueryParser` из библиотеки Lucene, указывая в качестве параметров имя поля, применительно к которому выполняется запрос, и анализатор, который следует использовать для разметки текста запроса.
3. Используем метод `parse` класса `QueryParser`, чтобы преобразовать запрос на естественном языке в экземпляр объекта `Query`, понятный для Lucene.
4. Выполняем поиск в индексе и получаем результаты в виде объекта `Hits` библиотеки Lucene.
5. Организуем циклический перебор первых *n* результатов и собираем их в коллекцию объектов `SearchResult`. Обратите внимание: объект `Hits` Lucene содержит только ссылки на исходные документы. Эти

ссылки используются для сбора данных из нужных полей: например, обращение `hits.doc(i).get("url")` обеспечит возврат URL-адреса, сохраненного нами в индексе.

6. Фиксируем *оценку релевантности* (relevance score) для каждого извлеченного документа. Эта оценка представляет собой число от 0 до 1.

Из таких элементов состоит механизм нашей конкретной реализации. Давайте вернемся на шаг назад и рассмотрим более общую картину организации поиска на основе индексирования. Это поможет осознать вклад систем поиска на основе индексирования и подготовит нас к обсуждению более развитых функциональных возможностей поиска.

2.1.2. Анализ основных этапов поиска

Если бы можно было вернуться в прошлое (например, в 1998 год), какие основные этапы нам пришлось бы пройти тогда, чтобы проделать работу по построению поисковой системы? Сегодня эти этапы остаются теми же, какими они были в 1998 году, но мы повысили их результативность и вычислительную производительность. На рис. 2.3 изображены основные этапы традиционной процедуры поиска:

- Краулинг
- Синтаксический разбор
- Анализ
- Индексирование
- Поиск

Краулинг – это процесс сбора документов, по которым мы хотим выполнять поиск. Возможно, этот этап не нужен, если документы уже существуют или включены в коллекцию. Синтаксический разбор необходим для преобразования документов (XML, HTML, Word, PDF) в простую структуру, которая обеспечит представление полей индексирования исключительно в виде текста. Для разбора рассматриваемых в этой книге примеров используется программный код из проекта NekoHTML. Проект NekoHTML включает простой парсер HTML-кода, способный просканировать файлы HTML и «исправить» многие распространенные ошибки, которые встречаются в HTML-документах: добавить пропущенные родительские элементы, автоматически закрыть элементы необязательными концевыми тегами и обработать случаи несовпадения тегов элементов внутри строки. Система NekoHTML вполне надежна и обеспечивает достаточное быстродействие, но если вы выполняете краулинг нестандартных сайтов, вам, возможно, придется написать собственный парсер.

Если вы планируете индексировать PDF-документы , можно воспользоваться программным кодом из проекта `PDFBox` (*http://www.pdfbox.org/*):

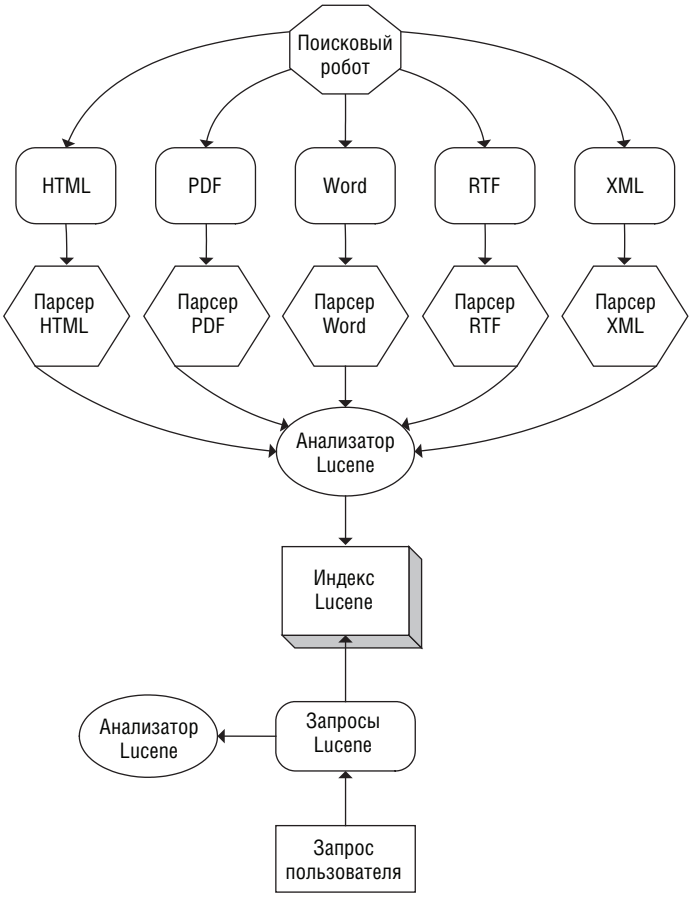

Рис. 2.3. *Обзор процедуры поиска в наборе документов разных форматов*

этот проект реализован в рамках лицензии BSD[1] и очень подробно документирован. Проект `PDFBox` включает класс `LucenePDFDocument`, позволяющий получить Lucene-объект `Document` с помощью единственной строки кода:

```
Document doc = LucenePDFDocument.convertDocument(File file)
```

Дополнительную информацию ищите в документации `Javadocs`. Как и для документов PDF, есть парсеры для документов Word. Например, проект Apache POI (*http://poi.apache.org/*) обеспечивает API для мани-

[1] Лицензия BSD (BSD license, Berkley Software Distribution license – Программная лицензия университета Беркли) – это лицензионное соглашение, впервые примененное для распространения UNIX-подобных операционных систем BSD. – *Прим. перев.*

пуляций с файловыми форматами на основе формата Microsoft OLE 2 Compound Document, используя для этого исключительно язык программирования Java. Кроме того, программный код TextMining, доступный по адресу *http://www.textmining.org/*[1], обеспечивает Java-библиотеку для извлечения текста из документов Microsoft Word 97/2000/XP/2003.

Очень важен этап, на котором осуществляется анализ документов. В листингах 2.2 и 2.3, в двух критичных местах в программном коде использован класс `StandardAnalyzer` из библиотеки Lucene, но о нем мы пока еще не говорили. Как видно из рис. 2.3, для извлечения текста из соответствующих документов будут использоваться парсеры, но прежде чем текстовый контент будет проиндексирован, его обрабатывает анализатор Lucene. Работа анализатора крайне важна, поскольку именно анализаторы отвечают за разметку (tokenizing) текста, который будет индексирован. Это значит, что какие-то слова из текста, которые анализаторы сочтут важными, они сохранят, но одновременно с этим проигнорируют все остальное. Если на этапе анализа вы игнорируете что-то, представляющее для вас интерес, после этого вам никогда не удастся обнаружить эти пропущенные данные в процессе поиска, как бы ни был сложен ваш алгоритм индексирования.

Разумеется, анализаторы не в состоянии самостоятельно выбрать за вас соответствующие поля. В качестве примера в листинге 2.2 мы явно определили четыре интересующих нас поля. Класс `StandardAnalyzer` обработает поле `content` – единственное индексируемое поле. Этот анализатор, используемый по умолчанию, является встроенным анализатором Lucene самого общего назначения. Он, помимо прочего, обеспечивает интеллектуальную разметку текста, состоящего из букв и цифр, разумно размечает акронимы, названия фирм, адреса электронной почты, имена компьютерных хостов и даже иероглифы CJK (китайские, японские и корейские).

В последней версии библиотеки Lucene (2.3 на момент написания этой книги) используется лексический анализатор на языке Java – JFlex (*http://jflex.de/*). Lucene-класс `StandardTokenizer` – это генератор меток (tokenizer), работа которого основана на грамматике, – для его создания был использован анализатор JFlex; класс `StandardTokenizer` используется в классе `StandardAnalyzer`. Чтобы убедиться в важности анализаторов, замените анализатор `StandardAnalyzer` анализатором `WhitespaceAnalyzer` и посмотрите, как отличаются полученные в результате оценки. Анализаторы из библиотеки Lucene обеспечивают массу возможностей, например добавление синонимов и модификацию стоп-слов (это слова, которые явно удаляются из текста перед индексированием), а кроме

[1] На момент подготовки книги в печать данный сайт был недоступен, но вы можете ознакомиться с материалом здесь: *http://kickjava.com/src/org.textmining.text.extraction.index.htm*. – *Прим. науч. ред.*

того позволяют работать не только с английским, но и с другими языками. На протяжении всей книги мы будем пользоваться анализаторами из библиотеки Lucene – даже в тех главах, материал которых не связан с поиском. Когда мы имеем дело с документами, исключительную важность имеет общее представление об идентификации уникальных характеристик текстового описания. Соответственно, анализаторы оказываются очень уместными в таких областях, как разработка спам-фильтров, выработка рекомендаций на основе текста, в бизнес-приложениях, приложениях для налогообложения и так далее.

Этап индексирования с применением библиотеки Lucene совершенно прозрачен для конечного пользователя, но при этом обладает широким набором возможностей. В одном индексе можно иметь Lucene-объекты `Document`, соответствующие различным предметным сущностям (таким как сообщения электронной почты, памятки, юридические документы) и, следовательно, характеризуемые различными полями. Объекты `Document` можно также удалять из индекса или обновлять в индексе. Еще одна интересная возможность индексирования Lucene – *повышение степени важности* (boosting). Повышение степени важности позволяет пометить определенные документы как более важные или менее важные, чем другие. В метод `indexDocument`, код которого приводится в листинге 2.2, можно добавить предложение вроде следующего:

```
if ( parsedDoc.getDocumentId().equals("g1-d14")) {
        doc.setBoost(2);
}
```

Это предложение можно обнаружить в программном коде, где оно закомментировано и помечено как «ToDo». Если удалить символы комментария, скомпилировать этот код и снова выполнить сценарий из листинга 2.1, то вы заметите, что документ, который был последним, окажется первым. Повышение степени важности привело к повышению – фактически, двукратному – оценки каждого поля `Field` для этого документа. Можно также применить повышение к отдельным полям `Field`, чтобы добиться более точных результатов повышения степени важности.

Поиск с помощью библиотеки Lucene нельзя упростить. Как вы видели, если использовать нашу оболочку `MySearcher`, все дело сводится к двум строкам кода. Хотя в примере из листинга 2.1 мы использовали просто одно слово, библиотека Lucene обеспечила изощренный синтаксический разбор выражения запроса посредством класса `QueryParser`. Не исключено, что иногда вам придется создавать Lucene-объект `Query` другими средствами. Чтобы найти выражение «nasdaq index» и обеспечить возможность получения результатов, имеющих отношение к выражению «nasdaq composite index», пришлось бы использовать класс `PhraseQuery`. В этом случае слово «index» не может существовать отдельно от слова «nasdaq». Максимальное количество слов, разделяющих слова

«nasdaq» и «index», задается в параметре, называемом *slope*. Если присвоить этому параметру значение 1, можно добиться желаемого результата. Чтобы получить представление об этой и более мощных функциональных возможностях поиска с помощью библиотеки Lucene, советуем вам изучить API-интерфейсы и документацию Lucene.

2.2. Зачем нужен поиск вне индексов?

Теперь, когда мы показали, как можно быстро индексировать документы с помощью библиотеки Lucene и выполнять запросы к полученным индексам, вы, вероятно, убедились в том, что пользоваться библиотекой Lucene легко и приятно. Вы можете спросить: «Если библиотека Lucene так интеллектуальна и эффективна, почему надо беспокоиться о чем-то еще?» В этом разделе мы покажем, зачем нужен поиск вне индексов. Мы уже говорили об этих причинах в главе 1, но в данном разделе рассмотрим вопрос глубже. Давайте добавим в список первичных URL-адресов новый документ. Листинг 2.4 похож на листинг 2.1, но теперь он включает URL-адрес страницы, которая содержит спам.

Листинг 2.4. Чтение, индексирование и поиск веб-страниц со спамом

```
FetchAndProcessCrawler crawler =
➥  new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);

crawler.setDefaultUrls();

crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-01.html");   ← Добавление веб-страницы со спамом
crawler.run();

LuceneIndexer luceneIndexer =
➥  new LuceneIndexer(crawler.getRootDir());                 ← Построение индекса Lucene

luceneIndexer.run();

MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());   ← Создание простой поисковой системы

oracle.search("armstrong",5);
```

На рис. 2.4 показаны результаты поиска по слову «Armstrong». Видно, что искусно подготовленная веб-страница со спамом заняла первое место в нашей классификации. Вы можете создать три или больше аналогичных страниц со спамом и добавить их в список URL-адресов, чтобы убедиться в этом самостоятельно: настоящий релевантный контент довольно скоро затеряется в море страниц со спамом!

В отличие от набора документов, хранящихся в базе данных или на вашем жестком диске, контент Веба не регламентируется. Следовательно, намеренное создание мошеннических веб-страниц способно превратить традиционные методики информационного поиска в практически бесполезные. Если поисковые системы полагаются лишь на традиционные методики информационного поиска, то «прогулка» по Интернету

```
bsh % oracle.search("armstrong",5);

Search results using Lucene index scores:
Query: armstrong

Document Title: Cheap medicine--low interest loans
Document URL: file:/c:/iWeb2/data/ch02/spam-01.html --> Relevance
Score: 0.591894507408142

_____________________________________________________
Document Title: Lance Armstrong meets goal in painful marathon debut
Document URL: file:/c:/iWeb2/data/ch02/sport-01.html  -->
Relevance Score: 0.370989531278610

_____________________________________________________
Document Title: New York 'tour' Lance's toughest
Document URL: file:/c:/iWeb2/data/ch02/sport-03.html -->
Relevance Score: 0.291807949542999

_____________________________________ _________________
Document Title: New York City Marathon
Document URL: file:/c:/iWeb2/data/ch02/sport-02.html -->
Relevance Score: 0.210920616984367

_____________________________________________________

bsh %
```

Рис. 2.4. *Всего лишь одна обманная веб-страница существенно изменила ранжирование результатов, полученных для запроса «Armstrong»*

в целях обучения или ради развлечений – наш национальный онлайн-спорт – была бы невозможна. Добро пожаловать в новый дивный мир *оценки ссылок* (link analysis)! Оценка ссылок стала первым (и значительным) усовершенствованием быстрого и точного поиска в наборе явно связанных друг с другом документов, таких как веб-страницы Интернета. Эта методика обеспечила эволюцию фирмы Google от полной неизвестности к мировому господству в области поиска, а также развитие многих других направлений в исследованиях и разработке.

Оценка ссылок – это структурная характеристика Интернета. Другая его характеристика, поведенческая, – *анализ экранных данных* (user click analysis) . Вкратце, анализом экранных данных называется запись переходов пользователя по ссылкам при навигации по найденным страницам и последующая обработка этих записей для уточнения ранжирования результатов, получаемых для конкретного пользователя. Данный анализ основан на том исходном допущении, что если вы ищете какое-то выражение и находите релевантную (с точки зрения своих критериев) страницу, то почти наверняка перейдете на эту страницу. И наоборот, не зайдете на страницы, не релевантные вашему выражению поиска и не

отвечающие *цели поиска* (search intention). Мы обращаем особое внимание на этот термин, поскольку он знаменует отход от традиционных приложений, где ответ системы зависел только от входных данных, введенных самим пользователем. Если приложение может выявить ваши намерения, значит, оно достигло главной вехи на пути к обретению интеллекта – способности изучить предпочтения пользователя без помощи программиста, вводящего ответ с «черного хода».

2.3. Уточнение результатов поиска на основе анализа ссылок

Продолжая попытки выполнить поиск без индексирования, рассмотрим алгоритм оценки ссылок PageRank, который принес успех поисковой системе Google. Алгоритм PageRank был обнародован в 1998 году, на седьмой международной конференции World Wide Web (WWW98), Сергеем Брином и Ларри Пейджем в статье, озаглавленной «The anatomy of a large-scale hypertextual Web search engine». Примерно в это же время Джон Клейнберг (Jon Kleinberg) из исследовательского центра IBM Almaden разработал алгоритм *Hypertext Induced Topic Search (HITS)*. Оба алгоритма являются моделями оценки ссылок, хотя алгоритм HITS не добился коммерческого успеха, выпавшего на долю алгоритма PageRank.

В этом разделе мы представим основные идеи алгоритма PageRank, а также механизм вычисления оценок ранжирования. Мы также проанализируем так называемый *механизм телепортации* (teleportation mechanism) и внутренние механизмы *степенного метода* (power method), составляющего основу алгоритма PageRank. В заключение продемонстрируем объединение индексных оценок с оценками алгоритма PageRank для уточнения результатов поиска.

2.3.1. Алгоритм PageRank

Ключевая идея алгоритма PageRank – рассматривать гиперссылки, ведущие с одной страницы на другую, как рекомендации или подтверждения. Следовательно, чем больше на странице подтверждений, тем выше должна быть степень ее важности. Другими словами, если на некую веб-страницу ссылаются другие страницы, которые являются важными, эта страница также является важной. Минуточку! Если для того, чтобы определить важные страницы, необходимо знать, какие страницы являются важными, как тогда это работает? Давайте возьмем конкретный пример и подробно рассмотрим его.

На рис. 2.5 показан направленный граф со всеми нашими примерами веб-страниц, имена файлов которых начинаются с префикса `biz`. Заголовки этих статей и имена их файлов приведены в табл. 2.1.

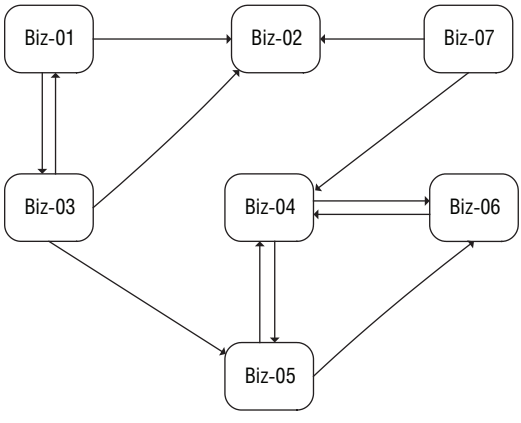

Рис. 2.5. *Направленный граф, который отображает взаимосвязь между веб-страницами «biz»*

Таблица 2.1. Документы с новостями бизнеса и их связи (см. также рис. 2.5)

Заголовок	**Имя файла**	**Ссылается на**
Google Expands into Newspaper Ads	biz-01.html	biz-02, biz-03
Google's Sales Pitch to Newspapers	biz-02.html	(Внешней ссылки нет: *висячий узел*)
Google Sells Newspaper Ads	biz-03.html	biz-01, biz-02, biz-05
NVidia Now a Supplier for MP3 Players	biz-04.html	biz-05, biz-06
Nvidia Shares Up on PortalPlayer Buy	biz-05.html	biz-04, biz-06
Chips Snap: Nvidia, Altera Shares Jump	biz-06.html	biz-04
Economic Stimulus Plan Helps Stock Prices	biz-07.html	biz-02, biz-04

Если на веб-странице A есть ссылка на веб-страницу B, в графе имеется стрелка от A к B. Опираясь на этот рисунок, введем понятия *матрицы гиперссылок* (hyperlink matrix) H и вектора-строки p (вектор PageRank). Считайте, что матрица H – не что иное, как таблица (двумерный массив), а вектор – одномерный массив в языке Java. Каждая строка матрицы H формируется как результат подсчета количества всех внешних ссылок, имеющихся на странице Pi, скажем $N(i)$, и присваивания элементу в столбце j значения $1/N(i)$, если на странице Pi есть внешняя ссылка на страницу Pj, или значения 0 в противном случае. Так, для графа, изображенного на рис. 2.5, наша матрица H выглядела бы как табл. 2.2.

Таблица 2.2. Матрица H для страниц с новостями бизнеса и их связей (см. также рис. 2.5)

0	1/2	1/2	0	0	0	0
0	0	0	0	0	0	0
1/3	1/3	0	0	1/3	0	0
0	0	0	0	1/2	1/2	0
0	0	0	1/2	0	1/2	0
0	0	0	1	0	0	0
0	1/2	0	1/2	0	0	0

Следует обратить внимание на две вещи:

- В матрице очень много нулей – такие матрицы мы называем *разреженными* (sparse). Здесь нет ничего плохого – как ни странно, это хорошая вещь. Это следствие того факта, что, как правило, веб-страница содержит ссылки лишь на небольшое число других страниц – небольшое относительно количества всех веб-страниц в Интернете. Разреженные матрицы приветствуются, потому что если реализовать их аккуратно, можно высвободить массу оперативной памяти и сэкономить время вычислений.
- Все значения в этой матрице меньше или равны 1. Оказывается, это очень важно. Существует связь между «случайным» интернет-серфером, которого представили себе Брин и Пейдж (см. раздел 2.3.2), и теорией вероятности перехода матриц, также известной как *теория цепей Маркова*. Эта связь гарантирует наличие у данного алгоритма некоторых нужных свойств.

2.3.2. Вычисление вектора PageRank

Алгоритм PageRank вычисляет вектор p по следующей итеративной формуле:

$$p(k+1) = p(k) \times H$$

Значениями вектора p являются оценки PageRank, полученные для каждой страницы в графе. Вначале у вас есть набор исходных значений, например $p(0) = 1/n$, где n – число страниц в графе, и по указанной формуле вы получаете значение $p(1)$, затем $p(2)$ и так далее, пока разность между двумя последовательными векторами PageRank не будет выражена достаточно малой величиной: эту произвольно малую величину называют также *критерием сходимости* (convergence criterion), или *порогом* (threshold). Описанный итеративный метод является *степенным методом* (power method), примененным к матрице H. По существу, это и есть алгоритм PageRank.

По техническим причинам – *сходимость* (convergence) итераций к *уникальному* (unique) вектору PageRank – матрица H замещается другой матрицей, которая обычно обозначается G (матрица Google) и обладает лучшими математическими свойствами. Мы не будем здесь подробно рассматривать математический аппарат алгоритма PageRank, но чтобы вы лучше представили происходящее, логически обоснуем сам алгоритм и проблемы, заставляющие нас изменять матрицу.

Алгоритм PageRank начинается с воображаемого пользователя, который «случайным» образом путешествует по Интернету. Наш веб-серфер может начать свой путь с любой из данных веб-страниц, содержащих внешние ссылки. Оттуда, следуя по одной из предоставленных внешних ссылок, он прибывает на другую страницу. Затем, для следующего перемещения, серфер выбирает новую внешнюю ссылку и так далее. После нескольких щелчков мышью на ссылках и переходов по графу доля времени, проведенного нашим веб-серфером на конкретной странице, рассматривается как мера относительной важности этой страницы среди других страниц, присутствующих в графе. Если перемещения веб-серфера носят действительно случайный характер – без явных предпочтений, – он посетит страницы, на которые указывают другие страницы, что повысит степень важности посещаемых страниц. Все это хорошо и понятно, но есть две проблемы.

Первая проблема заключается в том, что в Интернете встречаются некоторые страницы, которые не указывают ни на какие другие страницы: в нашем примере это веб-страница `biz-02` (см. рис. 2.5). Мы называем такие страницы графа *висячими узлами* (dangling nodes). Висячие узлы создают проблему, поскольку ограничивают перемещение веб-серфера: в отсутствие внешних ссылок идти некуда! Такие узлы соответствуют строкам матрицы H, имеющим нулевые значения во всех элементах. Чтобы разрешить эту проблему, мы вводим понятие случайного *скачка* (jump), означающее, что как только наш серфер добрался до висячего узла, он может перейти в адресную строку броузера и ввести там URL-адрес любой из страниц графа. В терминологии матрицы H это соответствует замене всех нулевых значений (в строке висячих узлов) значением $1/n$, где n – число страниц в графе. Формально такая коррекция матрицы H называется *стохастической поправкой* (stochasticity adjustment).

Вторая проблема заключается в том, что иногда наш серфер может утомиться, или его отвлекут, и он может «перескочить» на другую страницу, не следуя структуре связей веб-страниц: эквивалент телепортирующего луча из сериала «Звездный путь». Чтобы учесть такие произвольные «скачки», мы вводим новый параметр, который в нашем программном коде называется `alpha`. Данный параметр определяет время, в течение которого серфер будет бродить по просторам Интернета,

следуя по ссылкам, а не перескакивая произвольно с одной страницы на другую: это значение иногда называют *коэффициентом затухания* (damping factor). Формально такая коррекция матрицы H называется *поправкой примитивности* (primitivity adjustment).

В программном коде вы обнаружите явные ссылки на эти две проблемы. Вам не надо беспокоиться по поводу математических подробностей, но если они вас интересуют, превосходный справочный материал содержится в книге «Google's PageRank and Beyond: The Science of Search Engine Rankings» [Langville, A. N. and C. D. Meyer]. Итак, давайте перейдем к действиям и получим матрицу H, выполнив некоторый программный код. В листинге 2.5 показано, как загрузить только те веб-страницы, которые принадлежат категории новостей бизнеса, и вычислить соответствующий им вектор PageRank.

Листинг 2.5. Вычисление вектора PageRank

```
FetchAndProcessCrawler crawler =
➥  new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);

crawler.setUrls("biz");   ← Загрузка веб-страницы с бизнес-новостями
crawler.run();

PageRank pageRank = new PageRank(crawler.getCrawlData()); ← Создание экземпляра объекта PageRank
pageRank.setAlpha(0.8);

pageRank.setEpsilon(0.0001);

pageRank.build();         ← Поиск значения PageRank
```

На рис. 2.6 показан снимок экрана с полученными результатами. Наименее релевантной является страница *biz-07.html*; самая важная страница, согласно вектору PageRank, – *biz-04.html*. Для каждой страницы мы вычислили степень ее релевантности, которая не зависит от выражения поиска! Мы вычислили значения вектора PageRank для нашей сети веб-страниц.

2.3.3. alpha: эффект телепортации между веб-страницами

Давайте заменим значение `alpha`, равное 0,8, каким-то другим из интервала от 0 до 1, чтобы на практике наблюдать, как влияет эффект телепортации между веб-страницами на значения вектора PageRank. По мере того как значение коэффициента `alpha` приближается к нулю, значения вектора PageRank для всех страниц стремятся к 1/7 (приблизительно это десятичное значение 0,142857), именно эту величину вы и ожидали бы получить, потому что наш серфер выбирает свой очередной пункт назначения, не руководствуясь ссылками, а случайным об-

```
Iteration: 8,   PageRank convergence error:
1.4462733376210263E-4
Index: 0 -->  PageRank: 0.03944811976367004
Index: 1 -->  PageRank: 0.09409188129468615
Index: 2 -->  PageRank: 0.32404719855854225
Index: 3 -->  PageRank: 0.24328037107628753
Index: 4 -->  PageRank: 0.18555028886849476
Index: 5 -->  PageRank: 0.05593157626783124
Index: 6 -->  PageRank: 0.061816733771795335

 Iteration: 9,   PageRank convergence error:
5.2102415715682415E-5
Index: 0 -->  PageRank: 0.039443819850858625
Index: 1 -->  PageRank: 0.09407831778282823
Index: 2 -->  PageRank: 0.3240636997004271
Index: 3 -->  PageRank: 0.24328782624042117
Index: 4 -->  PageRank: 0.18555238603685822
Index: 5 -->  PageRank: 0.0559269660757835
Index: 6 -->  PageRank: 0.06181315844717868

_______________  Calculation Results  _______________
Page URL: file:/c:/iWeb2/data/ch02/biz-04.html  -->  Rank:
0.324063699700427
Page URL: file:/c:/iWeb2/data/ch02/biz-06.html  -->  Rank:
0.243287826240421
Page URL: file:/c:/iWeb2/data/ch02/biz-05.html  -->  Rank:
0.185552386036858
Page URL: file:/c:/iWeb2/data/ch02/biz-02.html   -->  Rank:
0.094078317782828
Page URL: file:/c:/iWeb2/data/ch02/biz-03.html   -->  Rank:
0.061813158447179
Page URL: file:/c:/iWeb2/data/ch02/biz-01.html   -->  Rank:
0.055926966075784
Page URL: file:/c:/iWeb2/data/ch02/biz-07.html   -->  Rank:
0.039443819850859
_____________________________________________________
```

Рис. 2.6. *Вычисление вектора PageRank для небольшой сети веб-страниц с новостями бизнеса*

разом. С другой стороны, по мере приближения значения `alpha` к единице значения вектора PageRank будут сходиться к вектору PageRank, который соответствует серферу, скрупулезно следующему структуре ссылок.

По мере приближения значения `alpha` к единице вы должны заметить еще один эффект – увеличение числа итераций, необходимых для до-

стижения сходимости. Фактически, для нашей небольшой сети веб-страниц действительна табл. 2.3 (мы поддерживаем для показателя устойчивости к ошибкам значение, равное 10^{-10}).

Таблица 2.3. Влияние увеличения значения параметра alpha на количество итераций для набора веб-страниц «biz»

Значение alpha	Число итераций
0,50	13
0,60	15
0,75	19
0,85	23
0,95	29
0,99	32

Как видим, количество итераций быстро увеличивается по мере возрастания значения `alpha`. Для случая семи веб-страниц эффект практически ничтожен, но для 8 миллиардов страниц (столько примерно страниц использует поисковая система Google) тщательный подбор значения `alpha` крайне важен. В сущности, выбор значения `alpha` – это компромисс между строгим следованием веб-структуре и эффективностью вычислений. Говорят, в поисковой системе Google для `alpha` используется значение 0,85. Значение в интервале между 0,7 и 0,9 должно обеспечивать вашему приложению хороший баланс производительности и результативности, в зависимости от природы графа и особенностей пользовательского просмотра (browsing).

Есть методики, способные обеспечить более быструю сходимость степенного метода; и есть также методики, которые вообще не полагаются на степенной метод, – так называемые *прямые методы* (direct methods). Последние лучше подходят для некрупных сетей (вроде типичной локальной сети) и больших значений `alpha` (например, 0,99). Если вы хотите изучить эти методы более подробно, мы приводим ссылки в разделе 2.10.

2.3.4. Основные сведения о степенном методе

Давайте подробнее проанализируем программный код, который обеспечивает вычисление значений вектора PageRank. В листинге 2.6 показан фрагмент кода, отвечающего за вычисление матрицы H на основе данных о ссылках: он принадлежит классу `iweb2.ch2.ranking.PageRankMatrixH`.

Листинг 2.6. Вычисление матрицы H на основе данных о ссылках между веб-страницами

```
public void addLink(String pageUrl) {
    indexMapping.getIndex(pageUrl);
}

public void addLink(String fromPageUrl,
➥   String toPageUrl, double weight) {

    int i = indexMapping.getIndex(fromPageUrl);
    int j = indexMapping.getIndex(toPageUrl);

    try {

         matrix[i][j] = weight;

    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("fromPageUrl:" + fromPageUrl
➥      + ", toPageUrl: " + toPageUrl);
    }
 }

public void addLink(String fromPageUrl, String toPageUrl) {
       addLink(fromPageUrl, toPageUrl, 1);
    }

public void calculate() {

    for(int i = 0, n = matrix.length; i < n; i++) {

       double rowSum = 0;

       for(int j = 0, k = matrix.length; j < k; j++) {
           rowSum += matrix[i][j];
       }

       if( rowSum > 0 ) {

           for(int j = 0, k = matrix.length; j < k; j++) {
              if( matrix[i][j] > 0 ) {

                  matrix[i][j] =
➥     (double)matrix[i][j] / (double) rowSum;
              }
           }
        } else {

           numberOfPagesWithNoLinks++;
        }
    }
}
```

❶ Присвоение исходных значений

❷ Вычисление субстохастической версии матрицы

```
/**
  * Висячий узел соответствует веб-странице без внешних ссылок. Такие узлы
  * в результате дают строку матрицы H, все значения которой равны 0.
  */
public int[] getDangling() {   ← ❸ Обработка вхождений висячих узлов

   int   n = getSize();
   int[] d = new int[n];
   boolean foundOne = false;

   for (int i=0; i < n; i++) {

     for (int j=0; j < n; j++) {

     if (matrix[i][j] > 0) {

        foundOne = true;
        break;
     }
      }

      if (foundOne) {
        d[i] = 0;
      } else {
        d[i] = 1;
      }

      foundOne = false;
   }
   return d;
}
```

❶ Методы `addLink` позволяют присвоить исходные значения переменной `matrix` в зависимости от наличия ссылок между страницами.

❷ В методе `calculate` подсчитывается сумма всех весовых коэффициентов в строке (внешние ссылки) и существующие значения заменяются их взвешенными эквивалентами. После этого, если суммировать все элементы в строке, полученный результат должен быть равен 1 для каждого узла, который не является висячим. Это – субстохастический вариант исходной матрицы.

❸ Висячие узлы обрабатываются отдельно, поскольку у них нет внешних ссылок. Метод `getDangling()` позволяет определить строки, соответствующие висячим узлам, и возвращает вектор висячих узлов.

Вспомните, окончательную структуру матрицы мы поделили на три части: доля основных ссылок, доля висячих узлов и доля телепортаций. Посмотрим, как их можно объединить, чтобы получить окончательные значения матрицы, которыми мы будем пользоваться для получения оценок PageRank. В листинге 2.7 показан программный код, который

отвечает за сборку этих составляющих и выполнение вычислений степенного метода. Этот программный код можно найти в классе `iweb2.ch2.ranking.Rank`.

Листинг 2.7. Применение степенного метода для вычисления вектора PageRank

```
public void findPageRank(double alpha, double epsilon) {

  // Счетчик итераций
  int k = 0;

  // Вспомогательная переменная
  PageRankMatrixH matrixH = getH();

  // Матрица H имеет размеры nxn, а размер вектора PageRank - n
  int n = matrixH.getSize();7

  // Вспомогательная переменная - величина, обратная n
  double inv_n = (double)1/n;

  // Это сама матрица n x n с значениями типа double
  double[][] H = matrixH.getMatrix();

  // Служебная переменная, которая содержит ошибку, она получает
  // произвольное значение 1
  double error = 1;

  // Эта переменная содержит значения вектора PageRank
  pR = new double[n];

  // Копия вектора PageRank, полученного на предыдущей итерации.
  // Она нужна только для того, чтобы вычислить ошибку
  double[] tmpPR = new double[n];

  // Задаем исходные значения (подходящие для данного случая)
  for (int i=0; i < n; i++) {
    pR[i] = inv_n;
  }

  // Разд. 2.3 -- Видоизменение матрицы H: висячие узлы

  double[][] dNodes= getDanglingNodeMatrix();

  // Разд. 2.3 -- Видоизменение матрицы H: телепортация

  double tNodes=(1 - alpha) * inv_n;

  //Замещение матрицы H матрицей G
  for (int i=0; i < n; i++) {
    for (int j=0; j < n; j++) {
      H[i][j] = alpha*H[i][j] + dNodes[i][j] + tNodes;
    }
  }
```

```
    // Повторяем цикл до тех пор, пока не получим сходимость!
    // Если ошибка меньше epsilon, значит, мы нашли значения PageRank
    while ( error >= epsilon) {

      // Прежде чем обновлять вектор PageRank, создадим его копию
      for (int i=0; i < n; i++) {
        tmpPR[i] = pR[i];
      }

      double dummy =0;

      // Теперь мы получили следующую точку в итерации
      for (int i=0; i < n; i++) {

        dummy =0;

        for (int j=0; j < n; j++) {
          dummy += pR[j]*H[j][i];
        }
        pR[i] = dummy;
      }

      // Получим ошибку, чтобы можно было проверить сходимость
      error = norm(pR,tmpPR);

      // Увеличиваем значение счетчика на единицу
      k++;
    }

    // Вывод итоговых значений
    System.out.println(
➥   "\n_____________  Calculation Results  _____________\n");
    for (int i=0; i < n; i++) {
      System.out.println("Page URL: "+
➥     matrixH.getIndexMapping().getValue(i)+"  -->  Rank: "+pR[i]);
    }
  }
```

Учитывая важность метода `search`, мы сделали все возможное, чтобы максимально облегчить его чтение. Мы удалили некоторые элементы документирования `Javadoc`, связанные с разделом «Сделать» («ToDo»), но в остальном этот сниппет не претерпел изменений. Итак, мы начинаем с того, что, основываясь на наличии ссылок, получаем значения матрицы H, а затем инициализируем вектор PageRank. После этого мы получаем долю висячих узлов и долю телепортаций. Обратите внимание: для висячих узлов необходим полный двумерный массив, тогда как для доли телепортаций достаточно одной переменной типа `double`. Получив все три компонента, суммируем их значения. Это наиболее эффективный способ подготовки данных для степенного метода, но вместо полных двумерных массивов вы должны использовать разреженные матрицы; эта модернизация описана в одном из пунктов раздела «Сделать» в конце этой главы.

После того как вычислена новая матрица H, приступаем к вычислениям степенного метода – это программный код в цикле `while`. Мы знаем, что значения вектора PageRank будут получены тогда, когда наша ошибка окажется меньше, чем произвольно малое значение `epsilon`. Вы можете спросить: «Что, если я изменю значение `epsilon`? Изменятся ли значения вектора PageRank? Если так, каким должно быть значение `epsilon`?» Давайте разберем эти вопросы по очереди. Во-первых, пусть, скажем, ошибка вычисляется как абсолютное значение почленной разности между новым и старым векторами PageRank. В листинге 2.8 показан метод `norm` класса `iweb2.ch2.ranking.Rank`, который оценивает ошибку.

Листинг 2.8. Оценка ошибки для двух последовательных векторов PageRank

```
private double norm(double[] a, double[] b) {

  double norm = 0;

  int n = a.length;

  for (int i=0; i < n; i++) {
     norm += Math.abs(a[i]-b[i]);
  }

  return norm;
}
```

Если выполнить этот программный код несколько раз или внимательно рассмотреть рис. 2.6, вы поймете, что к моменту сходимости значения вектора PageRank меняют цифру, которая соответствует наименьшему `epsilon`. Следовательно, значение `epsilon` должно быть достаточно малым, чтобы позволить нам отличить все веб-страницы друг от друга в соответствии с значениями вектора PageRank. Если у нас 100 страниц, то достаточным должно быть значение `epsilon`, равное 0,001. Если мы берем весь Интернет, около 10^{10} веб-страниц, то для `epsilon` необходимо малое значение порядка 10^{-10}.

2.3.5. Объединение оценок индексирования и оценок PageRank

Теперь, рассмотрев реализацию алгоритма PageRank, мы готовы показать, как можно объединить оценки результатов поиска, полученные средствами библиотеки Lucene, с оценками релевантности страниц по алгоритму PageRank. Воспользуемся теми же семью веб-страницами, которые относятся к новостям бизнеса, но на этот раз добавим к ним еще три страницы, содержащие спам (назовем их *spam-biz-0x.html*, где *x* заменяется числовым значением). Страницы со спамом обманут поиск на основе индексирования, но не обманут алгоритм PageRank.

Выполним этот сценарий и посмотрим, что происходит. В листинге 2.9 показано, как это сделать:

- Загружаем относящиеся веб-страницы с новостями бизнеса, как мы это делали раньше.
- Добавляем три страницы со спамом: по одной для каждой темы.
- Индексируем все страницы.
- Формируем вектор PageRank.
- Вычисляем гибридную рейтинговую оценку, которая включает как индексную оценку релевантности (от библиотеки Lucene), так и оценку PageRank.

Листинг 2.9. Объединение оценок Lucene и PageRank для ранжирования веб-страниц

```
FetchAndProcessCrawler crawler =
➥new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);

crawler.setUrls("biz");

crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-01.html");   ← Добавление страниц, содержащих спам
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-02.html");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-03.html");
crawler.run();

LuceneIndexer luceneIndexer =
➥new LuceneIndexer(crawler.getRootDir());

luceneIndexer.run();                                   ← Индексирование всех страниц

PageRank pageRank = new PageRank(crawler.getCrawlData());
pageRank.setAlpha(0.99);
pageRank.setEpsilon(0.00000001);
pageRank.build();                                      ← Формирование PageRank

MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());

oracle.search("nvidia",5, pageRank);                   ← Поиск с применением гибридной оценки
```

Результаты поиска слова «nvidia» показаны на рис. 2.7. Сначала выводится результирующий набор, полученный только средствами библиотеки Lucene, затем выводятся результаты, полученные заново с учетом значений вектора PageRank. Как видите, мы не лишены спамерских способностей! Мошенническая страница вышла на первое место в результирующем наборе, который мы получили, когда использовали только средства библиотеки Lucene. Но когда мы применили гибридное ранжирование, первыми оказались самые релевантные страницы. Страница со спамом опустилась в бездну нерелевантности, где ей и место! Вы только что написали свою первую Google-подобную поисковую систему. Поздравляем!

```
bsh % oracle.search("nvidia",5,pr);

Search results using Lucene index scores:
Query: nvidia

Document Title: NVIDIA shares plummet into cheap medicine for you!
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.html -->
Relevance Score: 0.519243955612183

_____________________________________________________
Document Title: Nvidia shares up on PortalPlayer buy
Document URL: file:/c:/iWeb2/data/ch02/biz-05.html
Relevance Score: 0.254376530647278

_____________________________________________________
Document Title: NVidia Now a Supplier for MP3 Players
Document URL: file:/c:/iWeb2/data/ch02/biz-04.html   -->
Relevance Score: 0.190782397985458

_____________________________________________________
Document Title: Chips Snap: Nvidia, Altera Shares Jump
Document URL: file:/c:/iWeb2/data/ch02/biz-06.html   -->
Relevance Score: 0.181735381484032

Document Title: Economic stimulus plan helps stock prices
Document URL: file:/c:/iWeb2/data/ch02/biz-07.html   -->
Relevance Score: 0.084792181849480

_____________________________________________________
Search results using combined Lucene scores and page rank scores:
Query: nvidia

Document URL: file:/c:/iWeb2/data/ch02/biz-04.html   -->
Relevance Score: 0.087211910261991
Document URL: file:/c:/iWeb2/data/ch02/biz-06.html   -->

Document URL: file:/c:/iWeb2/data/ch02/biz-05.html   -->
Relevance Score: 0.062737066556678
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.html  -->

Document URL: file:/c:/iWeb2/data/ch02/biz-07.html    -->
Relevance Score: 0.000359708275446

_____________________________________________________
```

Рис. 2.7. *Объединение оценок Lucene и оценок PageRank позволяет избавиться от спама*

Программный код, который обеспечивает объединение этих двух оценок, можно найти в классе `MySearcher`, в переопределенном методе `search`, который использует ссылку на объект класса `PageRank` в качестве аргумента. Фрагмент кода, приведенный в листинге 2.10, взят из этого метода, и именно в нем осуществляется объединение двух оценок.

Листинг 2.10. Объединение оценок Lucene и PageRank

```
double m = 1 - (double) 1/pR.getH().getSize();   ← Вычисление масштабного коэффициента

for (int i = 0; i < numberOfMatches; i++) {

  url = docResults[i].getUrl();

  double hScore = docResults[i].getScore()
➥   *Math.pow(pR.getPageRank(url),m);   ← Вычисление гибридной оценки

  docResults[i].setScore(hScore);

  urlScores.put(hScore, url);   ← Создание таблицы соответствия оценок URL-адресам
}
```

Итак, у вас, вполне оправданно, могли возникнуть несколько вопросов. Зачем мы ввели переменную `m`? Почему не взяли среднее значение этих двух оценок? Почему не использовали более сложную формулу для объединения оценки индексирования и оценки PageRank? Хорошие вопросы, а ответы могут вас удивить. Оставив в стороне тот факт, что наша формула удерживает значения оценок в интервале от 0 до 1, все использованные нами варианты – это произвольный выбор. Мы с таким же успехом могли взять произведение этих двух оценок, чтобы объединить их.

Логическим обоснованием для возведения значений вектора PageRank в степень `m` служит тот факт, что небольшое количество проиндексированных нами страниц может стать причиной получения слишком высокой оценки релевантности индексирования для страниц со спамом и тем самым искусственно понизить эффективность алгоритма PageRank. По мере увеличения количества страниц приведенное значение PageRank (вторая составляющая гибридной оценки) стремится к исходному значению PageRank, потому что величина `m` быстро приближается к 1. Мы полагаем, что в небольших сетях такое степенное приведение может способствовать повышению важности структуры ссылок в сравнении с индексом. Эта формула должна хорошо работать как для наборов, состоящих всего лишь из нескольких документов, так и для наборов, насчитывающих множество документов. Существует глубокая математическая связь между степенными зависимостями и графами, подобными Интернету, но мы не будем их здесь рассматривать [см. Adamic и др.]. Логический вывод: если имеете дело с набором из нескольких страниц, и при этом выражение поиска встречается в документе очень часто (как это происходит со спам-страницами), оценка индексированных страниц (число, которое возвращает библиотека Lucene в качестве оценки результатов поиска) будет близка к 1. Поэтому, чтобы уравновесить этот эффект, требуется повторное взвешивание.

2.4. Уточнение результатов поиска на основе анализа экранных данных

В предыдущем разделе мы показали, что оценка ссылок позволяет воспользоваться таким преимуществом Интернета, как его структуризация. В этом разделе мы поговорим о другом способе, позволяющем задействовать природу Интернета, – об анализе экранных данных. Как вам известно, каждый раз, когда пользователь выполняет запрос, он переходит по одной из полученных в результате ссылок или по ссылке, которая покажет следующую страницу с дополнительными результатами, если таковые имеются. В одном случае пользователь обнаружил что-то, представляющее интерес, и перешел по этой ссылке, потому что нашел то, что искал, или потому что этот результат ему любопытен и он хочет проанализировать связанную с ним информацию, чтобы решить, действительно ли это то, что он искал. В другом случае наилучшие результаты оказались не теми, которые были нужны пользователю, и он хочет взглянуть на следующую страницу, просто на всякий случай, – чтобы убедиться, стоит ли эта система поиска хоть копейку!

Шутки в сторону, одна из причин, по которым оценка релевантности является трудной задачей, заключается в том, что релевантность – понятие субъективное. Если вы и я видите результаты, полученные для запроса «elections» (выборы), вас могут интересовать выборы в США, тогда как я могу интересоваться выборами в Великобритании или даже местными выборами в том городе, где я живу. Поисковая система не в состоянии понять цель (или контекст) вашего поиска, если у нее нет дополнительной информации. Следовательно, самые релевантные результаты для одного человека могут отличаться – и довольно часто отличаются – от наиболее релевантных результатов, полученных для другого человека, несмотря даже на то, что условия запроса могут быть идентичными!

Мы собираемся представить анализ экранных данных как способ, позволяющий уточнить результаты поиска для каждого конкретного пользователя. Такое уточнение возможно благодаря алгоритму, который мы позже очень подробно изучим в этой книге, – наивному байесовскому классификатору (NaiveBayes). Мы продемонстрируем возможность объединения оценок индексирования, оценок PageRank и оценок, полученных в результате анализа экранных данных, с целью уточнения результатов поиска.

2.4.1. Первое знакомство с анализом экранных данных

Анализ экранных данных позволяет использовать в качестве входных данные о взаимодействии каждого пользователя с поисковой системой. Аристотель сказал: «Мы есть то, что мы часто делаем», и эта идея является исходной предпосылкой анализа экранных данных: ваше взаи-

модействие с поисковой системой определяет область ваших интересов и вашу индивидуальность. До сих пор мы еще не говорили об интеллектуальной методике, ответственной за *персонализацию* (personalization) веб-приложения. Разумеется, необходимым условием для реализации этой методики является наличие у поисковой системы возможности определить, какие запросы были получены от конкретного пользователя. Другими словами, пользователь обязан пройти процедуру регистрации в вашем приложении или каким-то иным образом зафиксировать сеанс работы с приложением. Должно быть понятно, что наш подход к реализации анализа экранных данных можно применить в каждом приложении, которое в состоянии записать действия пользователя, – этот подход не является специфичным для приложений из области информационного поиска.

Итак, предположим, что вы собрали коллекцию действий пользователей, вроде указанных в файле *user-clicks.csv*; этот файл можно найти в каталоге *data/ch02* вместе с остальными файлами, которые мы используем в этой главе. Наша задача – написать программный код, помогающий нам использовать эти данные, примерно как алгоритм PageRank помог нам воспользоваться информацией о нашей сети. То есть мы хотим использовать эти данные для *персонализации* результатов поиска, соответствующим образом изменив ранжирование на основании того, кем был выдан запрос. В трех полях указанного файла содержатся разделенные запятыми следующие значения:

- Строка, которая идентифицирует пользователя
- Строка, содержащая поисковый запрос
- Строка с url-адресом, который ранее выбрал пользователь, после того как просмотрел результаты, полученные для этого запроса

Если данный пользователь вам неизвестен (отсутствует какая-либо регистрация в системе или сеанс), можно использовать какое-то значение, принятое по умолчанию, например «anonymous» – разумеется, вы должны позаботиться о том, чтобы имя «anonymous» не могло быть допустимым именем пользователя в вашем приложении! Если ваши данные имеют какой-то другой формат, ничего страшного. Предлагаемый нами программный код легко адаптируется для обработки ваших кокретных данных. Чтобы обеспечить персонализацию результатов, необходимо знать пользователя, его вопрос и ссылки, выбранные им ранее по этому вопросу. Если эта информация доступна, пора действовать!

Вы могли заметить, что в наших данных есть несколько записей, относящихся к одному и тому же пользователю и к одному и тому же запросу. Тут нет ничего странного, примерно то же вы должны наблюдать и в своих данных. Смотря по тому, сколько раз встречается в файле *user-clicks.csv* переход по конкретному адресу, этот URL-адрес оказывается лучшим или худшим кандидатом на попадание в результаты поиска.

Как правило, один и тот же пользователь будет переходить по нескольким разным ссылкам, полученным для одного и того же запроса, потому что каждый раз его может интересовать что-то другое или, возможно, он разыскивает дополнительную информацию по теме. Интересным атрибутом, который следует учитывать, является *временная метка* (timestamp). Информация, связанная со временем, может помочь выявить в ваших данных некую непостоянную структуру. Одни переходы пользователя соответствуют периодически используемым шаблонам; другие управляются событиями; третьи совершенно случайны. Временная метка может помочь выявить такие шаблоны или корреляцию с другими событиями.

Посмотрим сначала, как можно получить для запросов персонифицированные результаты. В листинге 2.11 показан наш сценарий, напоминающий листинг 2.9, но на этот раз мы загружаем информацию о действиях пользователя и выполняем тот же запрос «google ads» дважды – один раз для пользователя dmitry, а другой – для пользователя babis.

Листинг 2.11. Учет действий пользователя в результатах поиска

```
FetchAndProcessCrawler crawler =
➥  new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);

crawler.setUrls("biz");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-01.html");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-02.html");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-03.html");
crawler.run();

LuceneIndexer luceneIndexer =
➥  new LuceneIndexer(crawler.getRootDir());

luceneIndexer.run();
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());

PageRank pageRank = new PageRank(crawler.getCrawlData());
pageRank.setAlpha(0.9);
pageRank.setEpsilon(0.00000001);
pageRank.build();

UserClick aux = new UserClick();
UserClick[] clicks =aux.load("C:/iWeb2/data/ch02/user-clicks.csv");   ← Загрузка действий пользователей

TrainingSet tSet = new TrainingSet(clicks);   ← Создание обучающего набора

NaiveBayes naiveBayes = new NaiveBayes("Nanve Bayes", tSet);   ← Определение классификатора

naiveBayes.trainOnAttribute("UserName");   ← Выбор атрибутов
naiveBayes.trainOnAttribute("QueryTerm_1");
naiveBayes.trainOnAttribute("QueryTerm_2");

naiveBayes.train();   ← Обучение классификатора
```

```
oracle.setUserLearner(naiveBayes);
UserQuery dmitryQuery = new UserQuery("dmitry","google ads");

oracle.search(dmitryQuery,5, pageRank);
UserQuery babisQuery = new UserQuery("babis","google ads");
oracle.search(babisQuery,5, pageRank);
```

Первую часть этого сценария вы видели в листинге 2.9. Прежде всего, мы загружаем страницы, в которых хотим искать данные. Затем индексируем эти страницы средствами библиотеки Lucene и формируем вектор PageRank, соответствующий их структуре. Та часть, которая включает новый программный код, берется из класса `UserClick`, представляющего переход указанного пользователя по конкретному URL-адресу. Мы также определили класс `TrainingSet`, в коллекции которого содержатся адреса всех переходов пользователя. Разумеется, вас может интересовать, что плохого случилось с массивом объектов `UserClick`. Почему мы не можем просто воспользоваться этими объектами? Ответ следующий: чтобы определить ссылки, которые с наибольшей вероятностью окажутся желательными для конкретных пользователя и запроса, мы собираемся загрузить переходы пользователя по ссылкам в *классификатор* (classifier), в частности в наивный байесовский классификатор.

2.4.2. Применение наивного байесовского классификатора

Мы много будем обращаться к классификации в главах 5 и 6, но основные положения рассмотрим для ясности здесь. Классификация опирается на ссылочные структуры, разбивающие пространство всех возможных результатов обработки данных на набор классов (их еще называют *категориями* (categories) или *концептами* (concepts)), которые (обычно) не перекрывают друг друга. Мы сталкиваемся с классификацией ежедневно. Из повседневного опыта нам известно, что можно составить список разных блюд в соответствии с меню ресторана: например, салаты, закуски, фирменные блюда, блюда из макарон, морепродукты и так далее. Аналогично, статьи в газете или новостной группе в Интернете классифицируются по темам: политические, спортивные, деловые, международные, развлекательные и так далее. Короче говоря, можно сказать, что алгоритмы классификации позволяют автоматически идентифицировать объекты как принадлежащие тому или иному классу.

В этом разделе мы будем использовать вероятностный классификатор, который реализует алгоритм, известный как *наивный байесовский алгоритм* (naïve Bayes algorithm); нашу реализацию обеспечивает класс `NaiveBayes`. Классификаторы не зависят от объектов `UserClick`, они связаны только с такими объектами, как концепты (`Concepts`), образцы (`Instances`) и атрибуты (`Attributes`). Считайте, что концепты, образцы

и атрибуты – это аналоги каталогов, файлов и атрибутов файлов в файловой системе.

Работа классификатора заключается в том, чтобы назначить концепт для образца: и это – все, что он делает. Чтобы узнать, какой концепт следует назначить конкретному образцу, классификатор считывает тренировочный набор `TrainingSet` – набор образцов `Instances`, которым уже назначен концепт (`Concept`). После загрузки этих образцов классификатор *тренируется* (*trains)*, или *обучается* (learns) сопоставлять концепты (`Concepts`) и образцы (`Instances`), беря за основу назначения из набора `TrainingSet`. Способ тренировки каждого классификатора зависит от типа классификатора.

Мы собираемся использовать наивный байесовский классификатор как средство получения оценки релевантности для конкретного URL-адреса в зависимости от пользователя и отправленного им запроса. Наивный байесовский классификатор хорош тем, что предоставляет некую величину, называемую *условной вероятностью* (conditional probability) X, при условии Y, которая скажет, какова вероятность наблюдения события X при условии, что мы уже наблюдали событие Y. В частности, этот классификатор использует в качестве входных следующие данные:

- Вероятность наблюдения концепта X в общем случае, ее еще называют *априорной* (prior) вероятностью и обозначают $p(X)$.
- Вероятность наблюдения образца Y, если мы случайным образом выбираем образец из числа тех, которым назначен концепт X; эту вероятность также называют *правдоподобием* (likelihood) и обозначают $p(Y \mid X)$.
- Вероятность наблюдения образца Y в общем случае, ее еще называют *подтверждением* (evidence) и обозначают $p(Y)$.

Важнейшая часть классификатора – вычисление вероятности того, что наблюдаемый образец Y принадлежит концепту X; эту вероятность еще называют *апостериорной вероятностью* (posterior probability) и обозначают $p(X \mid Y)$. Вычисление апостериорной вероятности выполняется по следующей формуле (ее называют теоремой Байеса):

$$p(X \mid Y) = p(Y \mid X)\, p(X) \,/\, p(Y)$$

Наивный байесовский классификатор позволяет получить вероятность, с которой пользователь A хочет видеть URL-адрес X при условии, что он отправил запрос Q: в нашем случае, $Y = A + Q$. Другими словами, мы не хотим использовать наивный байесовский классификатор для классификации чего угодно. Мы будем использовать только ту его способность, которая позволяет вычислить показатель релевантности, что в точности совпадает с нашими целями. В листинге 2.12 представлен соответствующий программный код из класса `NaiveBayes`; полное описание этого кода приведено в разделе 5.3.

Листинг 2.12. Оценка релевантности URL-адреса с помощью наивного байесовского классификатора

```
public class NaiveBayes implements Classifier {
  private String name;                                  ❶
  private TrainingSet tSet;                               ❷

  private HashMap<Concept,Double> conceptPriors;     ❸

  protected Map<Concept,Map<Attribute, AttributeValue>> p;     ❹

  private ArrayList<String> attributeList;     ❺

  public double getProbability(Concept c, Instance i) {
    double cP=0;
    if (tSet.getConceptSet().contains(c)) {

     cP = (getProbability(i,c)*getProbability(c))/getProbability(i);     ❻
    } else {

     cP = 1/(tSet.getNumberOfConcepts()+1);     ❼
    }
    return cP;
  }

  public double getProbability(Instance i) {
    double cP=0;

    for (Concept c : getTset().getConceptSet()) {

     cP += getProbability(i,c)*getProbability(c);
    }
    return (cP == 0) ? (double)1/tSet.getSize() : cP;     ❽
  }

  public double getProbability(Concept c) {
    Double trInstanceCount = conceptPriors.get(c);
    if( trInstanceCount == null ) {
        trInstanceCount = 0.0;
    }
    return trInstanceCount/tSet.getSize();     ❾
  }

  public double getProbability(Instance i, Concept c) {
    double cP=1;
    for (Attribute a : i.getAtrributes()) {
     if ( a != null && attributeList.contains(a.getName()) ) {

       Map<Attribute, AttributeValue> aMap = p.get(c);
       AttributeValue aV = aMap.get(a);
       if ( aV == null) {                                 ❿
             cP *= ((double) 1 / (tSet.getSize()+1));
       } else {
          cP *= (double)(aV.getCount()/conceptPriors.get(c));
```

```
            }
        }
    }
    return (cP == 1) ? (double)1/tSet.getNumberOfConcepts() : cP;
  }
}
```

Сначала проанализируем основные моменты в этом листинге.

❶ Это имя для данного экземпляра объекта наивного байесовского классификатора.

❷ Каждому классификатору нужен тренировочный набор. Имя классификатора и его тренировочный набор задаются предварительно на стадии конструирования объекта. После того как экземпляр наивного байесовского классификатора создан, задать его тренировочный набор `TrainingSet` нельзя, но всегда можно получить ссылку на него и добавить экземпляры объектов.

❸ В таблице `conceptPriors` хранятся счетчики для каждого из концептов, которые есть в нашем тренировочном наборе. Можно было бы использовать эту таблицу для хранения не только счетчиков, но и *априорных* вероятностей. Однако мы хотим использовать счетчики повторно, поэтому, ради эффективности вычислений, храним только их: априорные вероятности можно получить путем простого деления.

❹ В переменной `p` хранятся условные вероятности: вероятность наблюдения концепта X при условии, что мы наблюдали образец Y; или, для переходов пользователя по ссылкам, вероятность того, что пользователь A хочет видеть URL-адрес X, при условии, что он выдал запрос Q.

❺ Это список атрибутов, которые классификатор должен учитывать во время тренировки. Образцы из тренировочного набора могут иметь несколько атрибутов, и есть вероятность, что только некоторые из этих атрибутов являются релевантными (глава 5), поэтому мы отслеживаем, какие атрибуты должны быть использованы.

❻ Если мы сталкиваемся с концептом из тренировочного набора, используется вышеупомянутая формула и вычисляется апостериорная вероятность.

❼ Может случиться так, что конкретный образец до этого момента не встречался, поэтому обращение к методу `getProbability(i)` не имело бы смысла. В таком случае мы присваиваем в качестве *апостериорной* (posterior) вероятности какое-то другое обоснованное значение. Целесообразно указать значение, равное частному от деления единицы на число всех известных концептов, если нет данных для назначения более высокой вероятности какому-либо одному концепту. Мы также прибавили бы к полученному числу единицу. Эта модифика-

ция носит произвольный характер и предназначена для того, чтобы уменьшить вероятность, назначенную каждому концепту, особенно если число наблюдаемых концептов невелико. Поразмышляйте над тем, почему и при каких условиях могла бы оказаться полезной такая модификация.

❽ Этот метод класса `NaiveBayes` не является существенным при рассмотрении исключительно проблемы классификации, потому что возвращаемое им значение постоянно для всех концептов. Но в контексте этого примера мы решили его оставить. Вы вольны модифицировать этот программный код, с тем чтобы данный метод возвращал только вычислитель теоремы Байеса; на что похожи полученные вами результаты?

❾ Априорная вероятность для данного концепта *c* оценивается на основе числа появлений этого концепта в тренировочном наборе. Обратите внимание: не встретившимся концептам мы произвольно назначаем нулевую вероятность. Последствия этого решения могут оказаться как хорошими, так и плохими. Если вы обладаете достаточной уверенностью в том, что у вас в тренировочном наборе имеются все связанные концепты, то этот произвольный выбор помогает устранить случайности в данных. В более общем случае, когда вам, возможно, не встретились очень многие концепты, следует заменить нулевое значение чем-то более разумным – частным от деления единицы на общее число всех известных концептов. Какие еще варианты являются, по-вашему, рациональными? Важно ли иметь точную оценку этого количества? Каким бы ни был ответ, попытайтесь дать логическое обоснование вашему решению и, по возможности, максимально его уточнить.

❿ Мы добрались до ядра класса `NaiveBayes`. «Наивным» в теореме Байеса является тот факт, что мы оцениваем правдоподобие наблюдаемого образца `Instance i` как произведение вероятностей наблюдения каждого из значений атрибутов. Это допущение требует, чтобы атрибуты были статистически независимыми. Мы взяли в кавычки слово *наивный* (nanve), потому что наивный байесовский алгоритм очень надежен и находит широкое применение – даже для решения задач, в которых допущение о независимости атрибутов с очевидностью нарушается. Можно показать, что наивный байесовский алгоритм является оптимальным в совершенно противоположном случае – в тех случаях, где существует полностью детерминированная зависимость между атрибутами [см. Rish].

Если вы припомните сценарий из листинга 2.11, в нем был создан тренировочный набор и экземпляр классификатора с этим тренировочным набором, но прежде чем назначить классификатор образцу `MySearcher`, мы делаем следующие две вещи:

- Сообщаем классификатору, какие атрибуты должны быть учтены в целях тренировки.
- Указываем классификатору на то, что он должен тренироваться на наборе пользовательских переходов по ссылкам, который мы только что загрузили, и использовать указанные нами атрибуты.

Атрибут с меткой `UserName` соответствует пользователю. Атрибуты `QueryTerm_1` и `QueryTerm_2` соответствуют термам запроса, первому и второму соответственно. Эти термы получены с помощью класса `StandardAnalyzer` из библиотеки Lucene. Во время тренировки мы назначаем вероятности в зависимости от частоты появления каждого образца. Важным методом в нашем контексте является метод `getProbability(Concept c, Instance i)`, который мы будем использовать для получения оценки релевантности отдельного URL-адреса (`Concept`), когда данный пользователь выполняет конкретный запрос (`Instance`).

2.4.3. Объединение оценок индексирования Lucene, вектора PageRank и данных о переходах пользователя по ссылкам

Вооружившись вероятностью того, что пользователь для данного запроса отдаст предпочтение конкретному URL-адресу, мы можем пойти дальше и объединить все три методики, чтобы получить дополнительные результаты поиска. Соответствующий программный код приведен в листинге 2.13.

Листинг 2.13. Индексирование Lucene, вектор PageRank и вероятность перехода пользователя по ссылкам

```
public SearchResult[] search(UserQuery uQuery,
➥  int numberOfMatches, Rank pR) {

  SearchResult[] docResults =
➥  search(uQuery.getQuery(), numberOfMatches);      ← Результаты, полученные
                                                       с использованием индекса
  String url;

  StringBuilder strB = new StringBuilder();

  int docN = docResults.length;

  if (docN > 0) {                                    Сбор документов в количестве,
                                                     не превышающем numberOfMatches
    int loop = (docN<numberOfMatches) ? docN : numberOfMatches;

     for (int i = 0; i < loop; i++) {

       url = docResults[i].getUrl();
                                                     Сбор оценок для всех
       UserClick uClick = new UserClick(uQuery,url);  пользовательских действий

         double indexScore = docResults[i].getScore();
```

```
        double pageRankScore  = pR.getPageRank(url);

        BaseConcept bC = new BaseConcept(url);

        double userClickScore = learner.getProbability(bC, uClick);

        double hScore;

        if (userClickScore == 0) {        ← Вычисление итоговой (гибридной) оценки

          hScore = indexScore * pageRankScore * EPSILON;

        } else {

              hScore = indexScore * pageRankScore * userClickScore;

          }

          docResults[i].setScore(hScore);

      strB.append("Document URL   : ")
➥     .append(docResults[i].getUrl()).append("  -->  ");
      strB.append("Relevance Score: ")
➥     .append(docResults[i].getScore()).append("\n");
    }
  }
  strB.append(PRETTY_LINE);
  System.out.println(strB.toString());

  return docResults;
}
```

На рис. 2.8 показаны результаты, полученные для пользователя dmitry. Как можно видеть, благодаря тому факту, что в прошлом пользователь dmitry несколько раз переходил по ссылке на страницу *biz-03.html*, оценка релевантности для этой страницы является наивысшей. Второе наилучшее попадание – это страница *biz-01.html*, которая также находится в файле с записью пользовательских переходов по ссылкам. Страница со спамом оказывается на третьем месте, но это – побочный эффект небольшого количества страниц: мы намеренно не стали учитывать масштабирующий фактор `m`, чтобы продемонстрировать, как он влияет на результаты.

На рис. 2.9 мы выполняем тот же самый запрос – «google ads», – но на этот раз от имени пользователя babis. Мы изменили на обратный порядок переходов по ссылкам пользователя dmitry, чтобы создать данные о переходах для пользователя babis. Результаты свидетельствуют, что первое попадание – это страница *biz-01.html*; страница *biz-03.html* на втором месте. Все остальное осталось без изменений. Единственное отличие в результирующем наборе возникает благодаря тому факту, что запрос выполнялся разными пользователями, и это различие в точности отражает то, чему *обучилось* (learned) приложение на данных из файла *user-clicks.csv*.

```
bsh % UserQuery dQ = new UserQuery("dmitry", "google ads");
bsh % oracle.search(dQ,5,pr);

Search results using Lucene index scores:
Query: google ads

Document Title: Google Ads and the best drugs
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html  -->
Relevance Score: 0.788674294948578
-------------------------------------------------------
Document Title: Google Expands into Newspaper Ads
Document URL: file:/c:/iWeb2/data/ch02/biz-01.html  -->
Relevance Score: 0.382
-------------------------------------------------------
Document Title: Google sells newspaper ads
Document URL: file:/c:/iWeb2/data/ch02/biz-03.html  -->
Relevance Score: 0.317
-------------------------------------------------------
Document Title: Google's sales pitch to newspapers
Document URL: file:/c:/iWeb2/data/ch02/biz-02.html  -->
Relevance Score: 0.291
-------------------------------------------------------
Document Title: Economic stimulus plan helps stock prices
Document URL: file:/c:/iWeb2/data/ch02/biz-07.html  -->
Relevance Score: 0.031
-------------------------------------------------------
Search results using combined Lucene scores, page rank scores and user
clicks:
Query: user=dmitry, query text=google ads

Document URL: file:/c:/iWeb2/data/ch02/biz-03.html  -->
Relevance Score: 0.0057

Document URL: file:/c:/iWeb2/data/ch02/biz-01.html  -->
Relevance Score: 0.0044

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html  -->
Relevance Score: 0.0040

Document URL: file:/c:/iWeb2/data/ch02/biz-02.html  -->
Relevance Score: 0.0012

Document URL: file:/c:/iWeb2/data/ch02/biz-07.html  -->
Relevance Score: 0.0002
-------------------------------------------------
```

Рис. 2.8. *Объединяем оценки библиотеки Lucene и алгоритма PageRank, а также данные о действиях пользователя, чтобы обеспечить высокую степень релевантности результатов поиска для пользователя dmitry*

```
bsh % UserQuery bQ = new UserQuery("babis", "google ads");
bsh % oracle.search(bQ,5,pr);

Search results using Lucene index scores:
Query: google ads

Document Title: Google Ads and the best drugs
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html  -->
Relevance Score: 0.788674294948578
_____________________________________________________________
Document Title: Google Expands into Newspaper Ads
Document URL: file:/c:/iWeb2/data/ch02/biz-01.html  -->
Relevance Score: 0.382
_____________________________________________________________
Documen t Title: Google sells newspaper ads
Document URL: file:/c:/iWeb2/data/ch02/biz-03.html  -->
Relevance Score: 0.317
_____________________________________________________________
Document Title: Google's sales pitch to newspapers
Document URL: file:/c:/iWeb2/data/ch02/biz-02.html  -->
Relevance Score: 0.291
_____________________________________________________________
Document Title: Economic stimulus plan helps stock prices
Document URL: file:/c:/iWeb2/data/ch02/biz-07.html  -->
Relevance Score: 0.0314
_____________________________________________________________
Search results using combined Lucene scores, page rank scores and user
clicks:
Query: user=babis, query text=google ads

Document URL: file:/c:/iWeb2/data/ch02/biz-01.html  -->
Relevance Score: 0.00616

Document URL: file:/c:/iWeb2/data/ch02/biz-03.html  -->
Relevance Score: 0.00407

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html  -->
Relevance Score: 0.00393

Document URL: file:/c:/iWeb2/data/ch02/biz-02.html  -->
Relevance Score: 0.00117
```

Рис. 2.9. *Примененные совместно оценки библиотеки Lucene и алгоритма PageRank, а также данные о действиях пользователя обеспечивают получение результатов поиска с высокой степенью релевантности для пользователя Babis*

Великолепно! Теперь у нас есть вариант поиска значительно лучше поиска, основанного исключительно на индексировании. Улучшение учитывает структуру связанных гиперссылками документов, а также принимает во внимание предпочтения пользователей, выявленные на основе совершенных ими переходов по ссылкам. Но есть масса приложений, которым приходится осуществлять поиск среди документов, не связанных явно друг с другом. Можем ли мы сделать что-либо, чтобы уточнить результаты поиска в этом случае? Дальше мы проанализируем именно этот случай.

2.5. Ранжирование документов Word, PDF и других документов без ссылок

Предположим, есть сотни тысяч документов, созданных в текстовом процессоре Word или представленных в формате PDF, или документов любого другого типа, в которых вы хотите выполнить поиск. На первый взгляд, может показаться, что единственный выбор здесь – индексирование, и в лучшем случае вам, может быть, удастся также осуществить некоторый анализ экранных данных. Но мы покажем, что можно развить идеи оценки ссылок, которые уже применялись нами для поиска в Интернете. Будем надеяться, что убедим вас придумать и разработать еще более удачный метод. Кстати, насколько нам известно, описываемая здесь методика никогда прежде не была опубликована.

Можно ранжировать и документы без ссылок – чтобы показать это, мы возьмем документы в формате HTML и создадим документы текстового процессора Word с идентичным контентом. Это позволит сравнить результаты с полученными в разделе 2.3 и выявить все сходства и различия двух подходов. Синтаксический разбор документов Word легко осуществить с помощью библиотеки *TextMining* с открытым исходным кодом; обратите внимание: ее имя изменилось на *tm-extractor*. Лицензионное соглашение для этой библиотеки начиная с версии 1.0 – **LGPL**[1], что позволяет использовать ее в коммерческих приложениях. Исходный код можно получить по адресу *http://code.google.com/p/text-mining/source/checkout*. Воспользовавшись этой библиотекой, мы написали класс `MSWordDocumentParser`, который инкапсулирует синтаксический разбор документов текстового процессора Word.

[1] LGPL – GNU Lesser General Public License (Стандартная общественная лицензия ограниченного применения GNU), бывш. GNU Library General Public License (Стандартная общественная лицензия GNU для библиотек). – *Прим. перев.*

2.5.1. Введение в алгоритм DocRank

В листинге 2.14 мы используем для чтения документов Word те же самые классы, что и для чтения документов HTML (класс FetchAndProcessCrawler), а индексирование контента этих документов выполняется с помощью библиотеки Lucene.

Листинг 2.14. Ранжирование документов на основе контента

```
FetchAndProcessCrawler crawler =
➥  new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);

crawler.setUrls("biz-docs");          ← Загрузка бизнес-документов
                                        тестового процессора Word
crawler.addDocSpam();
crawler.run();

LuceneIndexer luceneIndexer =
➥  new LuceneIndexer(crawler.getRootDir());   ← Создание индекса Lucene

luceneIndexer.run();                                Создание простой
                                                    поисковой системы
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());
oracle.search("nvidia",5);

DocRank docRank = new DocRank(luceneIndexer.getLuceneDir(),7);

docRank.setAlpha(0.9);
docRank.setEpsilon(0.00000001);
docRank.build();                                    ← Создание системы DocRank

oracle.search("nvidia",5, docRank);
```

На рис. 2.10 показано, что в результате поиска слова «nvidia» был возвращен, как имеющий наивысшую оценку ранжирования, нежелательный файл со спамом *spam-biz-02.doc*, что аналогично поиску в документах HTML. Разумеется, если речь идет о документах Word, PDF и других текстовых документах, шансы на то, что среди них окажутся документы со спамом, не так велики, но вы могли бы получить маловажные документы с повторно встречающимися в них термами поиска.

До сих пор, в сравнении с листингом 2.9, все осталось по-прежнему. Новый программный код активизируется классом DocRank. Этот класс отвечает за создание показателя релевантности документов друг относительно друга, эквивалентного оценке релевантности, которая по алгоритму PageRank назначается веб-страницам. В отличие от класса PageRank, класс DocRank получает дополнительный аргумент, чью роль мы поясним позже. Как и в предыдущих разделах, мы хотим получить матрицу для отображения в ней важности страницы Y относительно страницы X. Проблема в том, что в отличие от веб-страниц, наши документы явно не связаны друг с другом. Веб-ссылки использовались

```
bsh % oracle.search("nvidia", 5);

Search results using Lucene index sco res:
Query: nvidia

Document Title: NVIDIA shares plummet into cheap medicine for you!
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.doc  -->
Relevance Score: 0.458221405744553
_____________________________________________________________
Document Title: Nvidia shares up on PortalPlayer buy
Document URL: file:/c:/iWeb2/data/ch02/biz-05.doc  -->
Relevance Score: 0.324011474847794
_____________________________________________________________
Document Title: NVidia Now a Supplier for MP3 Players
Document URL: file:/c:/iWeb2/data/ch02/biz-04.doc  -->
Relevance Score: 0.194406896829605
_____________________________________________________________
Document Title: Nov. 6, 2006, 2:38PM?Chips Snap: Nvidia, Altera Shares
Jump
Document URL: file:/c:/iWeb2/data/ch02/biz-06.doc  -->
Relevance Score: 0.185187965631485

_____________________________________________________________
```

Рис. 2.10. *Поиск на основе индексирования слова «nvidia» в документах Word, содержащих бизнес-новости и спам*

только для того, чтобы создать матрицу, значения которой говорят о том, насколько важна страница Y в сравнении со страницей X. Если бы удалось найти способ, позволяющий назначить показатель степени важности документа Y в сравнении с документом X, удалось бы использовать ту же математическую теорию, которая поддерживает алгоритм PageRank. Наш программный код обеспечивает получение такой матрицы.

2.5.2. Внутренние механизмы алгоритма DocRank

Наше измерение важности носит, по большей части, произвольный характер, и для его жизнеспособности крайне важны два свойства, которые относятся к элементам новой матрицы H. Эта матрица должна иметь элементы, отвечающие следующим условиям:

- Все они являются положительными числами.
- Сумма значений которых в любой строке равна 1.

Будет ли наш показатель иметь успех – зависит от того, какого рода документы мы обрабатываем. В листинге 2.15 показан программный код из класса `DocRankMatrixBuilder`, который формирует матрицу H для документов Word.

Листинг 2.15. Класс DocRankMatrixBuilder: ранжирование текстовых документов на основе контента

```
public class DocRankMatrixBuilder implements CrawlDataProcessor {
    private final int TERMS_TO_KEEP = 3;

    private int termsToKeep=0;
    private String indexDir;
    private PageRankMatrixH matrixH;

    public void run() {
        try {
            IndexReader idxR =
➥           IndexReader.open(FSDirectory.getDirectory(indexDir));
            matrixH = buildMatrixH(idxR);
        }
        catch(Exception e) {
            throw new RuntimeException("Error: ", e);
        }
    }
    // Сбор ключей (id) документов из индекса
    // для документов соответствующего типа
    private List<Integer> getProcessedDocs(IndexReader idxR)
        throws IOException {
        List<Integer> docs = new ArrayList<Integer>();
        for(int i = 0, n = idxR.maxDoc(); i < n; i++) {
            if( idxR.isDeleted(i) == false ) {
                Document doc = idxR.document(i);
                if( eligibleForDocRank(doc.get("doctype") ) ) {
                    docs.add(i);
                }
            }
        }
        return docs;
    }
// Подходит ли данная запись из индекса?
    private boolean eligibleForDocRank(String doctype) {
        return ProcessedDocument.DOCUMENT_TYPE_MSWORD
➥      .equalsIgnoreCase(doctype);
    }
    private PageRankMatrixH buildMatrixH(IndexReader idxR)
        throws IOException {
     // Учет только тех URL-адресов, чей контент был извлечен,
     // и для него был выполнен синтаксический разбор
        List<Integer> allDocs = getProcessedDocs(idxR);
```

```
        PageRankMatrixH docMatrix = new PageRankMatrixH( allDocs.size() );

        for(int i = 0, n = allDocs.size(); i < n; i++) {

             for(int j = 0, k = allDocs.size(); j < k; j++) {
                    double similarity = 0.0d;

                Document docX = idxR.document(i);
                    String xURL= docX.get("url");

                    if ( i == j ) {

                      // Остерегаемся бесстыдной саморекламы ;-)
                      docMatrix.addLink(xURL, xURL, similarity);

                    } else {

                     TermFreqVector x =
➥                   idxR.getTermFreqVector(i, "content");
                     TermFreqVector y =
➥                   idxR.getTermFreqVector(j, "content");

                     similarity = getImportance(x.getTerms(),
➥                   x.getTermFrequencies(), y.getTerms(),
➥                   y.getTermFrequencies());


                     // Добавление ссылки из документа docX на документ docY
                     Document docY = idxR.document(j);
                     String yURL = docY.get("url");
                     docMatrix.addLink(xURL, yURL, similarity);
                }
             }
        }
        docMatrix.calculate();

        return docMatrix;
    }

    // Вычисление  степени важности документа Y в контексте документа X
    private double getImportance(String[] xTerms, int[] xTermFreq,
                                 String[] yTerms, int[] yTermFreq){

     // xTerms - это массив наиболее часто встречающихся термов для первого
документа
        Map<String, Integer> xFreqMap =
➥       buildFreqMap(xTerms, xTermFreq);

       // yTerms - это массив наиболее часто встречающихся термов
       // для второго документа
        Map<String, Integer> yFreqMap =
➥       buildFreqMap(yTerms, yTermFreq);
```

```
        // sharedTerms - это пересечение двух множеств
        Set<String> sharedTerms =
➥       new HashSet<String>(xFreqMap.keySet());
        sharedTerms.retainAll(yFreqMap.keySet());
        double sharedTermsSum = 0.0;

        // Обратите внимание на отсутствие симметрии.
        // Если X и Y поменять местами, будет получено другое
        // значение; разумеется, только если частота неодинакова!

        double xF, yF;
        for(String term : sharedTerms) {
             xF = xFreqMap.get(term).doubleValue();
             yF = yFreqMap.get(term).doubleValue();

             sharedTermsSum += Math.round(Math.tanh(yF/xF));
        }

        return sharedTermsSum;
    }
    private Map<String, Integer> buildFreqMap(String[] terms, int[] freq) {

int topNTermsToKeep = (termsToKeep == 0)? TERMS_TO_KEEP: termsToKeep;

Map<String, Integer> freqMap =
➥ TermFreqMapUtils.getTopNTermFreqMap(terms, freq, topNTermsToKeep);

        return freqMap;
    }
}
```

В нашем решении есть два важнейших компонента. Во-первых, обратите внимание на то, что мы используем *векторы термов* (term vectors) библиотеки Lucene, которые представляют собой пару термов с частотой их появления. Если вспомнить обсуждение индексирования документов с помощью библиотеки Lucene, мы упоминали о том, что текст документа сначала подвергается синтаксическому разбору, а затем анализируется перед тем, как будет проиндексирован. На стадии анализа текст разбивается на *лексемы* (термы); способ разметки текста зависит от применяемого анализатора. Библиотека Lucene хороша тем, что позволяет извлечь и использовать эту информацию позже. Кроме термов из текста, библиотека Lucene также предоставляет данные о том, сколько раз каждый терм встречается в документе. Это все, что требуется от библиотеки Lucene, – набор термов и частота их появления в каждом документе.

Второй компонент нашего решения – процедура определения степени важности, приписываемой каждому документу. Представленный в листинге 2.15 метод `getImportance` демонстрирует, что для каждого документа X мы вычисляем степень важности документа Y за следующие

два шага: 1) находим пересечение между множеством наиболее часто встречающихся термов документа *X* и множеством наиболее часто встречающихся термов документа *Y*; 2) для каждого терма из множества общих термов (пересечение) вычисляем отношение числа появлений терма в документе *Y* (Y-частота появления) к числу появлений терма в документе *X* (X-частота появления). Важность документа *Y* в контексте документа *X* определяется как сумма всех этих относительных величин и фильтруется функцией гиперболического тангенса (`Math.tanh`), а также функцией округления (`Math.round`). Конечным результатом этих операций будет элемент матрицы *H* (строка *X*, столбец *Y*).

Мы используем функцию гиперболического тангенса, потому что хотим оценить с помощью шкалы оценок, стоит ли рассматривать конкретный терм, встречающийся в двух документах, как заслуживающий доверия индикатор для назначения степени важности документа. Нас не интересует точное значение – нам нужно только удержать показатель степени важности в разумных пределах. Гиперболический тангенс получает значения из интервала от 0 до 1, поэтому финальное округление гарантирует, что каждым термом можно или пренебречь, или принять его за единицу степени важности. Таково логическое обоснование построения формулы с применением этих функций.

На рис. 2.11 показано, что в результате поиска слова «nvidia» будет возвращен файл *biz-05.doc* как имеющий наивысшую оценку ранжирования; этот файл является подлинным (это не спам) и имеет отношение к фирме NVidia! Страница со спамом уцелела, потому что у нас мало документов, но мы получили дополнительную полезную возможность. Lucene-индекс с самого начала располагал точно такой же информацией, но на его оценку релевантности повлиял, исказив ее, новостной эрзац-документ. Алгоритм DocRank помог нам повысить степень релевантности документа *biz-05.doc*, и в более реалистичных ситуациях он может помочь вам идентифицировать наиболее подходящие документы в коллекции. Оценочные значения, которые обеспечивает алгоритм DocRank, как и значения, получаемые с помощью алгоритма PageRank, должны быть вычислены только однажды, но их можно использовать повторно для всех запросов.

Есть и другие средства, позволяющие улучшить результаты поиска обычных документов, и в конце этой главы мы привели соответствующие ссылки. DocRank – это алгоритм, который обладает наибольшими возможностями, если применять его к данным, хранящимся в реляционной базе данных. Чтобы убедиться в этом, предположим, что у нас есть две таблицы – таблица *A* и таблица *B*, связанные друг с другом посредством таблицы *C* (это распространенный случай). Например, вы могли бы иметь одну таблицу, где хранятся данные о пользователях, другую – с данными о группах, и еще одну таблицу, в которой хранятся данные о взаимосвязях (relationship) между пользователями и группа-

```
bsh % oracle.search("nvidia",5,dr);

Search results using Lucene index scores:
Query: nvidia

Document Title: NVIDIA shares plummet into cheap medicine for you!

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.doc  -->
Relevance Score: 0.4582
____________________________________________________________
Document Title: Nvidia shares up on PortalPlayer buy

Document URL: file:/c:/iWeb2/data/ch02/biz-05.doc  -->
Relevance Score: 0.3240
____________________________________________________________
Document Title: NVidia Now a Supplier for MP3 Players

Document URL: file:/c:/iWeb2/data/ch02/biz-04.doc  -->
Relevance Score: 0.1944
____________________________________________________________
Document Title: Chips Snap: Nvidia, Altera Shares Jump

Document URL: file:/c:/iWeb2/data/ch02/biz-06.doc   -->
Relevance Score: 0.1852
____________________________________________________________
Search results using combined Lucene scores and page rank scores:
Query: nvidia

Document URL: file:/c:/iWeb2/data/ch02/biz-05.doc  -->
Relevance Score: 0.03858

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.doc  -->
Relevance Score: 0.03515

Document URL: file:/c:/iWeb2/data/ch02/biz-04.doc  -->
Relevance Score: 0.02925

Document URL: file:/c:/iWeb2/data/ch02/biz-06.doc  -->
Relevance Score: 0.02233
____________________________________________________________
```

Рис. 2.11. *Поиск слова «nvidia» в документах Word с использованием индекса и ранжирования*

ми в виде взаимосвязи между идентификаторами ID каждой записи. По сути, у вас есть граф, который связывает пользователей на основании их принадлежности к группе, и другой граф, который связывает

группы на основании входящих в них пользователей. Каждый раз, когда у вас есть связное представление предметных сущностей, стоит попытаться применить алгоритм DocRank или аналогичный вариант. Не бойтесь экспериментировать! Задачи такого типа не имеют единственно верного решения, и иногда ответ может оказаться для вас сюрпризом.

2.6. Проблемы масштабной реализации

Все, о чем мы говорили до сих пор, может быть использовано в функциональных областях и в различных предметных областях, для которых создаются веб-приложения. Но если вы планируете обрабатывать очень большие объемы данных, обладая соответствующими вычислительными ресурсами, перед вами встают вопросы, главным образом из двух категорий. Вопросы из первой категории связаны с математическими свойствами алгоритмов, из второй – со сторонами разработки ПО, относящимися к манипулированию данными, размер которых измеряется тера- или даже петабайтами!

Первый признак ограничений масштабных вычислений – нехватка адресуемой оперативной памяти. Другими словами, объем данных настолько велик, что структуры данных больше не умещаются в оперативной памяти. Этот симптом должен быть особенно верным для интерпретируемых языков программирования, таких как Java, потому что, даже если вы ухитритесь втиснуть данные в память, придется, вероятно, побеспокоиться о сборе мусора. Для случаев масштабных вычислений есть две основные стратегии, позволяющие решить эту проблему. Первая стратегия – разработка более эффективных структур данных, позволяющих уместить данные в оперативной памяти; вторая стратегия – это разработка эффективной, распределенной инфраструктуры ввода/вывода для доступа к данным и манипулирования ими *in situ*[1]. Для очень больших наборов данных, размер которых сопоставим с объемами данных, обрабатываемых поисковой системой Google, придется реализовать обе стратегии, поскольку понадобится «выжать» из системы каждый бит в пользу эффективности.

Что касается более эффективного представления данных, рассмотрим структуры, которые мы использовали для хранения матрицы H. Для той части матрицы, которая отражает исходную структуру ссылок, требуется массив `double[n][n]`, и еще один массив `double[n][n]`, где `n` – число страниц (или документов в случае алгоритма DocRank), требуется для той части матрицы, которая содержит данные висячих узлов. Если вдуматься, налицо колоссальная трата ресурсов, когда `n` очень велико, потому что большая часть этих значений типа `double` являются нулями.

[1] По месту (лат.). – *Прим. перев.*

Рациональнее было бы использовать для хранения этой информации *список смежных узлов* (adjacency list*)*. В языке Java вы легко можете реализовать такой список, воспользовавшись коллекцией `Hashtable`, которая будет содержать объекты `HashSet`. Итак, определение переменной `matrix` в классе `PageRankMatrixH` выглядело бы следующим образом:

```
Hashtable<Integer, Hashtable <Integer,Double>> matrix;
```

Одно из предлагаемых упражнений – переписать нашу реализацию алгоритма, используя указанные рациональные структуры. Вы могли бы даже сжать данные в списке смежных узлов, применив методику *ссылочного кодирования* (reference encoding) или другие методики [см. Boldi, P., and S. Vigna]. Ссылочное кодирование основано на сходстве веб-страниц и жертвует простотой реализации ради эффективного использования оперативной памяти.

Еще один аспект реализации масштабного поиска – это точность оценок PageRank (или любой другой реализации базового класса `Rank`), которой вы собираетесь добиться. Чтобы обеспечить отличающиеся оценочные значения PageRank для любых двух веб-страниц из N, в численных расчетах нужно будет обеспечить как минимум точность порядка $1/N$. Таким образом, если вы имеете дело со страницами в количестве $N = 1000$, то удовлетворительной должна оказаться даже точность порядка 10^{-4}. Если вы хотите получить показатели ранжирования для миллиардов страниц, точность оценочных значений PageRank должна быть порядка 10^{-10}.

Рассмотрим ситуацию, в которой большая часть извлеченных веб-страниц представлена висячими узлами. Такая ситуация возможна, если вы хотите создать специализированную поисковую систему для главного сайта, такого как набор проектов Apache, или чего-то менее амбициозного, вроде единственного проекта Jakarta. Брин и Пейдж поняли, что обработка большого числа узлов, которые являются, по существу, искусственными – поскольку их элементы в матрице H не отражают структуру ссылок Интернета, скорее помогая матрице обзавестись некими подходящими математическими свойствами, – не обещает быть очень эффективной. Они предположили, что при вычислении вектора PageRank можно было бы удалить висячие узлы и вернуть их обратно после того, как значения для остальных векторов PageRank обретут достаточную сходимость.

Мы не знаем, конечно, как на самом деле реализована поисковая система Google – такие секреты тщательно охраняются, – но можно с уверенностью сказать, что объективная обработка всех страниц потребует включения висячих узлов с начала до конца вычислений PageRank. Стремясь одновременно к объективности и эффективности, мы можем применить методы, которые опираются на симметричное переупорядочивание матрицы H. Оказывается, эти методики обеспечивают ту же

быстроту сходимости, что и оригинальный алгоритм PageRank, примененный к менее масштабной задаче, что означает возможность существенного выигрыша во времени вычислений; подробности ищите в книге «Google's PageRank and Beyond: The Science of Search Engine Rankings» [Langville, A. N. and C. D. Meyer].

Подспудно беспокойство об оперативной памяти и быстродействии присутствует во всех дискуссиях по поводу масштабных вычислений, обеспечивающих поиск. Один из факторов, влияющих на скорость, – число итераций степенного метода, которое, как мы видели, зависит от значения `alpha`, а также от числа связанных страниц. К сожалению, в написанных практиками книгах, подобных нашей, мы обнаружили утверждения, что неважно, какие значения используются для инициализации вектора PageRank: можно было бы сделать все значения равными 1. На самом деле, это не так, и может привести к драматическим последствиям, если вы работаете с большими наборами данных, состав которых периодически меняется. Чем ближе исходный вектор к уникальным значениям PageRank, тем меньше требуется итераций. Ряд методик под общим названием *приближенные методики агрегирования* (approximate aggregation techniques), обеспечивающих вычисление вектора PageRank из матрицы меньшего размера, для того, чтобы сгенерировать оценку истинности, обновляют распределение вектора PageRank. Эта оценка, в свою очередь, будет использоваться в качестве исходного вектора для окончательных вычислений. В этой книге не будут рассматриваться математические обоснования этих методов. В разделе 2.10 приведены ссылки на дополнительную информацию об этих методиках.

Говоря о методиках, позволяющих ускорить вычисление вектора PageRank, мы должны упомянуть экстраполяцию Айткена (Aitken extrapolation), методику квадратичной экстраполяции Камвара (Kamvar) и др., а также более развитые методики, такие как применение методов спектрального анализа (например, методы спектрального анализа с использованием полинома Чебышева). Эти методики нацелены на получение лучшей аппроксимации вектора PageRank между итерациями. Вы можете применять их для вычисления ранжирования, и их реализация может оказаться уместной (см. раздел 2.10).

Что касается ПО для реализации в условиях масштабных вычислений, мы должны упомянуть проект Hadoop (*http://hadoop.apache.org/*). Hadoop – это полнофункциональный проект самого высокого уровня фонда Apache Software Foundation, предлагающий базовые программные средства с открытым исходным кодом, которые являются масштабируемыми, экономичными, эффективными и надежными. Проект Hadoop реализует программный фреймворк MapReduce [см. Dean, J. and S. Ghemawat] посредством собственной распределенной файловой системы (HDFS). MapReduce делит приложения на несколько небольших ра-

бочих блоков. Система HDFS для надежности создает несколько копий блоков данных и размещает их в вычислительных узлах, образующих вычислительный кластер (рис. 2.12). После этого фреймворк MapReduce может обрабатывать данные по месту их нахождения. Проект Hadoop демонстрировался на кластерах, насчитывающих до 2000 узлов. Текущая проектная задача для платформы Hadoop – работа с кластерами из 10 000 узлов.

Способность обрабатывать большие наборы данных, безусловно, очень важна в реальных производственных системах. Мы бегло ознакомили вас с вопросами, которые могут возникнуть, и указали некоторые подходящие проекты и релевантную литературу по этой теме. Проектируя поисковую систему, необходимо учитывать не только вашу способность к масштабированию и обработке более крупных объемов данных, но и качество получаемых результатов поиска. К концу дня пользователи жаждут, чтобы результаты можно было получить быстро и они оказались бы точными. Итак, давайте рассмотрим несколько способов, позволяющих количественно измерить, в какой степени то, что у нас есть, является тем, что нам надо.

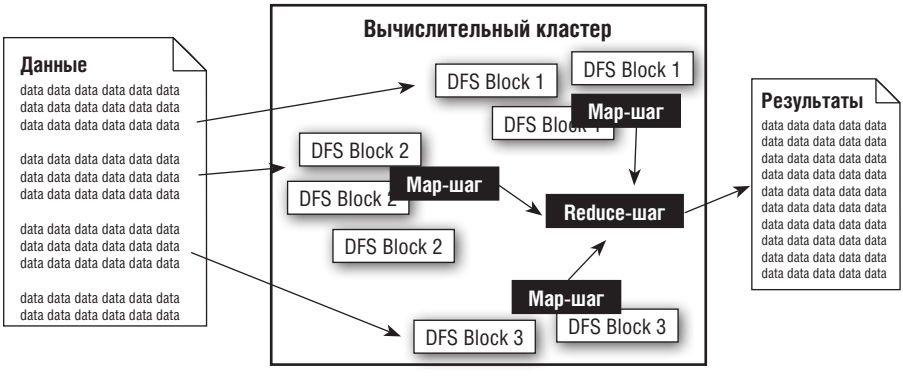

Рис. 2.12. *Реализация фреймворка MapReduce из проекта Hadoop с применением распределенной файловой системы*

2.7. Получили ли вы то, что искали? Точность и выборка

Фирмы Google и Yahoo! потратили немало времени на изучение качества своих поисковых систем. Подобно процессу проверки (verification) и аттестации (validation) в случае контроля качества (quality assurance, QA) систем ПО, качество поиска крайне важно для успеха поисковой системы. Введя запрос в поисковой системе, вы можете найти или не найти то, что вам нужно. Есть разные показатели, позволяющие изме-

рить степень успешности поисковой системы. Два наиболее распространенных показателя – *точность* (precision) и *выборка* (recall) – легко реализуются и качественно понятны.

На рис. 2.13 показаны возможные результаты выполнения типичного запроса. То есть для множества документов одно результирующее подмножество будет состоять из документов, релевантных запросу, а другим подмножеством, полученным в результате поиска, будут извлеченные документы. Понятно, что целью является извлечение всех релевантных документов, но это редкий случай. Таким образом, наше внимание быстро переключается на пересечение этих двух подмножеств, как следует из рис. 2.13.

В информационном поиске, точность – это отношение числа извлеченных релевантных документов (RR) к общему числу извлеченных документов (Rd): *точность* = *RR/Rd*. На рис. 2.13 точность была бы равна примерно 1/5, или 0,2. Это оценка «на глазок», она не является точной, мы ведь инженеры! С другой стороны, выборка – это отношение числа извлеченных релевантных документов к общему числу релевантных документов (Rt): *выборка* = *RR/Rt*.

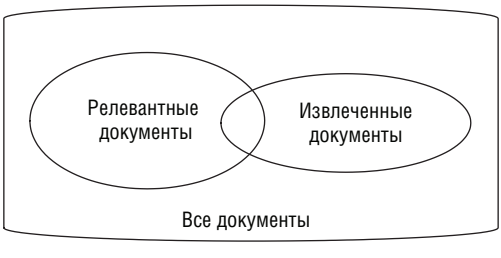

Рис. 2.13. *На этой схеме показан набор релевантных документов и набор извлеченных документов; пересечение этих двух наборов используется для определения показателей поиска – точности и выборки*

Качественно эти два показателя отвечают на разные вопросы. Точность – это ответ на вопрос: «В какой мере то, что я получаю, является тем, что мне надо?». Выборка позволяет ответить на вопрос: «Включает ли то, что я получил, все, что я могу получить?» Понятно, что легче вычислить точность, чем выборку, потому что вычисление выборки предполагает, что мы уже знаем набор всех релевантных документов для данного запроса. На самом деле, так почти никогда не бывает. Мы представляем эти два показателя вместе, на одной схеме, с тем чтобы можно было оценить, в какой степени хорошие результаты смешаны с плохими. Если то, что я получаю, – правда, только правда и ничего кроме правды, значит, оба значения, характеризующие точность и выборку моих запросов, будут близки к единице.

При оценке алгоритмов и во время манипуляций по тонкой настройке поисковой системы пригодятся графики этих двух величин для репре-

зентативных запросов, охватывающих вопросы, ответы на которые пытаются получить ваши пользователи. На рис. 2.14 показан типичный график этих величин. Для каждого запроса мы наносим на график точку, соответствующую значениям точности и выборки для этого запроса. Если выполнить несколько запросов и нанести эти точки на график, вы получите линию, вроде той, что показана на рис. 2.14. Ясно, что при небольшом количестве запросов будет лишним интерполировать эти значения. Лучше оставить их в виде отдельных точек.

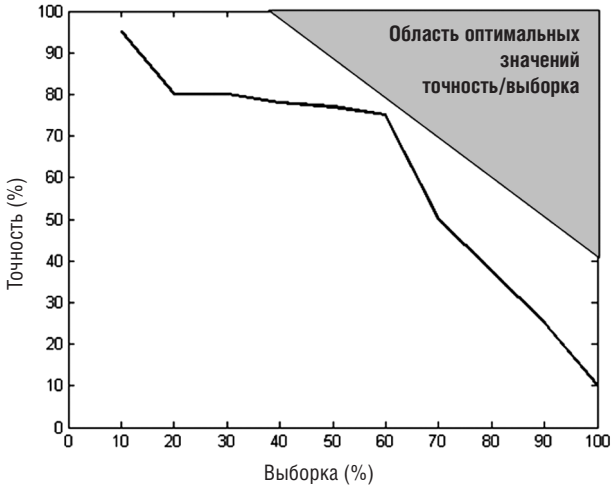

Рис. 2.14. *Типичный график точность/выборка для поисковой системы*

Хорошие точки точность/выборка располагаются в верхнем правом углу графа, потому что мы хотим иметь высокую точность и широкую выборку. Эти графики могут помочь объективно установить необходимость выполнения конкретной настройки в алгоритме или удостовериться в преимуществе одного подхода в сравнении с другим. Что помогло бы вам убедить ваше всегда скептически настроенное руководство применить алгоритмы, рассматриваемые в этой книге! Вы можете попрактиковаться, используя три подхода, которые мы представили в этой главе (поиск с помощью библиотеки Lucene; поиск с применением библиотеки Lucene и алгоритма PageRank; библиотека Lucene, алгоритм PageRank и анализ экранных данных). Можно применить эти подходы к набору данных, предоставленному нами, или опробовать их на другом наборе данных, созданном вами, и построить график точность/выборка, включающий результаты 10–20 запросов.

В разделе 5.5 мы рассмотрим несколько аспектов, связанных с достоверностью, которую можно оценить для конкретного алгоритма, а также обсудим, как сравнить два алгоритма. Мы также поговорим о том, каким образом следует проводить экспериментальную проверку пра-

вильности данных (validation experiments) с целью повышения степени доверия к полученным результатам. Точность и выборка – это вершина айсберга, когда речь идет о качестве результатов поиска. Мы отложим более подробный анализ правдоподобия до того момента, как будут рассмотрены все основные интеллектуальные алгоритмы, которые мы хотим представить. Такой подход позволит нам использовать общий фреймворк для оценки качества интеллекта.

2.8. Заключение

С начала 2000-х очень многие новостные статьи в Интернете провозгласили: «Поиск – вот король!» В прошлом тысячелетии подобное утверждение можно было бы назвать прозорливым или даже пророческим, но сегодня это всемирно признанная истина. Не верите нам – спросите у поисковой системы Google!

Эта глава показывает, что предоставление интеллектуальных ответов на пользовательские запросы к насыщенному контентом материалу, разбросанному по всему земному шару, заслуживает внимания и усилий, не ограничивающихся индексированием. Мы продемонстрировали стратегию поиска, выстраивание которой начинается с применения традиционных методик извлечения информации, обеспечиваемых библиотекой Lucene. Рассказали о сборе контента в Интернете (краулинг) и предоставили свою реализацию поискового робота. Использовали ряд парсеров документов, таких как NekoHTML и библиотека TextMining (tm-extractor), передав контент анализаторам Lucene. Стандартные анализаторы из библиотеки Lucene, обладая широкими возможностями и универсальностью, должны удовлетворить потребностям большинства задач. Если они вам не подходят, мы рассмотрели ряд возможностей, позволяющих расширить и модифицировать функциональность этих анализаторов. Мы также коснулись широких возможностей фреймворка запросов Lucene, его собственной способности к расширению имеющихся функций и гибкости.

Что еще более важно, мы подробнейшим образом описали самый знаменитый алгоритм оценки ссылок – PageRank. Мы предоставили полную реализацию, которая лишена каких-либо зависимостей и адаптирует представление *G*(oogle)-матрицы, позволяющей осуществить масштабную реализацию разреженных матриц. Кроме того, мы предоставили рекомендации, которые позволят вам завершить этот этап с чувством гордости за это самостоятельное великое свершение! Мы затронули несколько сложных моментов этого алгоритма и детально истолковали его ключевые характеристики, такие как компонент телепортации и степенной метод.

Также был представлен анализ экранных данных, который познакомил вас с интеллектуальными вероятностными методиками, такими как реализованный нами наивный байесовский классификатор. Мы

не только предоставили классы-оболочки, которые раскрывают все важнейшие этапы реализации, но и очень подробно проанализировали внутреннюю работу программного кода. Подобная методика позволяет изучить предпочтения пользователя в отношении конкретного сайта или темы, и ее можно в значительной степени развить и дополнить новыми функциональными возможностями.

Поскольку одним размером на всех не обойтись, мы предоставили материал, который поможет вам в работе с документами, не являющимися веб-страницами, если вы задействуете новый алгоритм, который мы назвали DocRank. Это довольно перспективный алгоритм, но важнее то, что он показал: математическую теорию, на которой основан алгоритм PageRank, можно легко дополнять и применять в иных контекстах путем аккуратной модификации. Наконец, мы поговорили о некоторых проблемах, которые могут возникнуть при работе в очень крупных сетях, и предоставили простой, но надежный способ, позволяющий оценить полученные результаты поиска и повысить достоверность поисковой системы.

Утверждение «поиск – вот король» можно было бы признать истиной, но системы выработки рекомендаций – тоже королевских кровей! В следующей главе рассматриваются исключительно вопросы выработки предложений и рекомендаций. Если добавить в приложение и то и другое, это может разительно повлиять на восприятие вашего приложения пользователями. Но прежде чем двинуться дальше, прочтите в разделе «Сделать» пункты, относящиеся к поиску, если вы еще этого не сделали. Там много интересной и ценной информации.

2.9. Сделать

Последний раздел каждой главы в оставшейся части этой книги будет содержать несколько заданий, которые послужат вам руководством при изучении различных тем. Везде, где это уместно, наш программный код снабжен тегами «ToDo», которые должны быть видны на панели задач Tasks интегрированной среды разработки (IDE) Eclipse. Если щелкнуть мышью на любой из задач в этом списке, соответствующая ссылка позволит увидеть фрагмент программного кода, которая относится к этой задаче. Если вы не пользуетесь средой Eclipse, просто ищите в программном коде метку «ToDo».

Некоторые из этих заданий нацелены на углубление в тему, рассмотренную в основной части главы, другие позволяют расширить обсуждаемую тему. Выполнение этих заданий обеспечит более глубокое и широкое представление об интеллектуальных алгоритмах. Мы настоятельно рекомендуем вам внимательно с ними ознакомиться.

Исходя из сказанного, вот наш список заданий, которые надо выполнить для главы 2.

1. *Создайте собственную систему поиска в Интернете.* С помощью выбранного вами поискового робота выполните краулинг своего любимого сайта, например *http://jakarta.apache.org/*, после чего воспользуйтесь нашим поисковым роботом, чтобы обработать извлеченные данные, построить для них индекс и выполнить поиск по его страницам.

   Как будут отличаться полученные результаты, если добавить к ним оценки алгоритма PageRank?

   Как насчет анализа экранных данных?

   Вы могли бы написать свою небольшую систему поиска данных в Интернете, применив материал этой главы. Попробуйте и расскажите об этом нам!

2. *Эксперименты с повышением степени важности.* В программном коде уберите символы комментария из строк 83–85 в классе `LuceneIndexBuilder` и посмотрите, как изменятся результаты ранжирования, которые обеспечивает библиотека Lucene. В зависимости от приложения, вы можете придумать уникальную стратегию повышения степени важности ваших документов, зависящую от факторов, специфичных для предметной области вашего приложения.

3. *Масштабирование значений вектора PageRank.* В примере с объединением поиска Lucene (индекс) и поиска PageRank (ранжирование) мы использовали масштабный коэффициент, который повышал оценки PageRank. Выбранная нами функция для экспоненты получала только один параметр: `m = (1 - 1/n)`, где `n` – размер матрицы H, и эта функция вела себя так, что для крупных сетей значение масштабного коэффициента приближалось к 1, тогда как для небольших сетей это было значение из интервала от 0 до 1. В реальных условиях вы получаете нулевое значение только в вырожденном случае, когда имеется лишь одна страница, но такая сеть в любом случае не вызывает большого интереса!

   Поэкспериментируйте с такими масштабными коэффициентами, наблюдая их влияние на ранжирование. Возможно, вам захочется изменить это значение, указав более высокую степень `n` – другой допустимой формулой была бы формула `m = (1 - 1 / Math.pow(n,k))`, поскольку при значениях $k > 1$ значение PageRank быстрее достигает своего вычисленного значения.

4. *Преобразование матрицы G: висячие узлы.* Для всех узлов мы присвоили значение $1/n$ каждому элементу в строке с висячими узлами. В отсутствие дополнительной информации об особенностях пользовательского просмотра веб-страниц (browsing habits) или в предположении, что имеется достаточное количество пользователей, применяющих все эти характерные особенности, такое назначение логически обосновано. Но что если исходить из других предположений, также логически обоснованных, – будет ли вся система работать и в этом случае?

Предположим, что пользователю встретился висячий узел. При появлении висячего узла, казалось бы, естественно предположить, что пользователь, скорее всего, выберет в качестве следующего пункта назначения поисковую систему или веб-сайт, похожий на сайт висячего узла, а не с совершенно другим контентом, чем у сайта висячего узла. Результатом подобного допущения была бы корректировка значений висячих узлов: более высокие оценки для поисковых систем и страниц с аналогичным контентом, более низкие – для всего остального. Как такое изменение повлияет на оценки PageRank? Как насчет результатов запросов? Изменится ли в этом случае график точность/выборка?

5. *Преобразование матрицы G: телепортация*. В нашей исходной реализации вклад телепортации был определен в уравнительной манере – всем страницам назначен вклад, равный *(1 – alpha)/n*, где *n* – число страниц. Но потенциал этого компонента огромен. Если выбрать его долю надлежащим образом, он может создать онлайн-капиталистов, и если эта доля выбирается на уровне пользователей, этот компонент может повлиять на предпочтения каждого пользователя во многом подобно тому, как это позволяет нам сделать методика перехода пользователя по ссылкам. Последнее является причиной того, почему долю телепортации называют также *вектором персонализации*.

   Попробуйте модифицировать вектор персонализации так, чтобы определенные страницы получили большие веса, чем остальные. Работает? Возросли ли оценки PageRank для этих страниц? Какие проблемы вы видите в такой реализации? Если мы предположим, что назначили вектор персонализации каждому пользователю, как это скажется на затратах на вычисления? Оправдывают ли себя такие действия? Осуществимо ли это? Этот вопрос рассматривается в работах [Haveliwala, T. H. и Jeh, G. and J. Widom, а также Richardson, M. and P. Domingos], там же можно найти дополнительную информацию и вникнуть в суть этой важной темы.

6. *Объединение различных оценок*. В разделе 2.4.3 мы показали способ, позволяющий объединить три различные оценки, чтобы обеспечить окончательное ранжирование результатов конкретного запроса. Это не единственный способ. Здесь тот случай, когда вы можете придумать, как сбалансировать эти три понятия, чтобы это в наибольшей степени отвечало вашим потребностям. Вот идея: введите понятия взвешенности для каждой из трех оценок и поэкспериментируйте с различными вариантами распределения весов для каждой из них.

   При условии, что вы рассматриваете фиксированную сеть страниц или документов, как меняются результаты в зависимости от различных значений этих весовых коэффициентов? Нанесите на график 20 значений точность/выборка, соответствующих 20 различным запросам, и проделайте это для трех различных комбинаций весов, на-

пример (0,6; 0,2; 0,2), (0,2; 0,6; 0,2), (0,2; 0,2; 0,6). Что вы наблюдаете? Как эти точки отличаются от распределения с равными весами (1; 1; 1)? Можете ли вы предложить другие формулы для уравновешивания долей различных компонентов?

2.10. Ссылки

- Adamic, L. A., R. M. Lukose, A. R. Puniyani, and B. A. Huberman. «Search in power-law networks». *Physical Review E*, vol. 64, 046135. 2001.
- Boldi, P., and S. Vigna. «The WebGraph Framework I: Compression Techniques». WWW 2004, New York.
- Dean, J. and S. Ghemawat. «MapReduce: Simplified Data Processing on Large Clusters». Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, 2004. *http://labs.google.com/papers/mapreduce-osdi04.pdf*.
- Haveliwala, T. H. «Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search». IEEE transactions on Knowledge and Data Engineering, 15 (4): 784. 2004. *http://www-cs-students.stanford.edu/~taherh/papers/topic-sensitive-pagerank-tkde.pdf*.
- Jeh, G. and J. Widom. «Scaling personalized web search». Technical report, Stanford University, 2002. *http://infolab.stanford.edu/~glenj/spws.pdf*.
- Kamvar, S. D., T. H. Haveliwala, Christopher D. Manning, and Gene H. Golub. «Extrapolation Methods for Accelerating PageRank Computations». WWW 2003. *http://www.kamvar.org/code/paper-server.php?filename=extrapolation.pdf*.
- Langville, A. N. and C. D. Meyer. «Google's PageRank and Beyond: The Science of Search Engine Rankings». Princeton University Press, 2006.
- Richardson, M. and P. Domingos. «The intelligent surfer: Probabilistic combination of link and content information in PageRank». Advances in Neural Information Processing Systems, 14:1441, 2002. *http://research.microsoft.com/users/mattri/papers/nips2002/qd-pagerank.pdf*.
- Rish, I. «An empirical study of the naпve Bayes classifier». IBM Research Report, RC22230 (W0111-014), 2001. *http://www.cc.gatech.edu/~isbell/classes/reading/papers/Rish.pdf*.

3

Выработка предложений и рекомендаций

- Нахождение расстояния и сходства между объектами
- Базовые понятия систем выработки рекомендаций, основанных на сходстве пользователей, предметов и контента
- Выработка рекомендаций по друзьям, статьям и новостным сообщениям
- Выработка рекомендаций для сайтов, подобных сайту Netflix

В современном мире голова идет кругом от выбора – почти по каждому аспекту нашего существования доступно множество вариантов. Ежедневно приходится что-то выбирать – от автомобиля до домашнего кинотеатра, от «Мистера (или Мисс) Совершенство» до адвоката или бухгалтера, от книг и газет до вики-сайтов и блогов, от фильмов до музыки и так далее. Кроме того, нас постоянно пичкают информацией, а временами – дезинформацией! В таких условиях способность порекомендовать, что выбрать, востребованна и тем более ценна, если рекомендуемый выбор близок предпочтениям того, для кого эта рекомендация предназначена.

Никто не заинтересован в положительных результатах воздействия на ваш выбор сильнее, чем фирмы, занимающиеся рекламой. *Raison d'être*[1] таких фирм – убедить вас в том, что вам *крайне необходим* товар *X* или услуга *Y*. Если товары вроде *X* или услуги вроде *Y* вас не интересуют, они лишь потратят зря время и раздосадуют вас! В этом слабое место «широковещательного» подхода традиционных методов рекламы (таких как рекламные щиты, реклама на телевидении или радио). Цель широкого оповещения – повлиять на ваши предпочтения и изменить их путем бесконечного повторения одного и того же сообщения. Альтернативным, более приятным и эффективным подходом была бы нацеленность на уже устоявшиеся ваши предпочтения. Вас соблазняли бы выбрать некий товар на основании его релевантности вашим личным желаниям и потребностям. И здесь на первый план выходят преимущества виртуальной среды и бизнеса, связанного с интеллектуальной рекламой в Интернете. Может, функциональность поиска и принесла известность фирме Google, но разбогатела она благодаря рекламе!

В этой главе мы расскажем все, что вам надо знать о построении систем выработки рекомендаций. Вы узнаете, что такое *коллаборативная фильтрация* (collaborative filtering) и системы выработки рекомендаций на основе контента. Также вы узнаете, как оптимизировать классические алгоритмы и расширить их возможности в более реалистичных приложениях. Мы начнем с описания задачи выработки рекомендации по записям в музыкальном интернет-магазине и обобщим ее, так чтобы предложенные решения можно было применять и в других обстоятельствах. Пример с музыкальным интернет-магазином прост, но конкретен и насыщен деталями, что облегчает понимание основных идей, используемых при написании системы выработки рекомендаций.

После того как все базовые понятия будут рассмотрены в применении к музыкальному интернет-магазину, мы извлечем из них больше практической пользы, представив более сложные случаи. Придерживаясь важного принципа коммерческого прозелитизма, мы рассмотрим системы выработки рекомендаций, жизненно необходимых для функционирования в Интернете пунктов видеопроката (см. наше описание сайта Netflix во введении к этой книге), книжных и универсальных магазинов.

3.1. Музыкальный интернет-магазин: основные понятия

Допустим, у вас есть интернет-магазин, где продаются музыкальные аудиозаписи в виде файлов для скачивания. Пройдя процедуру входа в соответствующее приложение, зарегистрированные пользователи мо-

[1] Смысл существования (франц.). – *Прим. перев.*

гут прослушивать отрывки из предлагаемых записей. Если конкретная запись понравилась пользователю, он может добавить ее в свою корзину для покупок и приобрести эту запись позже, когда будет готов подсчитать стоимость и оплатить все записи, отобранные им в вашем магазине. Естественно, когда пользователи завершают свой выбор или когда они оказываются на страницах нашего гипотетического приложения, мы хотим предложить им и другие записи. В наличии широкий ассортимент товаров, позволяющий выбирать из миллионов композиций, мириадов артистов и десятков музыкальных направлений – классика, духовная музыка, поп, тяжелый металл, кантри и многое другое на более или менее изысканный вкус! Вдобавок многие довольно остро реагируют на музыку, которая им не нравится. Да лучше бы вы окунули меня в Северный Ледовитый океан, чем показать мне хоть что-то похожее на рэп! У другого может быть аллергия на классику, и так далее.

Мораль сей басни такова: выводя на экран контент для пользователя, надо ориентироваться на музыкальные направления, которые нравятся этому пользователю, избегая тех, что ему не по душе. Если вам кажется, что достичь этой цели трудно, не бойтесь! Для того и существуют системы выработки рекомендаций, чтобы помочь вам донести до пользователей правильный контент!

Система выработки рекомендаций анализирует те варианты, на которых пользователь остановил свой выбор в прошлом, и может определить, в какой мере ему понравился бы некий предмет, который пользователь еще не видел. Этим можно воспользоваться для того, чтобы определить, какого типа музыку и в какой степени предпочитает ваш пользователь, устанавливая *сходство* (similarity) и сравнивая его предпочтения с характерными особенностями музыкальных направлений. Проявив больше изобретательности, мы могли бы помочь посетителям сайта организовать на нем социальную сеть, основанную на *сходстве* их музыкальных пристрастий. Таким образом, быстро выясняется, что жизненно важным функциональным элементом систем выработки рекомендаций является способность определить, насколько похожи друг на друга два (и более) пользователя или два (и более) предмета. Затем это сходство можно задействовать для предоставления рекомендаций.

3.1.1. Понятия расстояния и сходства

Возьмем какие-нибудь данные и подробно изучим названные понятия. Основные элементы, с которыми мы будем работать, – это предметы (`Item`), пользователи (`User`) и рейтинговые оценки (`Rating`). В контексте систем выработки рекомендаций сходство является мерой, позволяющей сравнить близость (proximity) двух предметов; это сравнение во многом напоминает близость двух городов, которая говорит нам о том, на каком расстоянии друг от друга города находятся географически. В случае двух городов, чтобы вычислить их географическую близость,

мы использовали бы координаты – широту и долготу. Считайте, что рейтинговые оценки (`Rating`) – это «координаты» в пространстве предметов (`Item`) или пользователей (`User`). Продемонстрируем эти понятия в действии. Выберем из списка трех пользователей `MusicUser` и назначим каждому пользователю список композиций (предметов) с гипотетическими рейтинговыми оценками этих записей.

Как принято в Интернете, рейтинговые оценки будут назначаться по пятибалльной шкале – от 1 до 5 (включительно). Назначенные нами композиции у первых двух пользователей (Frank и Constantine) получили оценки 4 и 5 – им и в самом деле понравились все выбранные нами произведения! А рейтинговые оценки третьего пользователя (Catherine) колеблются от 1 до 3. Понятно, таким образом, что мы предполагаем сходство у первых двух пользователей и их несходство с третьим пользователем. Когда мы загрузим наш пример данных в сценарий (вторая строка листинга 3.1), в нашем распоряжении окажутся данные о пользователях, композициях и рейтинговых оценках, показанные в табл. 3.1.

Таблица 3.1. Рейтинговые оценки пользователей показывают, что у пользователей Frank и Constantine наблюдается большее сходство, чем у пользователей Frank и Catherine (рис. 3.2)

Пользователь	**Композиция**	**Рейтинговая оценка**
Frank	Tears In Heaven	5
	La Bamba	4
	Mrs. Robinson	5
	Yesterday	4
	Wizard of Oz	5
	Mozart: Symphony #41 (Jupiter)	4
	Beethoven: Symphony No. 9 in D	5
Constantine	Tears in Heaven	5
	Fiddler on the Roof	5
	Mrs. Robinson	5
	What a Wonderful World	4
	Wizard of Oz	4
	Let It Be	5
	Mozart: Symphony #41 (Jupiter)	5

Catherine	Tears in Heaven	1
	Mrs. Robinson	2
	Yesterday	2
	Beethoven: Symphony No. 9 in D	3
	Sunday, Bloody Sunday	1
	Yesterday	1
	Let It Be	2

Все эти шаги можно выполнить в командной оболочке с помощью сценария, показанного в листинге 3.1.

Листинг 3.1. Малый список пользователей MusicUser с их оценками Rating музыкальных записей MusicItem

```
MusicUser[] mu = MusicData.loadExample();

mu[0].getSimilarity(mu[1],0);

mu[0].getSimilarity(mu[1],1);

mu[0].getSimilarity(mu[2],0);

mu[1].getSimilarity(mu[2],0);   ← Сходство симметрично
mu[2].getSimilarity(mu[1],0);

mu[0].getSimilarity(mu[0],0);   ← Сходство пользователя с самим собой
mu[0].getSimilarity(mu[0],1);
```

Мы предоставили два варианта определения сходства, которые инициируются путем передачи разных значений второго аргумента в метод `getSimilarity` класса `MusicUser`. Сейчас мы подробно опишем реализацию этого программного кода, но сначала рассмотрим рис. 3.1, где показаны результаты, полученные при сравнении трех пользователей друг с другом.

Согласно вычислениям, показанным на рис. 3.1, музыкальные предпочтения пользователя Frank обладают большим сходством с предпочтениями пользователя Constantine, чем с предпочтениями пользователя Catherine. Сходство между двумя пользователями не зависит от того, в каком порядке передаются аргументы в метод `getSimilarity`. Сходство пользователя Frank с самим собой равно 1.0 – это значение мы трактуем как максимальное возможное сходство между любыми двумя сущностями. Своим происхождением эти свойства обязаны тому факту, что многие показатели меры сходства основаны на расстоянии, подобном геометрическому расстоянию между двумя точками на плоскости, которое мы изучали в средней школе.

```
bsh % MusicUser[] mu = MusicData.loadExample();

bsh % mu[0].getSimilarity(mu[1],0);

 User Similarity between Frank and Constantine is equal to
0.3911406349860862

bsh % mu[0].getSimilarity(mu[1],1);

 User Similarity between Frank and Constantine is equal to
0.22350893427776353

bsh % mu[0].getSimilarity(mu[2],0);

 User Similarity between Frank and Catherine is equal to
0.004197074413470947

bsh % mu[1].getSimilarity(mu[2],0);

 User Similarity between Constantine and Catherine is equal to
0.0023790682635077554

bsh % mu[2].getSimilarity(mu[1],0);

 User Similarity between Catherine and Constantine is equal to
0.0023790682635077554

bsh % mu[0].getSimilarity(mu[0],0);

 User Similarity between Frank and Frank is equal to 1.0

bsh % mu[0].getSimilarity(mu[0],1);

User Similarity between Frank and Frank is equal to 1.0
```

Рис. 3.1. *Вычисление сходства пользователей на основе данных из табл. 3.1. Понятно, что пользователи Frank и Constantine больше сходятся во взглядах, чем пользователи Frank и Catherine*

В общем случае математические расстояния обладают следующими четырьмя важными свойствами:

- Все расстояния больше или равны нулю. Обычно, как в случае пользователя `MusicUser`, мы накладываем на показатели меры сходства то ограничение, что их значения, подобно расстояниям, должны быть неотрицательными. Фактически, мы ограничиваем показатели сходства интервалом [0, 1].

- Расстояние между любыми двумя точками, например A и B, равно нулю тогда и только тогда, когда точка A совпадает с точкой B. В рассматриваемом примере и на основании нашей реализации понятия сходства это свойство находит свое отражение в том факте, что если рейтинговые оценки, выставленные двумя пользователями, в точности совпадают, сходство между этими пользователями будет равно 1.0. Это подтверждает рис. 3.1, где мы дважды использовали одного и того же пользователя, чтобы показать, что сходство равно 1.0. Разумеется, вы можете создать четвертого пользователя и доказать, что сходство будет равно 1, при условии, что пользователи прослушали одни и те же композиции.
- Третьим свойством расстояний является *симметрия* (symmetry) – расстояние между точками A и B в точности равно расстоянию между точками B и A. Это значит, что если музыкальные вкусы пользователя Catherine схожи с музыкальными вкусами пользователя Constantine, верно будет и обратное, в той же самой мере. Следовательно, нам обычно требуется, чтобы мера сходства сохраняла свойство симметрии расстояний в отношении своих аргументов.
- Четвертое свойство математических расстояний – это *аксиома неравенства треугольника* (triangle inequality), потому что она соотносит расстояния между тремя точками. В терминах математики, если $d(A, B)$ – расстояние между точками A и B, то аксиома треугольника утверждает, что $d(A, B) <= d(A, C) + d(C, B)$ для любой третьей точки C. На рис. 3.1 сходство между пользователями Frank и Constantine равно 0.391, а сходство между пользователями Constantine и Catherine равно 0.002, тогда как сходство между пользователями Frank и Catherine равно 0.004, что меньше суммы показателей сходства в первых двух случаях. Однако в общем случае для наших показателей сходства это свойство не сохраняется.

Смягчение действия четвертого фундаментального свойства расстояний при переходе к сходству – это хорошо; нет настоятельной необходимости переносить на сходство все свойства расстояний. Мы должны всегда проявлять осмотрительность, чтобы привлеченный математический аппарат гарантированно не противоречил тому, что представляется нам рациональным. Для аксиомы неравенства треугольника в отношении сходства есть вековой давности контрпример, приписываемый Уильяму Джемсу[1]: «Пламя похоже на луну, потому что оба они светятся, а луна похожа на мяч, потому что они оба круглые, но вопреки ак-

[1] Уильям Джемс (1842–1910) – американский философ и психолог, один из основателей и ведущий представитель прагматизма и функционализма. – *Прим. ред.*

сиоме треугольника пламя не похоже на мяч»[1]. Для ознакомления с интересным толкованием сходства с точки зрения познания рекомендуем обратиться к работе [Estes, W.K. Classification and Cognition].

В верхней части рис. 3.2 мы визуально представляем сходство между пользователями Frank и Constantine графиками рейтинговых оценок, выставленных этими пользователями композициям, которые оценивали они оба. Чем ближе эти линии рейтинговых оценок, тем больше сходство между пользователями; чем дальше расходятся графики, тем меньше сходство. В нижней части рис. 3.2, где мы сравниваем рейтинговые оценки пользователей Frank и Catherine, их графики далеко отстоят друг от друга, что соответствует низкому значению показателя сходства, полученному нами в результате вычислений.

Графики пользователей Frank и Constantine располагаются близко друг к другу, что означает сходство между этими пользователями. Посмотрев на программный код метода `plot` класса `MusicUser`, вы увидите, что мы отсортировали выставленные пользователями оценки, чтобы подчеркнуть имеющееся различие. При большом количестве рейтинговых оценок будет видно, что расхождение между этими двумя графиками увеличивается по мере перемещения по графику слева направо.

Графики рейтинговых оценок, представленные на рис. 3.2, ясно показывают некую взаимосвязь понятий «расстояние» и «сходство». Чем больше расстояние между двумя графиками, тем меньше сходство между двумя пользователями; чем меньше расстояние между двумя графиками, тем больше сходство между двумя пользователями. Как мы увидим в следующем разделе, оценка сходства часто, хотя и не обязательно, включает оценку некоего расстояния. Понятие расстояния привычнее. Понятия расстояния и сходства – частные случаи общего понятия меры.

3.1.2. Подробнее о вычислении сходства

Теперь проанализируем программный код, который помог нам выявить сходство между пользователями, и рассмотрим, как можно вычислить сходство. Программный код в листинге 3.2 демонстрирует подробности метода `getSimilarity`, который получает два аргумента. Первый аргумент обеспечивает ссылку на другого пользователя, а второй указывает тип сходства, который мы хотим использовать.

Листинг 3.2. Два показателя сходства, вычисление которых реализовано в методе getSimilarity класса MusicUser

```
public double getSimilarity(MusicUser u, int simType) {

  double sim=0.0d;
  int commonItems=0;
```

[1] Источник: ScholarPedia (*http://www.scholarpedia.org/article/Similarity_measures*).

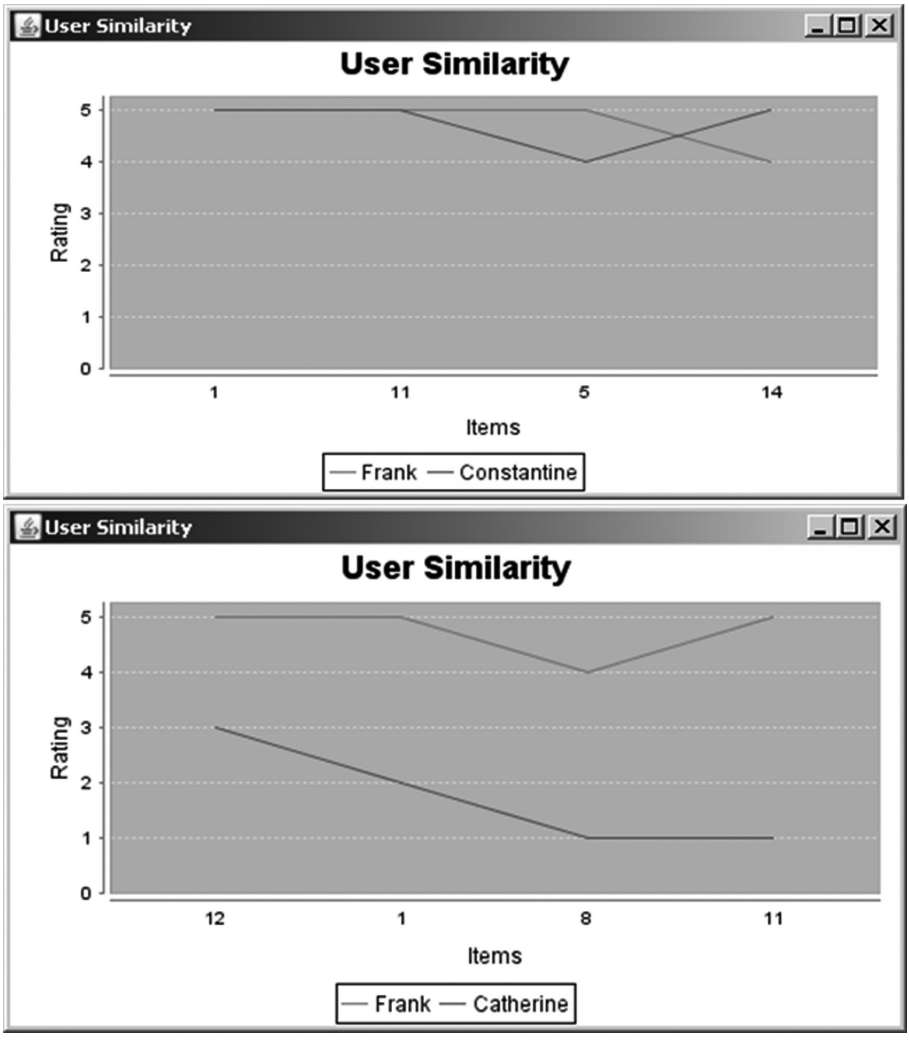

Рис. 3.2. *Сходство двух пользователей можно оценить, измерив длину участка, на котором два аналогичных графика перекрываются. Так, пользователи Frank и Constantine (вверху) обладают большим сходством, чем пользователи Frank и Catherine (внизу).*

```
 switch(simType) {

case 0:
   for (Rating r : this.ratingsByItemId.values()) {

        for (Rating r2 : u.ratingsByItemId.values()) {
```

Идентификация всех общих предметов, оцененных обоими пользователями

```
            // Ищем этот же предмет
            if ( r.getItemId() == r2.getItemId() ) {
             commonItems++;
             sim += Math.pow((r.getRating()-r2.getRating()),2);
             }
        }
    }
```

Возведение в квадрат разности рейтинговых оценок и их суммирование

```
// Если общих предметов нет, мы не можем сказать, обладают эти пользователи
// сходством или нет. Следовательно, позволим методу вернуть значение 0.
if (commonItems > 0) {

    sim = Math.sqrt(sim/(double)commonItems);

    // Значение показателя сходства должно находиться в интервале от 0 до 1:
    // 0 - максимальное отсутствие сходства между двумя пользователями,
    // 1 - предпочтения этих пользователей идентичны
    //
    sim = 1.0d - Math.tanh(sim);
}

break;

case 1:
  for (Rating r : this.ratingsByItemId.values()) {
    for (Rating r2 : u.ratingsByItemId.values()) {
```

Идентификация всех общих предметов, оцененных обоими пользователями

```
            // Поиск того же предмета
            if ( r.getItemId() == r2.getItemId() ) {
             commonItems++;
             sim += Math.pow((r.getRating()-r2.getRating()),2);
            }
        }
    }
```

Возведение в квадрат разности рейтинговых оценок и их суммирование

```
// Если общих предметов нет, мы не можем сказать, обладают пользователи
// сходством или нет. Следовательно, позволим методу вернуть значение 0
if (commonItems > 0) {

    sim = Math.sqrt(sim/(double)commonItems);

    // Значение показателя сходства должно находиться в интервале от 0 до 1:
    // 0 - максимальное отсутствие сходства между двумя пользователями,
    // 1 - предпочтения этих пользователей идентичны
    //
    sim = 1.0d - Math.tanh(sim);

    // Поиск максимального количества предметов,
    // которые могут быть общими для этих пользователей
    int maxCommonItems =
➥   Math.min(this.ratingsByItemId.size(), u.ratingsByItemId.size());
```

```
        // Уточнение показателя сходства с учетом значимости общих термов как
        // отношения числа общих предметов к числу всех возможных общих предметов

        sim = sim * ((double)commonItems/(double)maxCommonItems);
    }

    break;

    } // Завершение блока switch

    // Вывод результатов
    System.out.print("\n"); // Только чтобы красиво напечатать в среде Shell
    System.out.print(" User Similarity between");
    System.out.print(" "+this.getName());
    System.out.print(" and "+u.getName());
    System.out.println(" is equal to "+sim);
    System.out.print("\n"); // Только чтобы красиво напечатать в среде Shell

    return sim;
    }
```

Мы включили в этот программный код две формулы, позволяющие вычислить сходство, чтобы показать: понятие сходства довольно универсально и растяжимо. Давайте проанализируем основные этапы вычисления сходства по этим формулам. Прежде всего, мы берем разности между всеми рейтинговыми оценками общих музыкальных записей, которые оценивали оба пользователя, возводим эти разности в квадрат и суммируем полученные значения. Корень квадратный из полученной суммы называется *евклидовым расстоянием* (Euclidean distance) и на данном этапе этого показателя недостаточно, чтобы обеспечить меру сходства. Как мы уже сказали, понятия расстояния и сходства взаимосвязаны в том смысле, что чем меньше евклидово расстояние, тем больше сходство двух пользователей. Можно утверждать, что несочетаемость такого упорядочения с понятием сходства легко исправить. Например, мы могли бы заявить, что прибавим к евклидовой оценке 1 и инвертируем полученное значение.

На первый взгляд кажется, что инвертирование расстояния (после прибавления постоянного значения 1) могло бы сработать. Но эта невинная с виду модификация не лишена недостатков. Если два пользователя прослушали только одну музыкальную запись, и один из пользователей выставил оценку 1, а другой – 4, сумма разностей этих оценок, возведенная в квадрат, будет равна 9. В этом случае наивное (простое) сходство на основе евклидова расстояния обеспечило бы в результате значение показателя сходства 0,25. Такое же значение показателя сходства можно получить и в других случаях. Если два пользователя прослушали по три музыкальных записи и оценки, выставленные ими этим трем записям, отличались на 1 (для каждой записи), показатель сходства пользователей также был бы равен 0,25, согласно наивной

мере сходства. Интуитивно мы ожидали, что сходство этих двух пользователей будет больше, чем сходство пользователей, прослушавших единственную запись и разошедшихся во мнении на 3 единицы (из 5 возможных!).

Наивное сходство «выпячивает» значения показателя сходства для малых расстояний (потому что мы прибавили 1), тогда как большие расстояния (намного превышающие 1) остаются в прежнем состоянии. Что если мы прибавим другое значение? Общая формула наивного сходства: $y = beta / (beta + x)$, где $beta$ – это наш третий параметр, а x – евклидово расстояние. На рис. 3.3 показано, как выглядело бы наивное сходство для различных значений параметра $beta$ – 1 и 2.

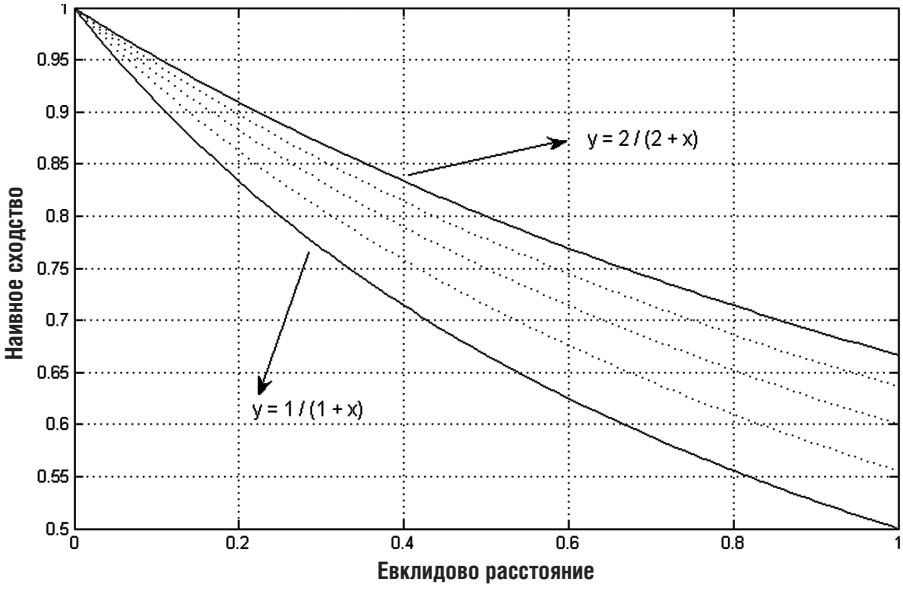

Рис. 3.3. *Кривые наивного сходства как функции евклидова расстояния*

Принимая во внимание недостатки наивного сходства, рассмотрим первый вариант определения сходства между двумя пользователями, как показано в листинге 3.2 в блоке `case 0`. Если пользователи оценивали одни и те же композиции, мы делим сумму квадратов разностей выставленных ими оценок на число общих композиций, берем положительный квадратный корень этой суммы и передаем полученное значение в специальную функцию. Она нам уже встречалась: это функция гиперболического тангенса. Значение гиперболического тангенса вычитается из 1, так что полученное нами окончательное значение для показателя сходства попадает в интервал от 0 до 1, где 0 предполагает отсутствие сходства, а 1 – наивысшую степень сходства. Вуаля! Мы по-

лучили свой первый вариант определения сходства пользователей на основе выставленных ими рейтинговых оценок.

Второй вариант определения сходства, который мы представляем в блоке `case 1` листинга 3.2, уточняет первую оценку сходства, принимая во внимание отношение числа общих предметов к числу всех возможных общих предметов. Это эвристика, смысл которой понятен интуитивно. Если я прослушал 30 композиций, а вы – 20, у нас могло бы быть до 20 общих композиций. Допустим, у нас только 5 общих композиций, и уровень совпадения наших оценок этих композиций довольно высок, что хорошо, но почему у нас нет других общих композиций? Не должно ли их отсутствие как-то отразиться на нашем сходстве? Это именно тот вопрос задачи, который мы пытаемся решить с помощью второй формулы сходства. Другими словами, на степень нашего сходства как слушателей должно как-то влиять количество композиций, прослушанных нами обоими.

3.1.3. Какую из формул вычисления сходства предпочесть?

К этому моменту должно быть ясно, что есть несколько формул, позволяющих установить наличие сходства между двумя пользователями или предметами, если уж на то пошло. Кроме двух вариантов вычисления сходства, реализованных в классе `MusicUser`, мы могли бы использовать метрическую формулу, известную как *сходство Жаккарда* (Jaccard similarity), между пользователями, которое определяется как отношение пересечения к объединению множеств предметов этих пользователей, – или, в случае сходства предметов, как отношение пересечения к объединению множеств пользователей. Другими словами, сходство Жаккарда между двумя множествами A и B определяется следующим псевдокодом: *Жаккард* = *пересечение* (*A*, *B*) / *объединение* (*A*, *B*). Мы будем использовать сходство Жаккарда в следующих разделах и представим также еще несколько формул, позволяющих вычислить сходство, в разделе «Сделать» в конце этой главы.

Разумеется, вполне естественно поинтересоваться: «Какая формула для вычисления сходства – наиболее подходящая?» Как всегда, ответом будет: «Это зависит от...» В данном случае это зависит от ваших данных. В одном из нескольких масштабных сравнений показателей сходства [Spertus E., Sahami M., Buyukkokten O.] из семи показателей наилучшие эмпирические результаты продемонстрировало простое евклидово сходство на основе расстояния, несмотря на то что другие формулы были сложнее и интуитивно предполагалось, что они лучше справятся с задачей. В этом сравнении измерения проводились на основе 1 279 266 переходов по ссылкам на рекомендации соответствующего сообщества, выполненных начиная с 22 сентября по 21 октября 2004 года, на веб-

сайте социальной сети Orkut (*http://www.orkut.com*); ссылки на подробную информацию – в конце главы.

Мы не советуем вам выбирать показатель сходства случайным образом, но если вы испытываете нехватку времени, используйте формулу, похожую на те две, которые реализованы в классе `MusicUser`, – евклидово сходство или сходство Жаккарда. Они должны обеспечить вам достаточно приемлемые результаты. Вам следует попытаться понять природу ваших данных, а также, что значит для двух пользователей или для двух предметов – быть похожими. Не понимая причин, по которым конкретный показатель (формула) сходства плох или хорош, вы готовите себе неприятность. Чтобы подчеркнуть эту важную мысль, задумайтесь о таком распространенном неправильном представлении, как «кратчайшее расстояние между двумя точками – это соединяющая их прямая». Это утверждение истинно только для того, что мы называем геометрией «на плоскости», вроде площадки для футбольного поля. Чтобы убедиться в этом, сравните расстояние, которое придется пройти при переходе через высокую, но не широкую в основании гору, с обходом этой же горы по периметру ее основания. Для гор, диапазон размеров которых широк, «прямая» не является кратчайшим путем.

Итак, один из краеугольных камней при выработке рекомендаций – способность измерить сходство между любыми двумя пользователями, а также между любыми двумя предметами. Мы предоставили несколько критериев сходства, которые вы можете использовать «в готовом виде», а музыкальный магазин иллюстрирует типичную структуру данных, с которыми вам придется иметь дело, чтобы создать рекомендации. Теперь мы переходим к анализу типов систем выработки рекомендаций и к изучению того, как они работают.

3.2. Как работают системы выработки рекомендаций?

Вооружившись хорошим пониманием того, что означает сходство между двумя пользователями или двумя предметами, мы можем продолжить наше описание систем выработки рекомендаций. Вообще говоря, есть две категории систем выработки рекомендаций. Первую называют *коллаборативной фильтрацией* (collaborative filtering, CF). В первой инкарнации методика CF появилась (в 1992 году) в экспериментальной почтовой системе, разработанной в исследовательском центре Xerox Palo Alto Research Center (PARC) Гольдбергом (Goldberg) и др. Методика CF опирается на информационные «крошки», которые пользователь оставляет после себя, взаимодействуя с программной системой. Как правило, такими «крошками» являются выставленные пользователем рейтинговые оценки, например оценки музыкальных записей, рассмотренные нами в предыдущем разделе. Коллаборативная фильтрация не

ограничивается одномерными или только дискретными переменными; ее главная отличительная черта – зависимость от поведения пользователя в прошлом, а не от контента каждого предмета из интересующей коллекции. Методика CF не требует ни знания предметной области, ни предварительной работы по сбору и анализу данных для выработки рекомендаций.

Вторая основная категория – это системы выработки рекомендаций на основе анализа контента, ассоциированного с предметами или пользователями либо и с теми, и с другими. Главная отличительная черта подхода, основанного на контенте, – накопление и анализ информации, относящейся как к пользователям, так и к предметам. Эту информацию могут предоставлять или программная система, или внешние источники. Система может собирать информацию о пользователях *явно*, когда пользователи отвечают на вопросы настоятельно предлагаемых опросных листов, или *неявно*, путем анализа профиля пользователя или склонностей, которые пользователь демонстрирует при чтении новостных сообщений, электронной почты, блогов и так далее.

В категории CF мы опишем выработку рекомендаций на основе сходства пользователей и предметов. Мы также рассмотрим категорию рекомендаций, выдаваемых на основе контента, охватывая, таким образом, все известные системы выработки рекомендаций.

3.2.1. Рекомендации на основе сходства пользователей

Есть древнегреческая пословица (с похожими вариантами почти в каждой из мировых культур): «Скажи мне, кто твой друг, и я скажу, кто ты». Коллаборативная фильтрация на основе близости обладающих сходством пользователей в большей или меньшей степени является алгоритмическим воплощением этой пословицы. Чтобы вычислить рейтинговую оценку, которую конкретный пользователь может выставить для данного предмета, мы рассматриваем рейтинговые оценки, выставленные этому же предмету пользователями (соседями, или друзьями, если хотите), похожими на этого пользователя. Затем рейтинговая оценка каждого друга умножается на весовой коэффициент, и эти взвешенные значения суммируются. Да, идея проста!

В листинге 3.3 показана последовательность шагов, демонстрирующая процесс создания и применения системы выработки рекомендаций, которую мы назвали `Delphi`. Во-первых, необходимо сформировать данные, с которыми мы будем работать. Мы создаем пример данных, присваивая всем пользователям их рейтинговые оценки музыкальных записей. Для каждого пользователя случайным образом подбирается набор музыкальных записей, который на 80% совпадает с записями, имеющимися в нашем музыкальном интернет-магазине. Для каждой назначенной пользователю музыкальной записи произвольным образом выбирается рейтинговая оценка, равная 4 или 5, если имя пользо-

вателя начинается с буквы от A до D (включительно), в остальных случаях рейтинговая оценка будет равна 1, 2 или 3.

Итак, мы организовали две большие группы пользователей со сходными предпочтениями; эти группы позволят быстро оценить результаты работы нашей системы.

Листинг 3.3. Выработка рекомендаций на основе сходства пользователей

```
BaseDataset ds = MusicData.createDataset();      ← Создание набора данных о музыкальных записях
ds.save("C:/iWeb2/deploy/data/ch3_2_dataset.ser");   ← Сохранение его на будущее

Delphi delphi = new Delphi(ds,RecommendationType.USER_BASED);   ← Создание системы выработки рекомендаций
delphi.setVerbose(true);

MusicUser mu1 = ds.pickUser("Babis");
delphi.findSimilarUsers(mu1);

MusicUser mu2 = ds.pickUser("Lukas");
delphi.findSimilarUsers(mu2);

delphi.recommend(mu1);   ← Рекомендация нескольких композиций
```

Поиск похожих пользователей

В первой строке создается набор данных о пользователях и рейтинговых оценках композиций, как мы это описали ранее. Логика этого программного кода проста, и вы можете модифицировать данные в классе `MusicData`, как вам удобнее. Во второй строке мы запоминаем используемый в этом примере набор данных, чтобы можно было на него ссылаться и дальше. В третьей строке создается экземпляр системы выработки рекомендаций `Delphi`, а в четвертой строке для этой системы устанавливается режим отображения всей информации, позволяющий нам видеть подробности процесса получения результатов. Обратите внимание: конструкторы `Delphi` используют интерфейс `Dataset`, а не классы из наших примеров. Вы можете использовать этот интерфейс в своей реализации прямо «из коробки» – точнее, из Java-архива (JAR).

На рис. 3.4 показаны результаты выполнения нашего сценария, предусмотренного для метода `findSimilarUsers`. В первом случае имя пользователя начинается с буквы B, а имена всех выбранных для него друзей начинаются с букв от A до D. Во втором случае имя пользователя начинается с буквы J, а все выбранные для него друзья имеют имена, начинающиеся с букв от E до Z. В обоих случаях мы получаем результаты, которые не противоречат нашим ожиданиям.

Итак, похоже на то, что наша система выработки рекомендаций работает хорошо! Заметим, что степень сходства между друзьями в первом случае выше, чем степень сходства пользователей в группе, соответствующей второму случаю, потому что рейтинговые оценки распределялись только между двумя значениями (4 и 5) в первом случае, а во втором они распределялись по трем значениям (1, 2 и 3). Такого рода

проверки на здравомыслие полезны, и вы всегда должны бдительно следить за тем, что возвращает интеллектуальный алгоритм; он был бы не слишком интеллектуальным, если бы возвращал значение, не отвечающее критерию здравого смысла, верно?

В дополнение к сказанному, на рис. 3.4 показаны результаты выработки рекомендаций музыкальных произведений для одного из пользователей, а также предсказанные рейтинговые оценки для каждой рекомендации. Обратите внимание: хотя рейтинговые оценки, выставляемые пользователями, являются целыми числами, система выработки рекомендаций использует для своих предсказаний тип `double`. Это объясняется тем, что предсказание выражает только степень доверия к рейтинговой оценке, а не саму оценку. Вас, может быть, удивляет,

```
bsh % MusicUser mu1 = ds.pickUser("Bob");
bsh % delphi.findSimilarUsers(mu1);

Top Friends for user Bob:

name: Babis                                , similarity: 0.692308
name: Alexandra                            , similarity: 0.666667
name: Bill                                 , similarity: 0.636364
name: Aurora                               , similarity: 0.583333
name: Charlie                              , similarity: 0.583333

bsh % MusicUser mu2 = ds.pickUser("John");
bsh % delphi.findSimilarUsers(mu2);

Top Friends for user John:

name: George                               , similarity: 0.545455
name: Jack                                 , similarity: 0.500000
name: Elena                                , similarity: 0.461538
name: Lukas                                , similarity: 0.454545
name: Frank                                , similarity: 0.416667

bsh % delphi.recommend(mu1);

Recommendations for user Bob:

Item: I Love Rock And Roll          , predicted rating: 4.922400
Item: La Bamba                      , predicted rating: 4.758600
Item: Wind Beneath My Wings         , predicted rating: 4.540900
Item: Sunday, Bloody Sunday         , predicted rating: 4.526800
```

Рис. 3.4. *Система выработки рекомендаций Delphi ищет друзей и предоставляет рекомендации, основанные на сходстве пользователей*

что сайты не позволяют выставлять оценки, не являющиеся целыми числами, или, в равной степени смягчив ограничение, не предлагают в качестве оценки значение из более широкого интервала, например, от 1 до 10 или даже от 1 до 100. Мы вернемся к этому вопросу в одном из пунктов нашего раздела «Сделать» в конце этой главы.

Заметим, что система выработки рекомендаций корректно назначила значения 4 и 5, поскольку все пользователи, чьи имена начинаются с буквы от A до D, выставили оценку 4 или 5.

Каким образом класс `Delphi` пришел к этим выводам? Как ему удается найти похожих пользователей (друзей) для любого данного пользователя? Как ему удается рекомендовать музыкальные произведения из списка произведений, которые пользователь никогда не слышал? Чтобы разобраться в происходящем, давайте пройдем основные этапы. Системы выработки рекомендаций, основанные на коллаборативной фильтрации, функционируют в два этапа. Сначала они вычисляют сходство между пользователями или между предметами. Затем эти системы используют средневзвешенное значение, чтобы вычислить рейтинговую оценку, которую выставил бы пользователь тому предмету, с которым он еще не сталкивался.

Вычисление сходства пользователей

Поскольку мы имеем дело с рекомендациями на основе сходства пользователей, первое, что делает для нас система `Delphi`, – вычисляет сходство между пользователями. Это вычисление показано в листинге 3.4, где демонстрируется программный код метода `calculate` класса `UserBasedSimilarity` – вспомогательного класса, используемого в классе `Delphi`. Обратите внимание: двойной цикл был оптимизирован, чтобы учитывалась симметрия матрицы сходства; эту и еще одну оптимизацию мы рассмотрим после листинга.

Листинг 3.4. Класс UserBasedSimilarity: вычисление сходства пользователей

```
protected void calculate(Dataset dataSet) {

  int nUsers = dataSet.getUserCount();   ← Определение размера матрицы сходства

  int nRatingValues = 5;   ← Определение размера матрицы счетчика рейтинговых оценок

  similarityValues = new double[nUsers][nUsers];

  if( keepRatingCountMatrix ) {
      ratingCountMatrix = new RatingCountMatrix[nUsers][nUsers];
  }

  // Если вместо идентификатора userId используется индекс, генерируем
значение индекса для каждого userId
  if( useObjIdToIndexMapping ) {

      for(User u : dataSet.getUsers() ) {
```

```
            idMapping.getIndex(String.valueOf(u.getId()));
        }
    }

    for (int u = 0; u < nUsers; u++ ) {
        int userAId = getObjIdFromIndex(u);
        User userA = dataSet.getUser(userAId);

        for (int v = u + 1; v < nUsers; v++) {   ← ❶ Матрица сходства

            int userBId = getObjIdFromIndex(v);
            User userB = dataSet.getUser(userBId);

            RatingCountMatrix rcm =
➥           new RatingCountMatrix(userA, userB, nRatingValues);   ← Совпадение рейтинговых оценок двух пользователей

            int totalCount = rcm.getTotalCount();
            int agreementCount = rcm.getAgreementCount();

            if (agreementCount > 0) {   ← Вычисление значения показателя сходства или присваивание ему нулевого значения

               similarityValues[u][v] =
➥              (double) agreementCount / (double) totalCount;

            } else {
                similarityValues[u][v] = 0.0;
            }

            // Для больших наборов данных
            if( keepRatingCountMatrix ) {
                ratingCountMatrix[u][v] = rcm;
            }
        }

        // Для u == v присваиваем показателю сходства значение 1
        // Для этого случая RatingCountMatrix не был создан
        similarityValues[u][u] = 1.0;   ← ❶ Матрица сходства
    }
}
```

❶ Вот оптимизация, о которой мы упомянули ранее. Вы ожидали, что в первом цикле будет выбираться первый пользователь, а во втором – все остальные. Но в этом листинге при организации второго цикла использован тот факт, что матрица сходства является симметричной. Это означает следующее: если пользователь A обладает сходством с пользователем B и значение показателя в этом случае равно X, то пользователь B будет обладать сходством с пользователем A с показателем сходства, значение которого равно X. В рассматриваемом программном коде мы не вычисляем сходство объекта пользователя с ним самим, поскольку оно всегда будет равно 1. Эти две оптимизации программного кода просто отражают фундаментальные свойства, которым должен подчиняться каждый критерий сходства, как утверждалось в разделе 3.1.1.

Как видите, сходство определяется метрикой Жаккарда, в которой совпадение оценок представляется как пересечение двух множеств рейтинговых оценок, а для представления итогового счетчика используется объединение двух множеств рейтинговых оценок. Значения показателей сходства хранятся в двумерном массиве типа `double`. Однако сходство является симметричным свойством, что просто означает: если я похож на вас, значит, вы похожи на меня, независимо от того, как было определено наше сходство. Поэтому понятно, что мы можем использовать значения показателя сходства намного эффективнее, если воспользуемся разреженными матрицами или какой-то другой структурой, которая по своей идее предназначена для хранения только половины всех значений; эта последняя структура формально называется *верхней треугольной матрицей* (upper triangular form). С точки зрения вычислений, мы уже задействовали этот фактор в программном коде (см. листинг 3.4). Еще раз заметим, что во втором цикле реализован перебор не всех пользователей, но начиная с пользователя, который следует в нашем списке за пользователем из внешнего цикла.

Вычисление сходства каждой пары пользователей доверено вспомогательному классу, который мы назвали `RatingCountMatrix`. Задача этого класса – обеспечить хранение рейтинговых оценок одного пользователя относительно оценок других пользователей в удобной табличной форме и позволить нам легко и прозрачно вычислить окончательное значение показателя сходства. В листинге 3.5 представлен программный код класса `RatingCountMatrix`.

Листинг 3.5. Хранение распределения совпадений в оценках двух пользователей в табличной форме

```
public class RatingCountMatrix implements Serializable {

   private int matrix[][] = null;

   public RatingCountMatrix(Item itemA, Item itemB,
➥     int nRatingValues) {

        init(nRatingValues);
                                          Вычисление сходства
        calculate(itemA, itemB);          на основе предметов
   }

   public RatingCountMatrix(User userA, User userB,
➥     int nRatingValues) {

        init(nRatingValues);
                                          Вычисление сходства
        calculate(userA, userB);          на основе пользователей
   }

   private void init(int nSize) {
        // Отправная точка - все элементы имеют нулевые значения
        matrix = new int[nSize][nSize];
```

Инициализация матрицы счетчиков рейтинговых оценок

```
}

private void calculate(Item itemA, Item itemB) {

    for (Rating ratingForA : itemA.getAllRatings()) {

        // Проверка: оценивал ли тот же самый пользователь предмет itemB
        Rating ratingForB =
➥   itemB.getUserRating(ratingForA.getUserId());

        if (ratingForB != null) {

            int i = ratingForA.getRating() - 1;
            int j = ratingForB.getRating() - 1;

            matrix[i][j]++;
        }
    }
}

private void calculate(User userA, User userB) {

    for (Rating ratingByA : userA.getAllRatings()) {

        Rating ratingByB =
➥   userB.getItemRating(ratingByA.getItemId());

        if (ratingByB != null) {

            int i = ratingByA.getRating() - 1;
            int j = ratingByB.getRating() - 1;

            matrix[i][j]++;
        }
    }
}

public int getTotalCount() {

    int ratingCount = 0;
    int n = matrix.length;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            ratingCount += matrix[i][j]; :)
        }
    }
    return ratingCount;
}

public int getAgreementCount() {
    int ratingCount = 0;
    for (int i = 0, n = matrix.length; i < n; i++) {
        ratingCount += matrix[i][i];
    }
```

Вычисление сходства на основе предметов

Вычисление сходства на основе пользователей

Вспомогательные методы для получения различных счетчиков

```
        return ratingCount;
    }

    public int getBandCount(int bandId) {

        int bandCount = 0;
        for (int i = 0, n = matrix.length; (i + bandId) < n; i++) {
            bandCount += matrix[i][i + bandId];
            bandCount += matrix[i + bandId][i];
        }
        return bandCount;
    }
}
```

Вспомогательные методы для получения различных счетчиков

Основным компонентом этого класса является двумерный массив типа `int` (размером 5 на 5, в данном случае), где хранится коэффициент совпадения рейтинговых оценок, выставленных двумя пользователями. Допустим, оба пользователя A и B прослушали по 10 записей и выставили одинаковые оценки для 6 из них, а остальные оценили по-разному. Все элементы матрицы инициализируются нулями; для каждого случая совпадения оценок мы прибавляем 1 к тому элементу матрицы, который находится в строке и столбце совпавшей оценки. Так, если три совпадения были получены для оценки со значением 4, а остальные три – для оценки 5, то оба элемента `matrix[3][3]` и `matrix[4][4]` будут иметь значение 3. В общем случае, просуммировав диагональные элементы массива `matrix`, вы выясните, сколько раз эти два пользователя выставили одинаковые оценки.

Такой способ хранения пользовательских рейтинговых оценок имеет ряд преимуществ. Во-первых, с рейтинговыми оценками из интервала от 1 до 10 (или, если хотите, до 100) можно обращаться точно так же, как с оценками пятибалльной шкалы, от 1 до 5. Во-вторых, как мы увидим позже, вы получаете возможность вывести более сложные критерии сходства, учитывающие не только число совпадений оценок двух пользователей, но также число несовпадений и степень расхождения. В-третьих, можно обобщить и преобразовать эту матричную форму в более универсальный объект, которым может быть не просто двумерный массив, но более сложная структура; что, возможно, является желательным в ситуации, где ваша оценка полагается не только на простое ранжирование.

Внутренние механизмы класса Delphi

Итак, код листинга 3.4 получил исчерпывающее объяснение. Значение показателя сходства между пользователем *A* и пользователем *B* в этом случае – это просто частное от деления числа совпадений в оценках пользователей *A* и *B* на суммарное число оценок, выставленных обоими пользователями конкретному предмету. Итак, мы в одном шаге от выработки рекомендаций.

Листинг 3.6. Delphi: выработка рекомендаций на основе сходства пользователей

```
public List<PredictedItemRating> recommend(User user, int topN) {

  List<PredictedItemRating> recommendations =
➥  new ArrayList<PredictedItemRating>();

  for (Item item : dataSet.getItems()) {        ← Циклический перебор всех предметов

       // Рассматриваются только те предметы,
       // которые пользователь еще не оценивал
       if (user.getItemRating(item.getId()) == null) {

          double predictedRating = predictRating(user, item);   ← Рейтинговые оценки, предсказанные для этого пользователя

   if (!Double.isNaN(predictedRating)) {
       recommendations.add(new PredictedItemRating(user.getId(),   ← Добавление предсказания в список кандидатов на рекомендацию
                          item.getId(), predictedRating));
   }
       }
   }

  Collections.sort(recommendations);          ← Сортировка кандидатов на рекомендацию

  Collections.reverse(recommendations);       ← Сортировка кандидатов на рекомендацию

  List<PredictedItemRating> topRecommendations =
➥  new ArrayList<PredictedItemRating>();

  for(PredictedItemRating r : recommendations) {   ← Выборка N первых рекомендаций
      if( topRecommendations.size() >= topN ) {
                // Получено достаточно рекомендаций.
                break;
      }
       topRecommendations.add(r);
    }
    return recommendations;
 }
```

В листинге 3.6 показан метод верхнего уровня `recommend` класса `Delphi`, который вызывается для того, чтобы предоставить рекомендации, как мы видели в листинге 3.3. В этом методе из рассмотрения выбрасываются предметы, которые пользователь уже оценивал. Может быть, в этом есть необходимость, а может, и нет; продумайте ваши требования, прежде чем использовать наш программный код «как есть». Если вам потребуется внести изменения, можно изменить функциональное поведение этого метода: например, дополнить первую конструкцию `if` предложением `else`.

Метод `recommend` делегирует предсказание рейтинговой оценки пользователя `user` (первый аргумент) для каждого предмета `item` в метод `predictRating(user, item)`, который в свою очередь делегирует вычисление взве-

шенного среднего в метод `estimateUserBasedRating`. В листинге 3.7 показан метод `predictRating(user, item)`. Задача этого метода – создать «фасад», который скрывает все возможные реализации процедуры получения оценки сходства, такие как сходство на основе пользователей, сходство на основе предметов и так далее. Некоторые случаи предполагаются, но не реализованы, так что вы можете над ними поработать!

Листинг 3.7. Предсказание пользовательской рейтинговой оценки предмета

```
public double predictRating(User user, Item item) {

    switch (type) {
        case USER_BASED:
            return estimateUserBasedRating(user, item);

        case ITEM_BASED:
            return estimateItemBasedRating(user, item);

        case USER_CONTENT_BASED:
            throw new IllegalStateException(
➥   "Not implemented similarity type:" + type);

        case ITEM_CONTENT_BASED:
            throw new IllegalStateException(
➥   "Not implemented similarity type:" + type);

        case USER_ITEM_CONTENT_BASED:
            return MAX_RATING * similarityMatrix
➥   .getValue(user.getId(), item.getId());
        }
        throw new RuntimeException("Unknown type:" + type);
    }
```

Метод `estimateUserBasedRating` – это реализация для предсказания пользовательских рейтинговых оценок на основе сходства пользователей. Если пользовательские рейтинговые оценки нам известны, незачем что-либо вычислять. Такая ситуация невозможна в той последовательности выполнения, которую мы описали в листинге 3.6, потому что в ней инициируется обращение к методу только для тех предметов, которые пользователь еще не оценил. Но указанный программный код был написан таким образом, что обрабатываются также независимые обращения к этому методу.

Листинг 3.8. Оценка рейтинга на основе сходства пользователей

```
private double estimateUserBasedRating(User user, Item item) {

   double estimatedRating = Double.NaN;

   int itemId = item.getId();
   int userId = user.getId();
```

```
    double similaritySum = 0.0;

    double weightedRatingSum = 0.0;

    // Проверка: оценивал ли пользователь этот предмет ранее
    Rating existingRatingByUser = user.getItemRating(item.getId());

    if (existingRatingByUser != null) {

      estimatedRating = existingRatingByUser.getRating();

    } else {

        for (User anotherUser : dataSet.getUsers()) {

            Rating itemRating = anotherUser.getItemRating(itemId);

            // Учитываем только тех пользователей,
            // которые оценивали эту книгу
            if (itemRating != null) {

               double similarityBetweenUsers =
               similarityMatrix.getValue(userId, anotherUser.getId());

               double ratingByNeighbor = itemRating.getRating();

               double weightedRating =
               similarityBetweenUsers * ratingByNeighbor;

               weightedRatingSum += weightedRating;

               similaritySum += similarityBetweenUsers;
            }
        }

        if (similaritySum > 0.0) {
            estimatedRating = weightedRatingSum / similaritySum;
        }
    }
    return estimatedRating;
}
```

Цикл по всем остальным пользователям

Получение рейтинговой оценки для того же предмета

Вычисление сходства двух пользователей

Масштабирование коэффициента с учетом сходства

Вычисление рейтинговой оценки как отношение прямой и масштабированной сумм

В более интересном случае, когда пользователь еще не оценивал конкретный предмет, мы организуем цикл для перебора всех пользователей и выявляем тех из них, кто выставил оценку конкретному предмету `item`. Каждый из этих пользователей внес в средневзвешенный коэффициент свою долю, прямо пропорциональную его сходству с получающим рекомендацию пользователем `user`. Переменная `similaritySum` была введена в целях нормализации – сумма всех весовых коэффициентов должна быть равна 1.

Как видно из листингов 3.4–3.6, такой способ выработки рекомендаций может оказаться исключительно трудным, если число пользователей

в вашей системе значительно возрастает, как это часто бывает в крупных интернет-магазинах. Есть много возможностей, позволяющих оптимизировать этот программный код. Мы уже упоминали об оптимизации хранения данных, но можно также реализовать еще одно структурное изменение, результатом которого будет как более эффективное использование дискового пространства, так и повышение производительности на этапе выполнения. Вычисляя сходство пользователей, можно запомнить только *N* первых похожих пользователей и получить взвешенную оценку (предсказание) на основании рейтинговых оценок, выставленных только этими *N* пользователями, вместо того чтобы учитывать рейтинги всех пользователей, оценивавших данный предмет; эту версию называют *kNN*, где *NN* означает ближайших соседей, а *k* указывает на то, скольких из них следует учитывать. Выработка рекомендаций на основе сходства пользователей – это надежная методика, но для большого количества пользователей она может оказаться неэффективной; в таком случае предпочтительнее использовать сходство на основе предметов.

3.2.2. Рекомендации на основе сходства предметов

Коллаборативная фильтрация на основе сходства предметов работает почти так же, как CF-фильтрация по сходству пользователей, за исключением того, что сходство между пользователями заменяется сходством между предметами. Давайте определим такую конфигурацию класса `Delphi`, которая позволяет осуществить фильтрацию по сходству между предметами (музыкальные записи), и посмотрим, что из этого выйдет. В листинге 3.9 приведен сценарий, который будет использован для этой цели. Мы загружаем данные, сохраненные в листинге 3.3, и запрашиваем рекомендации для того же пользователя, чтобы сравнить полученные результаты. Кроме того, мы запрашиваем список предметов, похожих на композицию «La Bamba», которая появляется в обоих списках.

Листинг 3.9. Выработка рекомендаций на основе сходства предметов

```
BaseDataset ds = BaseDataset
➥  .load("C:/iWeb2/deploy/data/ch3_2_dataset.ser");   ← Загрузка данных, сохраненных в листинге 3.3

Delphi delphi = new Delphi(ds,RecommendationType.ITEM_BASED);   ← Создание системы выработки рекомендаций на основе сходства предметов
delphi.setVerbose(true);

MusicUser mu1 = ds.pickUser("Bob");
delphi.recommend(mu1);   ← Рекомендация нескольких предметов пользователю Bob

MusicItem mi = ds.pickItem("La Bamba");
delphi.findSimilarItems(mi);   ← Поиск предметов, похожих на "La Bamba"
```

На рис. 3.5 показаны результаты выполнения программного кода из листинга 3.9. Если сравнить эти результаты с результатами, представленными на рис. 3.4, вы увидите, что рекомендации остались прежни-

ми, но изменилась их очередность. Нет никакой гарантии, что рекомендации, полученные на основе сходства пользователей, будут идентичны тем, которые получены на основе сходства предметов. Кроме того, почти наверняка будут отличаться сами оценки. В нашем конкретном примере искусственно сгенерированных данных интересно то, что порядок следования рекомендаций изменился на обратный.

```
bsh % MusicUser mu1 = ds.pickUser("Bob");
bsh % delphi.recommend(mu1);

Recommendations for user Bob:

    Item: Sunday, Bloody Sunday  , predicted rating: 4.483900
    Item: La Bamba               , predicted rating: 4.396600
    Item: I Love Rock And Roll   , predicted rating: 4.000000
    Item: Wind Beneath My Wings  , predicted rating: 4.000000

bsh % MusicItem mi = ds.pickItem("La Bamba");
bsh % delphi.findSimilarItems(mi);

Items like item La Bamba:

    name: Yesterday                    , similarity: 0.615385
name: Fiddler On The Roof              , similarity: 0.588235
name: Vivaldi: Four Seasons            , similarity: 0.555556
name: Singing In The Rain              , similarity: 0.529412
name: You've Lost That Lovin' Feelin'  , similarity: 0.529412
```

Рис. 3.5. *Обнаружение сходных предметов и предоставление рекомендаций с помощью Delphi на основе сходства предметов*

Этот результат не является стандартным, так уж вышло в данном случае. В других случаях, в частности в реальных наборах данных, результаты могут быть упорядочены любым другим образом; выполните этот сценарий несколько раз, чтобы посмотреть, как варьируются результаты всякий раз, когда вы генерируете другой набор данных.

Программный код для выработки рекомендаций на основе сходства предметов остается почти таким же, за исключением, конечно, того, что мы используем предметы вместо пользователей. Вычисление осуществляется в методе `calculate` класса `ItemBasedSimilarity`.

Листинг 3.10. Вычисление сходства на основе предметов

```
protected void calculate(Dataset dataSet) {

  int nItems = dataSet.getItemCount();   ← Определение размера матрицы сходства
```

```
  int nRatingValues = 5;   ← Определение размера матрицы счетчиков рейтинговых оценок

  similarityValues = new double[nItems][nItems];

  if( keepRatingCountMatrix ) {
      ratingCountMatrix = new RatingCountMatrix[nItems][nItems];
  }

  // Если вместо идентификатора userId используется индекс, генерируем
значение индекса для каждого userId
  if( useObjIdToIndexMapping ) {
     for(Item item : dataSet.getItems() ) {
         idMapping.getIndex(String.valueOf(item.getId()));
     }
  }

  for (int u = 0; u < nItems; u++) {
    int itemAId = getObjIdFromIndex(u);
    Item itemA = dataSet.getItem(itemAId);

    // Нам достаточно вычислить только элементы,
    // расположенные выше главной диагонали
    for (int v = u + 1; v < nItems; v++) {  ❶

      int itemBId = getObjIdFromIndex(v);
      Item itemB = dataSet.getItem(itemBId);
                                                  Совпадение рейтинговых
      RatingCountMatrix rcm =                     оценок для двух предметов
➥      new RatingCountMatrix(itemA, itemB, nRatingValues);

      int totalCount     = rcm.getTotalCount();
      int agreementCount = rcm.getAgreementCount();

      if (agreementCount > 0) {        ← Вычисление сходства или
                                         присваивание нулевого значения
        similarityValues[u][v] =
➥        (double) agreementCount / (double) totalCount;

      } else {

        similarityValues[u][v] = 0.0;
      }

      if( keepRatingCountMatrix ) {
        ratingCountMatrix[u][v] = rcm;
      }
    }

    // Для u == v присваиваем 1
    similarityValues[u][u] = 1.0;  ❶
  }
}
```

Оптимизация программного кода ❶ совпадает с той, которую мы видели при получении оценки на основе сходства пользователей в листинге 3.4.

Класс RatingCountMatrix снова используется для того, чтобы отслеживать совпадения и несовпадения в рейтинговых оценках, хотя теперь нас интересует совпадение и несовпадение оценок, данных двум разным предметам, а не оценок, выставленных двумя разными пользователями. В этом программном коде осуществляется последовательный перебор всех возможных пар предметов и назначение значений сходства на основе метрики Жаккарда. Программный код в классе Delphi, обеспечивающий выработку рекомендаций на основе сходства предметов, в точности следует соответствующему коду для выработки рекомендаций на основе сходства пользователей. В листинге 3.11 мы демонстрируем оценку сходства для выработки рекомендаций на основе сходства предметов; сравните этот программный код с кодом из листинга 3.8. Для оценки сходства любого типа используется программный код, идентичный коду из листингов 3.6 и 3.7.

Листинг 3.11. Класс Delphi: выработка рекомендаций на основе сходства предметов

```
private double estimateItemBasedRating(User user, Item item) {

  double estimatedRating = Double.NaN;

  int itemId = item.getId();
  int userId = user.getId();

  double similaritySum = 0.0;
  double weightedRatingSum = 0.0;

  // Проверка: оценивал ли пользователь этот предмет ранее
  Rating existingRatingByUser = user.getItemRating(item.getId());

  if (existingRatingByUser != null) {

    estimatedRating = existingRatingByUser.getRating();

  } else {

    double similarityBetweenItems = 0;
    double weightedRating = 0;

    for (Item anotherItem : dataSet.getItems()) {

      // Учитываем только те предметы,
      // которые были оценены этим пользователем
      Rating anotherItemRating = anotherItem.getUserRating(userId);

      if (anotherItemRating != null) {
```

Цикл по всем остальным предметам

Получение оценки для этого же пользователя

```
        similarityBetweenItems =
➥   similarityMatrix.getValue(itemId, anotherItem.getId());
                                                  Получение сходства двух предметов
        if (similarityBetweenItems > similarityThreshold) {
                                                  Масштабирование рейтинговой
          weightedRating =                        оценки с учетом сходства
➥   similarityBetweenItems * anotherItemRating.getRating();
          weightedRatingSum += weightedRating;

          similaritySum += similarityBetweenItems;
        }
      }
    }

    if (similaritySum > 0.0) {          Вычисление рейтинговой оценки как отношение
                                        прямой суммы к взвешенной сумме
      estimatedRating = weightedRatingSum / similaritySum;
    }
  }
  return estimatedRating;
}
```

Этими листингами завершается наше первоначальное рассмотрение коллаборативной фильтрации (CF), или выработки рекомендаций на основе сходства пользователей или предметов. Как правило, предпочтение отдается CF на основе сходства предметов из-за большого количества покупателей (миллионы или даже десятки миллионов), но иногда, стремясь к получению более точных рекомендаций, используют комбинации этих двух методов коллаборативной фильтрации. В последующих разделах мы приведем примеры настройки такого сайта, как Amazon.com (*http://www.amazon.com*), где применяется подход с объединением «предмет–предмет», а также пример предоставления рекомендаций на таком сайте, как Netflix.com (*http://www.netflix.com*), который продемонстрирует комбинацию этих двух методов.

3.2.3. Рекомендации на основе контента

Выработка рекомендаций на основе контента зависит от сходства контента разных пользователей, разных предметов или разных пользователей и разных предметов. Вместо рейтинговых оценок у нас теперь есть показатель того, насколько «близки» друг к другу два документа. Понятие расстояния между документами – это обобщение оценки релевантности документа запросу, которую мы обсудили в главе 2. Всегда можно рассматривать один документ как запрос, а другой – как ссылку. Разумеется, вы должны сравнивать только значимые части каждого документа, иначе информация, которую несет в себе каждый документ, может быть потеряна из-за путаницы.

Подготовка к анализу конкретного случая

В качестве источников контента воспользуемся документами из главы 2 и назначим номера этих веб-страниц каждому пользователю, прибегнув к способу, похожему на тот, который использовался для назначения пользователям музыкальных записей в рассмотренном ранее примере. Для каждого пользователя случайным образом выберем набор страниц, которые на 80% совпадают со всеми подходящими страницами из нашей коллекции. Подходящие документы подбираются для каждого пользователя со строгим смещением, следующим образом:

- Если имя пользователя начинается с буквы от a до d (включительно), мы назначаем пользователю документы, 80% которых принадлежат одной из двух категорий – business или sports
- В остальных случаях мы назначаем пользователю документы, 80% которых принадлежат категории usa или world

Итак, мы создаем две большие группы пользователей со сходными (хотя, отчасти искусственными) предпочтениями, которые позволят быстро оценить результаты. Рассмотрим этапы, из которых складывается создание объектов нашего рекомендателя `Delphi`, работающего на основе сходства контента. В листинге 3.12 приведен программный код, обеспечивающий подготовку данных и последующее выявление похожих пользователей и похожих предметов. Кроме того, мы обеспечиваем выработку рекомендаций по предметам на основе гибридного сходства контента «пользователь–предмет».

Листинг 3.12. Выработка рекомендаций на основе сходства контента

```
BaseDataset ds = NewsData.createDataset();

Delphi delphiUC = new Delphi(ds,RecommendationType.USER_CONTENT_BASED);
delphiUC.setVerbose(true);
                                  Создание системы на основе контента пользователей

NewsUser nu1 = ds.pickUser("Bob");
delphiUC.findSimilarUsers(nu1);

NewsUser nu2 = ds.pickUser("John");
delphiUC.findSimilarUsers(nu2);

Delphi delphiIC = new Delphi(ds,RecommendationType.ITEM_CONTENT_BASED);
                                  Создание системы на основе контента предметов
delphiIC.setVerbose(true);

ContentItem i = ds.pickContentItem("biz-05.html");
delphiIC.findSimilarItems(i);

Delphi delphiUIC =
    new Delphi(ds,RecommendationType.USER_ITEM_CONTENT_BASED);
delphiUIC.setVerbose(true);
                                  Создание системы на основе контента «пользователь-предмет»

delphiUIC.recommend(nu1);
```

В первой строке этого сценария описанным ранее способом создается набор данных. Получив набор данных, мы создаем экземпляр объекта `Delphi` на основе матрицы сходства «пользователь–пользователь», которая вычисляется в классе `UserContentBasedSimilarity`. Поскольку у каждого пользователя имеется больше одного документа, мы должны сравнить каждый документ каждого пользователя с каждым документом каждого другого пользователя. Есть несколько способов, позволяющих провести такое сравнение. В нашем программном коде, как показано в листинге 3.13, для каждой возможной пары пользователей – пользователь *A* и пользователь *B* – в цикле осуществляется перебор всех документов пользователя *A*, для каждого из которых ищется документ пользователя *B*, обладающий наибольшим сходством с текущим документом пользователя *A*. Затем мы усредняем наилучшие показатели сходства для каждого документа пользователя *A* и назначаем полученное среднее значение в качестве меры сходства между пользователями *A* и *B*.

Листинг 3.13. Вычисление сходства пользователей на основе их контента

```
protected void calculate(Dataset dataSet) {

  int nUsers = dataSet.getUserCount();

  similarityValues = new double[nUsers][nUsers];

  // Если вместо идентификатора userId используется индекс,
  // генерируем значение индекса для каждого userId
  if( useObjIdToIndexMapping ) {
    for(User u : dataSet.getUsers() ) {
        idMapping.getIndex(String.valueOf(u.getId()));
    }
  }

  CosineSimilarityMeasure cosineMeasure =
➥   new CosineSimilarityMeasure();   ← Создание косинусной меры сходства

  for (int u = 0; u < nUsers; u++ ) {

    int userAId = getObjIdFromIndex(u);
    User userA = dataSet.getUser(userAId);

    for (int v = u + 1; v < nUsers; v++) {  ❶

      int userBId = getObjIdFromIndex(v);
      User userB = dataSet.getUser(userBId);

      double similarity = 0.0;
                                  Последовательный перебор всех имеющих
                                  оценку предметов пользователя A
      for(Content userAContent : userA.getUserContent() ) {

         double bestCosineSimValue = 0.0;
```

```
        for(Content userBContent : userB.getUserContent() ) {
                                  Последовательный перебор всех имеющих оценку предметов пользователя B
          double cosineSimValue = cosineMeasure
          .calculate(userAContent.getTFMap(), userBContent.getTFMap());

          bestCosineSimValue =
          Math.max(bestCosineSimValue, cosineSimValue);
        }

        similarity += bestCosineSimValue;
      }                                   Накопление показателей сходства с самым
                                          высоким значением из числа показателей,
                                          полученных для всех документов
      similarityValues[u][v] = similarity /
      userA.getUserContent().size();      Вычисление сходства как простое среднее
      }

    // Для u == v назначаем 1.
    similarityValues[u][u] = 1.0;  ❶
  }
}
```

Эта оптимизация ❶ программного кода совпадает с той, которую мы видели, когда оценивали сходство на основе пользователей в листинге 3.4.

Ключевые понятия, позволяющие определить сходство на основе контента

Ключевой элемент всех методов, основанных на сходстве контента, – представление текстовой информации в виде числовой величины. Легким способом, позволяющим добиться такого представления, является идентификация в каждом документе N наиболее часто встречающихся термов и использование множества термов, наиболее часто встречающихся во всех документах, в качестве пространства координат. Мы можем воспользоваться классом `StandardAnalyzer` из библиотеки Lucene, чтобы удалить стоп-слова и вырезать корневую основу терма, повышая тем самым значимость важных термов и при этом существенно уменьшая шум. С этой целью мы создали класс `CustomAnalyzer`, расширяющий возможности класса `StandardAnalyzer` и позволяющий удалить некоторые слова, которые являются общими и добавляют, если они есть, ощутимый уровень шума в наши векторы.

Давайте отвлечемся на некоторое время, чтобы конкретизировать эти важные понятия. Пусть, скажем, для удобства обсуждения, $N = 4$, у вас имеются три документа и следующие (с высокой частотой их появления в документах) термы:

- D1 = {Google, shares, advertisement, president}
- D2 = {Google, advertisement, stock, expansion}
- D3 = {NVidia, stock, semiconductor, graphics}

Каждый из рассматриваемых документов можно представить математически с помощью девятимерного вектора, который отражает, содержится ли в конкретном документе один из девяти уникальных термов – {Google, shares, advertisement, president, stock, expansion, Nvidia, semiconductor, graphics}. Таким образом, наши три документа были бы представлены следующими тремя векторами:

- D1 = {1, 1, 1, 1, 0, 0, 0, 0, 0}
- D2 = {1, 0, 1, 0, 1, 1, 0, 0, 0}
- D3 = {0, 0, 0, 0, 1, 0, 1, 1, 1}

Вуаля! Мы создали три чисто математические величины, которые можно использовать для количественного сравнения документов. Мы собираемся работать с показателем сходства, который называется *косинусной мерой сходства* (cosine similarity). Мы уже видели несколько формул, позволяющих оценить сходство, и эта мера не сильно от них отличается. Вместо того чтобы приставать к вам с математической формулой, опишем класс, который инкапсулирует ее определение. В листинге 3.14 показан программный код класса `CosineSimilarityMeasure`.

Листинг 3.14. Вычисление косинусной меры сходства векторов термов

```
public class CosineSimilarityMeasure {
     public double calculate(double[] v1, double[] v2) {

        double a = getDotProduct(v1, v2);

        double b = getNorm(v1) * getNorm(v2);

        return a / b;
    }

    private double getDotProduct(double[] v1, double[] v2) {

        double sum = 0.0;

        for(int i = 0, n = v1.length; i < n; i++) {
            sum += v1[i] * v2[i];
        }
        return sum;
    }

    private double getNorm(double[] v) {

        double sum = 0.0;

        for( int i = 0, n = v.length; i < n; i++) {
            sum += v[i] * v[i];
        }
        return Math.sqrt(sum);
    }
}
```

Нормализация двух векторов и вычисление произведения

Получение косинусной меры сходства

Вычисление скалярного произведения

Вычисление евклидовой нормы вектора

Как видите, сначала мы формируем то, что называется *скалярным (внутренним) произведением* (dot (inner) product) двух векторов – это переменная `a` типа `double`. Затем вычисляем норму (модуль) каждого вектора и запоминаем их произведение в переменной `b` типа `double`. Косинусная мера сходства – это просто отношение `a / b`. Если мы обозначим косинусную меру сходства документа X и документа Y как CosSim(X, Y), в случае нашего простого примера имеются следующие показатели сходства:

- $CosSim(D1, D2) = 2 / (2 \times 2) = 0.5$
- $CosSim(D1, D3) = 0 / (2 \times 2) = 0$
- $CosSim(D2, D3) = 1 / (2 \times 2) = 0.25$

Методика представления документов на основе содержащихся в них термов – фундамент теории информационного поиска. Мы должны обратить внимание на то, что идентификация термов – это ключевой этап, и получить ее непосредственно для базы документов общего характера нелегко. Модифицируем, например, наш программный код так, чтобы вместо написанного нами класса `CustomAnalyzer` использовался класс `StandardAnalyzer`. Что вы наблюдаете? Результаты могут существенно измениться, даже несмотря на то, что в нашем собственном пользовательском классе, на первый взгляд, не слишком много нового по сравнению с базовым классом. Этот небольшой эксперимент должен убедить вас в том, что подход, основанный на сходстве контента, очень чувствителен к стадии лексического анализа.

Три типа рекомендаций, вырабатываемых на основе контента

Возвращаясь к нашему примеру, рассмотрим полученные результаты. На рис. 3.6 показана часть результатов, полученных при выполнении программного кода из листинга 3.12, отвечающего за нахождение похожих пользователей.

Этот алгоритм отработал успешно, поскольку две отдельные группы пользователей корректно идентифицированы как обладающие сходством – пользователи, чьи имена начинаются с букв от A до D, и пользователи, чьи имена начинаются с букв от E до Z. Обратите внимание на небольшую разницу в значениях показателя сходства. Подход, основанный на сходстве контента, по-видимому, не приводит к хорошему разграничению пользователей при сравнении их друг с другом. На рис. 3.7 показано выполнение программного кода, который отвечает за нахождение похожих предметов. Как видите, было идентифицировано несколько релевантных предметов, но также были идентифицированы несколько предметов, у которых пользователь-человек не обнаружил бы большого сходства.

```
bsh % BaseDataset ds = NewsData.createDataset();
bsh % Delphi delphiUC =
new Delphi(ds,RecommendationType.USER_CONTENT_BASED);

bsh % delphiUC.setVerbose(true);
bsh % NewsUser nu1 = ds.pickUser("Bob");
bsh % delphiUC.findSimilarUsers(nu1);

Top Friends for user Bob:

    name: Albert        , similarity: 0.950000
    name: Catherine     , similarity: 0.937500
    name: Carl          , similarity: 0.937500
    name: Alexandra     , similarity: 0.925000
    name: Constantine   , similarity: 0.925000

bsh % NewsUser nu2 = ds.pickUser("John");
bsh % delphiUC.findSimilarUsers(nu2);

Top Friends for user John:

    name: George        , similarity: 0.928571
    name: Lukas         , similarity: 0.914286
    name: Eric          , similarity: 0.900000
    name: Nick          , similarity: 0.900000
    name: Frank         , similarity: 0.900000
```

Рис. 3.6. *Имена пользователей, обладающих сходством с пользователем Bob, начинаются с букв от A до D. Алгоритм позволил успешно идентифицировать две группы пользователей, обладающих сходством!*

И снова видим, что значения показателей сходства варьируются не сильно; применение этого алгоритма вряд ли позволило бы предоставить превосходные рекомендации. Возможность устранить неоднозначность отсутствует по причине недостаточности нашего лексического анализа. *Обработка естественных языков* (natural language processing, NLP) – это богатая возможностями и трудная область. Тем не менее за последние два десятка лет в ней достигнут большой прогресс; и хотя в настоящей книге мы не углубляемся в эту увлекательную тему, различные компоненты систем NLP будут кратко перечислены в приложении D.

На рис. 3.8 мы привели рекомендации, полученные на основе сходства «пользователь–предмет». Хотя обычно методика коллаборативной фильтрации имеет дело со сходством «пользователь–пользователь» или «предмет–предмет», для выработки рекомендаций на основе сходства

```
bsh % Delphi delphiIC =
new Delphi(ds,RecommendationType.ITEM_CONTENT_BASED);

bsh % delphiIC.setVerbose(true);
bsh % ContentItem biz1 = ds.pickContentItem("biz-01.html");
bsh % delphiIC.findSimilarItems(biz1);

Items like item biz-01.html:

    name: biz-03.html    , similarity: 0.600000
    name: biz-02.html    , similarity: 0.600000
    name: biz-04.html    , similarity: 0.100000
    name: biz-07.html    , similarity: 0.100000

bsh % ContentItem usa1 = ds.pickContentItem("usa-01.html");
bsh % delphiIC.findSimilarItems(usa1);

Items like item usa-01.html:

    name: usa-02.html    , similarity: 0.300000
    name: usa-03.html    , similarity: 0.300000
    name: world-03.html  , similarity: 0.100000
    name: world-05.html  , similarity: 0.100000
    name: usa-04.html    , similarity: 0.100000

bsh % ContentItem sport1 = ds.pickContentItem("sport-01.html");
bsh % delphiIC.findSimilarItems(sport1);

Items like item sport-01.html:

    name: sport-03.html  , similarity: 0.400000
    name: sport-02.html  , similarity: 0.300000
```

Рис. 3.7. *Предметы, принадлежащие к той же категории, что и запрашиваемые, безошибочно определены как похожие*

«пользователь–предмет» предпочтительнее подход на основе контента. Впрочем, проблемы лексического анализа при этом не исчезают, и без кропотливой и специфической работы с применением методов NLP получить удовлетворительные результаты не удастся. Если увеличить объем набора данных и выполнить рассматриваемый сценарий несколько раз для разных пользователей, очень многие рекомендации будут иметь идентичные рейтинговые оценки, и предсказанные оценки не будут сильно разниться.

```
bsh % Delphi delphiUIC = new Delphi(
➥    ds,RecommendationType.USER_ITEM_CONTENT_BASED);
bsh % delphiUIC.setVerbose(true);
bsh % delphiUIC.recommend(nu1);

Recommendations for user Bob:

    Item: biz-06.html      , predicted rating: 2.500000
    Item: biz-04.html      , predicted rating: 1.500000
    Item: usa-02.html      , predicted rating: 0.500000
    Item: world-03.html    , predicted rating: 0.500000
    Item: world-05.html    , predicted rating: 0.500000
```

Рис. 3.8. *Мы получили рекомендации предметов на основе сходства контента, связанного с пользователем Bob*

Итак, в основе построения систем выработки рекомендаций лежит сходство «пользователь–пользователь», «предмет–предмет», а также сходство контента. Выработка рекомендаций на основе сходства пользователей является надежной методикой, но может оказаться неэффективной при большом числе пользователей.

В последнем случае предпочтительнее воспользоваться коллаборативной фильтрацией по сходству предметов, потому что число пользователей (миллионы или даже десятки миллионов) на порядки превышает число предметов. Подход, основанный на сходстве контента, не находит широкого применения, но у него есть определенные преимущества и его можно сочетать с коллаборативной фильтрацией, чтобы повысить качество рекомендаций. Обычно в промышленных системах применяется комбинация этих двух методик. Давайте рассмотрим концепцию объединения систем выработки рекомендаций.

3.3. Выработка рекомендаций по друзьям, статьям и новостным сообщениям

В этом разделе мы представляем более реалистичный пример, который поможет проиллюстрировать объединение рассмотренных методик. Мы будем работать с гипотетическим веб-сайтом, предназначенным для идентификации индивидуумов с похожими мнениями, статей со сходными комментариями, а также новостных сообщений с похожим контентом. Давайте назовем наш веб-сайт MyDiggSpace.com. Как подсказывает его имя, этот сайт будет пользоваться интерфейсом Digg API для извлечения статей, которые вы предлагаете под своей учетной записью интерфейса Digg (информацию об этой записи можно было бы предоставлять при регистрации). Затем этот сайт идентифицирует

и предоставит вам новостные сообщения, похожие на те, которые «откопали» вы. Кроме того, он позволит оценить прочитанные новости, так что в будущем система сможет уточнить свой выбор рекомендуемых новостей на основе полученного от вас обратного отклика. А еще (как будто перечисленного мало) сайт покажет вам группы по интересам, к которым вы можете присоединиться, если захотите, содействуя, таким образом, взаимному общению лиц с похожим складом ума.

3.3.1. Знакомство с сайтом MyDiggSpace.com

Рассмотрим поочередно этапы, из которых складывается построение такого сайта. Как и обещали во введении, мы не будем рассматривать такие вопросы, как проектирование пользовательского интерфейса UI, персистентность и другие важные составляющие проектирования. Чтобы все это было интересно, мы применим для извлечения данных интерфейс Digg API, сделав наш пример более реалистичным. Во-первых, надо объяснить, что Digg – это такой интернет-ресурс (*http://digg.com/*), на котором пользователи совместно используют контент, обнаруженный ими где-то в Интернете. Идея заключается в том, что контент накапливается не редакторами, которые лучше знают (или не знают), что вам надо, но поступает от самих пользователей. Каким бы ни был источник предмета, который вы хотите обсудить, – будь это выпуск широко обсуждаемых коммерческих новостей или никому не известный блог, сайт Digg позволит вам опубликовать ваши находки и обеспечит известность лучшему контенту через голоса участвующих пользователей.

Интерфейс Digg API позволяет сторонним лицам программно взаимодействовать с сайтом Digg. Большая часть данных, имеющихся на веб-сайте Digg, доступна посредством этого API. Вы можете получить списки новостей, упорядоченные по популярности, по времени или по категориям (обсуждаемые темы). Мы написали набор классов-оболочек, использующих интерфейс Digg API, и позже вы можете его расширить в своих целях.

Мы сформируем набор данных для сайта MyDiggSpace.com, выполнив несколько простых шагов. Прежде всего, подберем новости, возглавляющие список новостей в каждой категории, определенной на сайте Digg. В результате будут созданы список пользователей и списки новостей (предметов) для каждого пользователя.

Для каждой новости каждого пользователя мы найдем 10 похожих новостей, предоставленных другими пользователями, определяя сходство по контенту. Другими словами, мы создадим систему выработки рекомендаций по сходству «предмет–предмет» на основе контента и найдем первые 10 похожих новостей.

Чтобы наш набор данных был полным, мы предполагаем, что пользователи оценивают новости, и, следовательно, назначаем каждой новости

случайную рейтинговую оценку. Оценки назначаются в соответствии с тем же соглашением, которое мы использовали в предыдущих примерах: пользователи, чьи имена начинаются с букв от A до D, назначают оценки 4 или 5, остальные назначают оценки 1, 2 или 3.

Цель этого примера – познакомить вас с концепцией объединения результатов работы различных систем выработки рекомендаций для получения более точных результатов, чем те, которые могла бы предоставить вам каждая из этих систем в отдельности. Такая практика представляется разумной в приложениях, диапазон которых много шире, чем системы выработки рекомендаций. Позже в этой книге мы поговорим об объединении результатов работы систем классификации. Предлагаемый к рассмотрению пример обеспечивает подступы к широкому и перспективному полю деятельности. Он также содержит более важное послание, которое мы хотим передать вам в этой книге, – о значении синергизма различных составляющих интеллекта для получения результатов высокого качества в реальных приложениях.

3.3.2. Нахождение друзей

Выполним наш сценарий, приведенный в листинге 3.15, и приступим к взаимодействию с гипотетическими данными сайта MyDiggSpace.com[1].

Листинг 3.15. Сайт MyDiggSpace.com: пример объединения систем выработки рекомендаций

```
BaseDataset ds = DiggData
➥   .loadDataFromDigg("C:/iWeb2/data/ch03/digg_stories.csv");   ← Сохранение данных с сайта Digg

// BaseDataset ds = DiggData
[CA].loadData("C:/iWeb2/data/ch03/digg_stories.csv");   ← Или загрузка локальных данных

iweb2.ch3.collaborative.model.User user = ds.getUser(1);   ← Отбор пользователя

DiggDelphi delphi = new DiggDelphi(ds);   ← Создание экземпляра рекомендателя

delphi.findSimilarUsers(user);   ← Нахождение похожих пользователей

delphi.recommend(user);   ← Рекомендация по новостям
```

Пользователей, обладающих сходством с выбранным пользователем, можно было бы отобразить в боковой панели, например, как рецензентов новостных сообщений данного пользователя. Рекомендованные но-

[1] Оговорка: данные, которые позволяет собрать этот сценарий, являются общедоступными. Мы, разумеется, не отвечаем за контент, который вы можете извлечь, выполняя наш пример. Наша задача – предоставить работающий пример применения интерфейса Digg API и продемонстрировать, как с его помощью сделать что-то полезное.

вости также можно было бы показать в специальной панели, а чтобы улучшить качество рекомендаций, предлагаемых каждому пользователю, мы могли бы применить подход, использующий переходы по ссылкам и аналогичный тому, который был рассмотрен в главе 2. Кроме того, можно было бы предоставить пользователю возможность оценить каждую рекомендованную новость, чтобы с еще большей уверенностью судить о его предпочтениях. Мы сейчас обсудим эти улучшения, но сначала давайте посмотрим на результаты, которые обеспечил наш сценарий, пока мы писали эту книгу.

Для 33 пользователей мы отобрали 146 предметов (новостей), принадлежащих 7 категориям; число категорий и их контент можно регулировать в классе `iweb2.ch3.content.digg.DiggCategory`. Отобранным пользователям мы назначили 811 рейтинговых оценок предметов. Выбор предметов и рейтинговых оценок для каждого пользователя случаен, за исключением того, что мы следуем уже применявшемуся соглашению о кластеризации оценок по инициалам пользователей. Минимальное количество рейтинговых оценок, выставленных пользователем для новостей из этого набора, равно 7, максимальное – 31, а среднее значение выборки равно 26.

Эффект треугольника

На рис. 3.9 представлено множество пользователей из нашего списка, обладающих сходством с первым пользователем (adamfishercox); затем перечисляются пользователи, имеющие сходство с пользователем (adrian67), который больше всех похож на первого пользователя; далее следуют пользователи, похожие на пользователя с именем DetroitThang1, у которого есть сходство (хотя и не самое большое) с пользователем adrian67. Данные на рис. 3.9 позволяют сделать интересное наблюдение, которое может быть, а может и не быть очевидным. Пользователь amipress попадает в первую пятерку пользователей, похожих на пользователя adamfishercox, но не входит в первую пятерку пользователей, обладающих сходством с пользователем adrian67. Тем не менее пользователь amipress находится в первой пятерке пользователей, похожих на пользователя DetroitThang1, а сходство между ними оценивается как 0,7, что почти совпадает с оценкой сходства, которое мы обнаружили между пользователями amipress и adamfishercox. Интересно, да? Мы называем это явление *эффектом треугольника*, который демонстрирует наличие *результатов второго порядка*, которые можно задействовать, чтобы повысить точность – и, следовательно, эффективность – вырабатываемых рекомендаций.

Поясним эту мысль с помощью рис. 3.9. Ранг пользователя adamfishercox относительно adrian67 равен 1, и оценка их сходства равна 1; ранг пользователя amipress относительно adamfishercox – 2, и их сходство оценивается (приблизительно) как 0,67.

```
bsh % delphi.findSimilarUsers(user);
Top Friends for user adamfishercox:

    name: adrian67       , similarity: 1.000000
    name: amipress       , similarity: 0.666667
    name: dvallone       , similarity: 0.500000
    name: cosmikdebris   , similarity: 0.500000
    name: cruelsommer    , similarity: 0.500000

bsh % iweb2.ch3.collaborative.model.User u2 =
ds.findUserByName("adrian67");

bsh % delphi.findSimilarUsers(u2);

Top Friends for user adrian67:

    name: adamfishercox  , similarity: 1.000000
    name: dvallone       , similarity: 1.000000
    name: ambermacbook   , similarity: 1.000000
    name: DetroitThang1  , similarity: 0.800000
    name: cruelsommer    , similarity: 0.750000

bsh % iweb2.ch3.collaborative.model.User u3 =
ds.findUserByName("DetroitThang1");

bsh % delphi.findSimilarUsers(u3);

Top Friends for user DetroitThang1:

    name: adrian67       , similarity: 0.800000
    name: cosmikdebris   , similarity: 0.750000
    name: amipress       , similarity: 0.700000
```

Рис. 3.9. *Нахождение похожих пользователей и эффект треугольника при использовании случайного набора данных Digg*

Ранг пользователя amipress по отношению к пользователю adrian67 – 7, а оценка их сходства равна 0,57. Мы отобразили эти взаимосвязи графически на рис. 3.10, где adamfishercox – это Пользователь 1, amipress – Пользователь 2, а adrian67 – Пользователь 3.

Число в круглых скобках – это относительная оценка ранжирования, а число в квадратных скобках – это наша оценка сходства; основание стрелки указывает на пользователя, для которого мы ищем похожих на него пользователей. Линия со стрелкой, связывающая Пользователя 3 и Пользователя 2, нарисована пунктиром, чтобы графически обозначить взаимосвязь, которую мы можем улучшить на основе информации других взаимосвязей (стрелки со сплошными линиями).

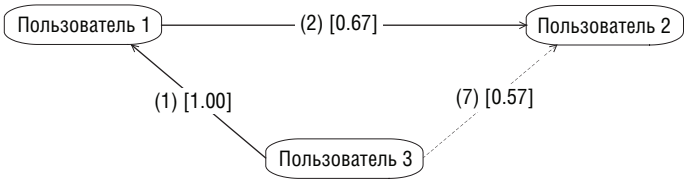

Рис. 3.10. *Эффект треугольника и возможность улучшить относительное ранжирование*

3.3.3. Внутренние механизмы класса DiggDelphi

Теперь рассмотрим программный код, который позволил выработать эти рекомендации. В листинге 3.16 представлен код из класса `DiggDelphi`.

Листинг 3.16. Объединение систем выработки рекомендаций для сайта MyDiggSpace.com

```
public class DiggDelphi {

  private Dataset ds;

  private Delphi delphiUC;
  private Delphi delphiUIC;
  private Delphi delphiUR;
  private Delphi delphiIR;

  private boolean verbose = true;

  public DiggDelphi(Dataset ds) {
      this.ds = ds;

      delphiUC  =
➥      new Delphi(ds,RecommendationType.USER_CONTENT_BASED);

      delphiUIC =
➥      new Delphi(ds,RecommendationType.USER_ITEM_CONTENT_BASED);

      delphiUR  = new Delphi(ds,RecommendationType.USER_BASED);

      delphiIR  = new Delphi(ds,RecommendationType.ITEM_BASED);
  }

public SimilarUser[] findSimilarUsers(User user, int topN) {  ❶

        List<SimilarUser> similarUsers =
➥        new ArrayList<SimilarUser>();

        similarUsers.addAll(
➥        Arrays.asList(delphiUC.findSimilarUsers(user, topN)));

        similarUsers.addAll(
➥        Arrays.asList(delphiUR.findSimilarUsers(user, topN)));
```

Инициализация различных систем выработки рекомендаций

```
        return SimilarUser.getTopNFriends(similarUsers, topN);
    }

    public List<PredictedItemRating> recommend(User user, int topN) {  ❷

        List<PredictedItemRating> recommendations =
➥        new ArrayList<PredictedItemRating>();

        recommendations.addAll(delphiUIC.recommend(user, topN));
        recommendations.addAll(delphiUR.recommend(user, topN));
        recommendations.addAll(delphiIR.recommend(user, topN));

        return PredictedItemRating
➥        .getTopNRecommendations(recommendations, topN);
    }
}
```

Мы хотим выявить похожих пользователей на основе сходства пользователей и сходства «пользователь–контент» ❶ и рекомендовать новости, основываясь на сходстве «пользователь–предмет–контент», сходстве пользователей и сходстве предметов ❷.

Как можно видеть, в методе `findSimilarUsers` прменяется простейший подход к объединению списков похожих пользователей: мы добавляем в список все результаты и сортируем записи по оценке их сходства (это делается в методе `getTopNFriends`). Применение подхода, основанного на сходстве контента, обеспечивает экземпляр класса `delphiUC`, а использование сходства «пользователь–пользователь» на основе рейтинговых оценок (коллаборативная фильтрация) реализовано в экземпляре класса `delphiUR`. Обратите внимание: данные о сходстве, которые обеспечивают эти две системы выработки рекомендаций, ни в коей мере не являются нормализованными. Это значит, что результаты будут отчасти смешанными, даже несмотря на то, что мы их упорядочили.

Чтобы лучше понять этот момент, представьте себе список из 20 банковских счетов. Если 10 из этих счетов в долларах США, а другие 10 в евро, сортировка списка, содержащего и те и другие счета, по их общей сумме лишена точного смысла, если только мы не выразим все эти счета в долларах США или в евро. Тем не менее счета, на которых мало денег, все-таки оказались бы в конце списка, тогда как счета с большими суммами оказались бы вверху списка; просто упорядочивание было бы неточным.

Наша аналогия с валютами, хотя и проливает свет, чрезмерно упрощает главное отличие этих двух случаев. Нормализация валют вполне понятна, и логика ее проста. Если бы мне надо было конвертировать 100 долларов США в 100 евро, я использовал бы обменный курс для этих двух валют, чтобы получить номинальную стоимость 100 долларов США в евро. В реальности, если вы хотите получить на руки (или на ваш банковский счет) евро, придется уплатить банку комиссионный сбор, но ваша формула нормализации по-прежнему остается исклю-

чительно простой. К сожалению, сходство пользователей и оценка рекомендаций с трудом поддаются нормализации. Объединение оценок, предлагаемых системами выработки рекомендаций, это и искусство, и наука. Здесь часто применяются сложные эвристики, и алгоритмы машинного обучения играют важную роль в создании поверх первичных рекомендаций еще одного слоя обработки информации.

На рис. 3.11 показаны результаты простого объединения рекомендаций, полученных для трех пользователей при использовании трех разных подходов, которые мы уже исследовали.

```
bsh % delphi.recommend(user);

Recommendations for user adamfishercox:

Item: Lumeneo Smera: French Concept of Car and MotorCycle, predicted
rating: 5.0
Item: Bill Gates to Congress: Let us hire more foreigners - CNET N,
predicted rating: 5.0
Item: The Best Tools for Visualization, predicted rating: 5.0
Item: Coolest Cubicle Contest, Part Three, predicted rating: 5.0
Item: Bush: Telecoms Should Be Thanked For Their Patriotic Service,
predicted rating: 5.0

bsh % delphi.recommend(u2);

Recommendations for user adrian67:

Item: Can women parallel park on Mars?, predicted rating: 5.0
Item: Coast Guard loses a few flares and ..., predicted rating: 5.0
Item: 10.5.2 released, predicted rating: 5.0
Item: They are all hot!, predicted rating: 5.0
Item: 11 Greatest Basketball Commercials Ever Made, predicted rating: 5.0

bsh % delphi.recommend(u3);

Recommendations for user DetroitThang1:

Item: The Best Tools for Visualization, predicted rating: 5.0
Item: Coolest Cubicle Contest, Part Three, predicted rating:
5.000000
Item: Stink Films comes correct with 3 Adidas Original Films, predicted
rating: 5.0
Item: The Power Rangers Meet The Teenage Mutant Ninja Turtles, predicted
rating: 5.0
```

Рис. 3.11. *Пример результатов объединения трех разных систем выработки рекомендаций*

Как показано в методе `recommend` из листинга 3.16, мы создаем список рекомендаций, являющихся результатом работы рекомендателя, работа которого основана на сходстве контента «пользователь–предмет», рекомендателя, использующего коллаборативную фильтрацию «пользователь–пользователь», а также рекомендателя, использующего коллаборативную фильтрацию «предмет–предмет».

Это хорошие результаты, в том смысле что все рекомендованные рейтинговые оценки равны 5, что и следовало ожидать по причине введенного нами искусственного отклонения в ранжировании: пользователи, чьи имена начинаются с букв от A до D, всегда выставляют оценку 5 или 4. Вспомните, мы говорили, что отсутствие нормализации оценок сходства способно обеспечить превосходство одного рекомендателя над другими. Нам нужен механизм, позволяющий рассматривать рекомендации различных систем на равных.

Взгляните на реализацию метода `recommend`, в которой это учтено (листинг 3.17). Первый шаг – нормализовать все предсказанные рейтинговые оценки, приняв за начало отсчета максимальную рейтинговую оценку из всех предсказанных для этого пользователя оценок всеми системами выработки рекомендаций. Мы также вводим произвольное пороговое значение, позволяющее отбросить рекомендации с предсказанными рейтинговыми оценками ниже определенного значения. Для вас это будет знакомством с интересной темой подсчета стоимости плохих рекомендаций. Другими словами, наше пороговое значение (хотя и искуственное) воздвигает барьер для предсказанных рейтинговых оценок: наши рекомендации должны превысить этот барьер, прежде чем будут приняты к серьезному рассмотрению.

Последняя часть этой реализации заключается в усреднении всех предсказанных рейтинговых оценок для конкретного предмета, с тем чтобы получить единственную предсказываемую оценку. Такой подход допустим, потому что мы нормализовали рейтинговые оценки; без нормализации усреднение не имело бы особого смысла. Если конкретная система выработки рекомендаций не оценивает конкретный предмет, то значение соответствующей рейтинговой оценки должно быть равно нулю, и следовательно, конкретный предмет опустился бы ниже в списке рекомендаций. То есть наш подход комбинирует усреднение и голосование за предсказанные рекомендателями рейтинговые оценки. После того как комбинированная оценка вычислена, рекомендации добавляются в список и результаты сортируются на основе новой предсказанной оценки.

Листинг 3.17. Улучшенная реализация процедуры выработки рекомендаций за счет объединения рекомендателей

```
public List<PredictedItemRating> recommend(User user, int topN) {

    List<PredictedItemRating> recommendations =
➥   new ArrayList<PredictedItemRating>();
```

```
  double maxR=-1.0d;

  double maxRatingDelphiUIC =
➥   delphiUIC.getMaxPredictedRating(user.getId());

  double maxRatingDelphiUR  =
➥   delphiUR.getMaxPredictedRating(user.getId());

  double maxRatingDelphiIR  =
➥   delphiIR.getMaxPredictedRating(user.getId());

  double[] sortedMaxR =
➥   {maxRatingDelphiUIC, maxRatingDelphiUR, maxRatingDelphiIR};

  Arrays.sort(sortedMaxR);

  maxR = sortedMaxR[2];

  // Вспомогательная переменная
  double scaledRating = 1.0d;

  // Рекомендатель 1 - На основе контента "пользователь-предмет"
  double scaling = maxR/maxRatingDelphiUIC;

  // Задание произвольного порогового значения и его масштабирование
  double scaledThreshold = 0.5 * scaling;

  List<PredictedItemRating> uicList =
➥   new ArrayList<PredictedItemRating>(topN);

  uicList = delphiUIC.recommend(user, topN);

  for (PredictedItemRating pR : uicList) {

    scaledRating = pR.getRating(6) * scaling;

    if (scaledRating < scaledThreshold) {
       uicList.remove(pR);
    } else {
       pR.setRating(scaledRating);
    }
  }

  // Рекомендатель 2 - Коллаборативная фильтрация на основе пользователей
  scaling = maxR/maxRatingDelphiUR;

  scaledThreshold = 0.5 * scaling;

  List<PredictedItemRating> urList =
➥   new ArrayList<PredictedItemRating>(topN);

urList = delphiUR.recommend(user, topN);

  for (PredictedItemRating pR : urList) {

    scaledRating = pR.getRating(6) * scaling;
```

Максимальные оценки, предсказанные рекомендателем

Максимальная из оценок, предсказанных всеми рекомендателями

maxR – это максимальная предсказанная оценка

Создание масштабирующего коэффициента для каждой системы

Получение рекомендации от каждой системы

Масштабированные оценки должны быть выше порогового значения

Создание масштабирующего коэффициента для каждой системы

Получение рекомендации от каждой системы

```
        if (scaledRating < scaledThreshold) {
            urList.remove(pR);
        } else {
            pR.setRating(scaledRating);
        }
}

// Рекомендатель 3 -- Коллаборативная фильтрация на основе предметов
scaling = maxR/maxRatingDelphiIR;

scaledThreshold = 0.5 * scaling;

List<PredictedItemRating> irList =
➥ new ArrayList<PredictedItemRating>(topN);

irList = delphiIR.recommend(user, topN);

for (PredictedItemRating pR : irList) {

       scaledRating = pR.getRating(6) * scaling;

       if (scaledRating < scaledThreshold) {
            irList.remove(pR);
       } else {
            pR.setRating(scaledRating);
       }
}

double urRating=0;
double irRating=0;
double vote=0;

for (PredictedItemRating uic : uicList) {

      //Инициализация
      urRating=0; irRating=0; vote=0;

    for (PredictedItemRating ur : urList) {
      if (uic.getItemId() == ur.getItemId()) {
         urRating = ur.getRating(6);
      }
    }

    for (PredictedItemRating ir : irList) {
       if (uic.getItemId() == ir.getItemId()) {
         irRating = ir.getRating(6);
       }
    }

    vote = (uic.getRating(6)+urRating+irRating)/3.0d;

  recommendations.add(
➥ new PredictedItemRating(user.getId(), uic.getItemId(), vote));
}
```

Масштабированные оценки должны быть выше порогового значения

Создание масштабирующего коэффициента для каждой системы

Получение рекомендации от каждой системы

Масштабированные оценки должны быть выше порогового значения

Получение среднего значения и выполнение соответствующего масштабирования

```
    rescale(recommendations,maxR);

    return PredictedItemRating
➥   .getTopNRecommendations(recommendations, topN);
}
```

Вы можете и дальше улучшать свои рекомендации, нацеливая их на предпочтения каждого отдельного пользователя, зарегистрированного на сайте MyDiggSpace.com, путем объединения результатов, полученных от класса `DiggDelphi` и классификатора `NaiveBayes`, с которым мы впервые познакомились в главе 2. Более подробно этот подход описан в разделе «Сделать» в конце этой главы. Дополнить результаты базовых рекомендателей можно с помощью любых систем обучения (некоторые их них представлены в главе 5) и методик оптимизации. Данный подход, состоящий в объединении методик со слоем, инкапсулирующим обучение, набирает популярность, получая поддержку со стороны как лидеров индустрии, так и академических кругов (см. также главу 6).

К этому времени у вас должно сложиться прочное представление об объединении систем выработки рекомендаций и совместного использования их возможностей, позволяющих находить друзей и интересные статьи для пользователей вашего веб-приложения. В следующем разделе мы рассмотрим другой пример – выработку рекомендаций по фильмам на сайте, подобном сайту Netflix. Главная особенность таких примеров – наборы данных большого объема.

3.4. Рекомендации фильмов на сайте, подобном сайту Netflix.com

Во введении мы говорили о фирме Netflix, Inc., которая организовала крупнейшую в мире интернет-службу видеопроката, где подписчикам (а их больше 7 000 000) предлагается 90 000 наименований DVD-дисков и пополняющаяся библиотека, которая предоставляет для онлайн-просмотра на пользовательских компьютерах больше 5000 полнометражных фильмов и телепрограмм. Если вы помните, отчасти своим успехом в Интернете фирма Netflix обязана тому, что обеспечивает пользователям легкий способ выбора фильмов из громадного перечня наименований. Ядро этой функциональной возможности – система выработки рекомендаций Cinematch. Ее задача – предсказать, понравится ли данный фильм конкретному пользователю, исходя из того, насколько ему нравятся или нет другие фильмы.

3.4.1. Введение в наборы данных о кинофильмах и рекомендателях

В этом разделе мы опишем систему выработки рекомендаций, задача которой – та же, что и у системы Cinematch. Мы будем работать с обще-

доступными данными из проекта MovieLens. Проект MovieLens – это бесплатная услуга, предоставляемая исследовательской лабораторией GroupLens Миннесотского университета. В рамках этого проекта организован веб-сайт, на котором предлагаются рекомендации по фильмам. Вы можете опробовать, как он работает: *http://www.movielens.org/quickpick*. На веб-сайте лаборатории GroupLens доступны два набора данных MovieLens.

Первый набор данных[1] содержит 100 000 рейтинговых оценок, которые 943 пользователя выставили для 1682 фильмов. Второй набор данных[2] насчитывает 1 000 000 рейтинговых оценок, выставленных 6040 пользователями для 3900 фильмов. Пожалуйста, убедитесь в том, что вы прочли лицензионное соглашение и условия использования. Формат данных из этих двух наборах отличается. Мы нашли формат второго (1 Мбайт рейтинговых оценок) набора данных более подходящим и удобным; он включает только три файла: *movies.dat*, *ratings.dat* и *users.dat*. Однако из-за проблем производительности мы хотим использовать набор данных поменьше. Так что мы для удобства преобразовали исходный формат меньшего набора (100 Кбайт рейтинговых оценок) в формат большего набора. Исходные данные и больший набор данных можно получить на веб-сайте GroupLens. Вы должны извлечь эти данные в каталог *C:/iWeb2/data/ch03/MovieLens/*; если хотите извлекать данные в другое место, измените метод `createDataset` в листинге 3.18, так чтобы этот метод получал путь доступа к каталогу с данными в качестве аргумента.

Крупные системы выработки рекомендаций, например, такие как Netflix и Amazon.com, в значительной степени полагаются на коллаборативную фильтрацию на основе предметов [см. Linden, G., B. Smith, and J. York]. Этот подход, который мы описали в разделах 3.2.1 и 3.2.2, можно улучшить с помощью трех основных компонентов.

Первый компонент – это *нормализация данных* (data normalization). Этим модным термином определяют то, что легко понять на наглядном примере. Если пользователь склонен ставить всем фильмам высокие оценки (шаблон оценивания, который мы взяли на вооружение для нашего искусственного ранжирования предметов в предыдущих разделах), имеет смысл принимать во внимание относительные рейтинговые оценки пользователя, а не их абсолютные значения.

Второй основной компонент – это *выбор соседей* (neighbor selection). Применяя коллаборативную фильтрацию, мы определяем набор пред-

---

[1] Исходные данные доступны по адресу: *http://www.grouplens.org/system/files/ml-data.tar__0.gz*.

[2] Исходные данные доступны по адресу: *http://www.grouplens.org/system/files/million-ml-data.tar__0.gz*.

метов (или пользователей), чьи рейтинговые оценки будут использованы для того, чтобы высказать предположение о рейтинговой оценке неоцененного предмета. Исходя из этого требования, естественны два вопроса: Сколько нам нужно соседей? Как выбрать «наилучших» соседей, то есть обеспечивающих наиболее точное предсказание рейтинговой оценки?

Третий основной компонент коллаборативной фильтрации – определение *веса соседа* (neighbor weights), который покажет, насколько значащей является рейтинговая оценка каждого соседа. В [Bell, R.M., and Y. Koren] показано, что нормализация данных и выбор веса соседа – два самых важных компонента для повышения точности подхода с применением коллаборативной фильтрации.

Начнем с описания сценария этого примера, предназначенного для выполнения в среде Bean Shell. В листинге 3.18 показано, как загрузить данные, создать экземпляр нашего рекомендателя (`MovieLensDelphi`), отобрать пользователей и получить рекомендации для каждого из них.

Листинг 3.18. MovieLensDelphi: рекомендации для наборов данных MovieLens

```
MovieLensDataset ds = MovieLensData.createDataset();   ← Загрузка набора данных
MovieLens

MovieLensDelphi delphi = new MovieLensDelphi(ds);      ← Создание рекомендателя

iweb2.ch3.collaborative.model.User u1 = ds.getUser(1);  ← Отбор пользователей
                                                          и получение рекомендации
delphi.recommend(u1);
iweb2.ch3.collaborative.model.User u155 = ds.getUser(155);
delphi.recommend(u155);
iweb2.ch3.collaborative.model.User u876 = ds.getUser(876);
delphi.recommend(u876);
```

Первым мог бы быть любой пользователь, поэтому берем пользователя с идентификатором, равным 1. Два других пользователя были найдены при выполнении команды `Delphi.findSimilarUsers(u1);`. Мы сделали это так, что можем быстро проверить, имеют ли смысл наши рекомендации. Можно обоснованно предположить, что если два пользователя обладают сходством и ни один из них не видел конкретный фильм, то, рекомендуя фильм одному из этих пользователей, вполне можно порекомендовать его и другому. На рис. 3.12 показаны результаты, которые мы получили, выполнив этот сценарий, и адекватность которых подтверждена проверкой.

Эти наборы данных не так велики, как те, которые можно найти в приложениях Amazon.com или Netflix, но они, безусловно, намного больше, чем все то, с чем мы сталкивались до сих пор, и их объем достаточно велик, чтобы эти данные были реалистичными. На выполнение сцена-

```
bsh % iweb2.ch3.collaborative.model.User u1 = ds.getUser(1);
bsh % delphi.recommend(u1);

Recommendations for user 1:

Item: Yojimbo (1961)              , predicted rating: 5.000000
Item: Loves of Carmen, The (1948) , predicted rating: 4.303400
Item: Voyage to
the Beginning of the World (1997) , predicted rating: 4.303400
Item: Baby, The (1973)            , predicted rating: 4.303400
Item: Cat from Outer Space,
The (1978)                        , predicted rating: 4.123200

bsh % iweb2.ch3.collaborative.model.User u155 = ds.getUser(155);
bsh % delphi.recommend(u155);

Recommendations for user 155:

Item: Persuasion (1995)              , predicted rating: 5.000000
Item: Close Shave, A (1995)          , predicted rating: 4.373000
Item: Notorious (1946)               , predicted rating: 4.181900
Item: Shadow of a Doubt (1943)       , predicted rating: 4.101800
Item: Crimes and Misdemeanors (1989) , predicted rating: 4.061700

bsh % iweb2.ch3.collaborative.model.User u876 = ds.getUser(876);
bsh % delphi.recommend(u876);

Recommendations for user 876:

Item: Third Man, The (1949)      , predicted rating: 5.000000
Item: Bicycle Thief,
The (Ladri di biciclette)(1948)  , predicted rating: 4.841200
Item: Thin Blue Line, The (1988) , predicted rating: 4.685600
Item: Loves of Carmen, The (1948), predicted rating: 4.600200
Item: Heaven's Burning (1997)    , predicted rating: 4.600200
```

Рис. 3.12. *Рекомендации, полученные от рекомендателя MovieLensDelphi и основанные на наборе данных MovieLens*

рия для малого набора данных MovieLens (100 Кбайт рейтинговых оценок) потребуется примерно от 30 секунд до минуты, время будет затрачено, главным образом, на создание рекомендателя. Как мы увидим, за это время рекомендатель проделает большую работу по обработке данных. Выработка самих рекомендаций осуществляется относительно быстро, как правило, меньше чем за одну секунду.

3.4.2. Нормализация данных и коэффициенты корреляции

Как было обещано, в примере из этого раздела мы обогатили подход с применением коллаборативной фильтрации введением двух новых инструментов. Первый инструмент – нормализация данных, второй – новый показатель сходства, позволяющий установить наличие корреляции между предметами. Этот новый показатель сходства называется *коэффициентом линейной корреляции* (linear correlation coefficient), который также называют *коэффициентом корреляции со смешанным моментом* (product-moment correlation coefficient) или *r Пирсона* (Pearson's r). Логика вычисления этого коэффициента для двух массивов x и y довольно проста. В листинге 3.19 показаны три метода, отвечающих за это вычисление.

Листинг 3.19. Вычисление коэффициента линейной корреляции (r Пирсона)

```
public double calculate() {

    if( n == 0) {
        return 0.0;
    }
    double rho=0.0d;
    double avgX = getAverage(x);
    double avgY = getAverage(y);

    double sX = getStdDev(avgX,x);
    double sY = getStdDev(avgY,y);

    double xy=0;

    for (int i=0; i < n; i++) {

        xy += (x[i]-avgX)*(y[i]-avgY);  ❶
    }

      if( sX == ZERO || sY == ZERO) {  ❷

         double indX = ZERO;
         double indY = ZERO;

      for (int i=1; i < n; i++) {

         indX += (x[0]-x[i]);
         indY += (y[0]-y[i]);
      }

          if (indX == ZERO && indY == ZERO) {
         // Все точки ссылаются на одно и то же значение
         // Это - вырожденный случай корреляции
         return 1.0;
      } else {
```

Вычисление среднего значения для каждого вектора

Вычисление стандартных отклонений для каждого вектора

```
            // Отличаются или значения X, или значения Y
            if (sX == ZERO) {
                sX = sY;
                } else {
                sY = sX;
                }
            }
        }
        rho = xy / ((double)n*(sX*sY));  ← Значение r Пирсона
        return rho;
    }

    private double getAverage(double[] v) {
        double avg=0;

        for (double xi : v ) {
            avg += xi;
        }
        return (avg/(double)v.length);
    }

    private double getStdDev(double m, double[] v) {
        double sigma=0;

        for (double xi : v ) {
               sigma += (xi - m)*(xi - m);
        }

        return Math.sqrt(sigma / (double)v.length);
    }
```

❶ – это вычисление векторного произведения точечных отклонений от среднего значения. ❷ – особый (вырожденный) случай, где все точки имеют точно совпадающие значения для X или для Y либо для обоих. Этот случай должен обрабатываться отдельно, потому что он приводит к делению на ноль.

Метод `getAverage` не требует объяснений; он обеспечивает вычисление среднего вектора, который предоставляется в качестве аргумента. Метод `getStdDev` позволяет вычислить среднеквадратическое отклонение *(стандартное отклонение)* для данных вектора, переданного в качестве второго аргумента; первым аргументом метода должно быть среднее значение. Есть более интеллектуальный способ получения этой величины, позволяющий избежать возни с численными расчетами, который называется *ошибкой округления* (roundoff error); прочтите статью [Chan, T.F., G.H. Golub, and R.J. LeVeque] об улучшенном двухпроходном алгоритме.

Сходство, вычисленное на основе корреляции Пирсона, – широко используемый показатель, обладающий следующими свойствами.

- Если он равен нулю, два предмета являются (статистически) *некоррелированными* (uncorrelated).
- Если он равен 1, рейтинговые оценки этих двух предметов точно соответствуют прямой с положительным наклоном; например (1, 2), (3, 4), (4, 5), (4, 5), где первое число в круглых скобках означает рейтинговую оценку первого предмета, а второе – рейтинговую оценку второго предмета. Это явление называется *строго положительной корреляцией* (complete positive correlation). Другими словами, зная рейтинговые оценки одного предмета, можно с высокой степенью вероятности сделать вывод о рейтинговых оценках другого предмета.
- Если он равен –1, рейтинговые оценки этих двух предметов точно соответствуют прямой, но с отрицательным наклоном; например (1, 5), (2, 4), (3, 3), (4, 2). Это явление называется *строго отрицательной корреляцией* (complete negative correlation). В этом случае мы также можем сделать вывод о рейтинговых оценках одного предмета на основании рейтинговых оценок другого предмета, но теперь при каждом повышении рейтинговых оценок первого предмета рейтинговые оценки второго предмета будут снижаться.

Если предметы коррелированы линейно, коэффициент линейной корреляции является хорошим показателем силы этой корреляции. Фактически, если вы получаете для своего набора данных прямую, то коэффициент линейной корреляции отражает расстояние, на которое ваши рейтинговые оценки отклоняются от этой линии. Но эту радужную картину кое-что портит. К сожалению, этот коэффициент является скорее слабым показателем корреляции, если никакой связи не существует! Что вы сказали? Да, это правда.

Знаменитый контрпример известен как *квартет Энскомба* (Anscombe's quartet). На рис. 3.13 представлен квартет Энскомба для четырех различных пар значений; этот рисунок в формате SVG доступен по адресу *http://en.wikipedia.org/wiki/Image:Anscombe.svg*.

Говоря обычным языком, если вы наносите на график рейтинговые оценки двух предметов в функциональной зависимости друг от друга и эта кривая подобна графику, изображенному слева вверху на рис. 3.13, коэффициент линейной корреляции является показателем, поддающимся интерпретации. На других графиках коэффициент корреляции Пирсона имеет то же самое значение, но его смысл сомнителен; наборы данных тщательно подобраны искусственно, так что они к тому же имеют одинаковую среднюю величину, одинаковое среднеквадратическое отклонение и однаковый подбор прямой *(y = 3 + 0,5 * x)*. Эта неспособность определить значение коэффициента линейной корреляции (Пирсона) вынуждает использовать другую метрику сходства – *непараметрическую корреляцию* (nonparametric correlation). Есть два популярных коэффициента непараметрической корреляции: *коэффици-*

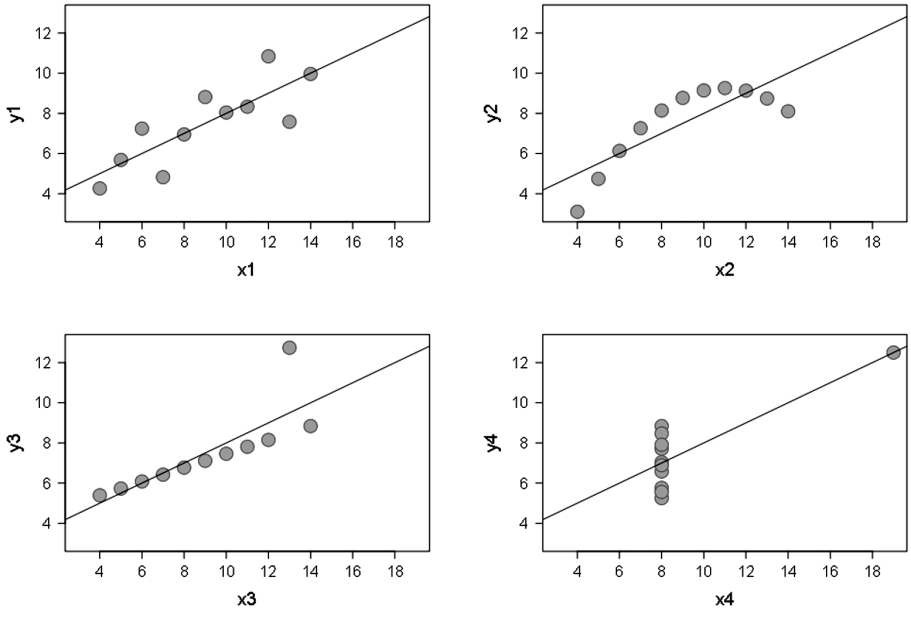

Рис. 3.13. *Квартет Энскомба: четыре набора данных, имеющих одинаковую корреляцию Пирсона, но разные распределения*

ент ранговой корреляции Спирмана (Spearman rank-order correlation coefficient, rs) и *тау Кендалла* (Kendall's tau, τ). Эти метрики ценой некоторой потери информации гарантируют, что выявленная корреляция действительно присутствует в данных, если на это указывают их значения. Непараметрическую корреляцию мы рассматриваем в разделе «Сделать», поскольку во многих случаях распределение рейтинговых оценок будет выглядеть так, как это изображено на графике в правом нижнем углу рис. 3.13. Тем не менее в дальнейшем будем исходить из предположения, что всякий раз, когда рейтинговые оценки предметов коррелированы, они коррелированы линейно, и мы можем уверенно использовать корреляцию Пирсона. Дополнительную информацию о непараметрических корреляциях вам поможет найти раздел ссылок.

Обсудив новые возможности оценки сходства, открываемые линейным коэффициентом (r Пирсона) и непараметрической корреляцией, мы пойдем дальше и продемонстрируем способ, позволяющий добиться нормализации данных. В листинге 3.20 показан программный код, предназначенный именно для этой цели, – один из конструкторов класса `PearsonCorrelation`. Первый аргумент обеспечивает ссылку на исходный набор данных, а два другие аргумента являются ссылками на предметы, корреляцию которых мы хотим вычислить. Видно, что массивы, созданные для вычисления корреляции Пирсона, ссылаются не на исходно

зафиксированные рейтинговые оценки каждого пользователя, а на новый набор данных, для получения которого из пользовательских оценок вычли среднюю оценку предмета. Понятно, что это не единственный способ достижения нормализации данных. [Bell, R.M., and Y. Koren] описывают более изощренные методики нормализации в применении к набору данных, для которого проводится конкурс на приз Netflix.

Листинг 3.20. Нормализация данных по усредненной рейтинговой оценке предметов

```
public PearsonCorrelation(Dataset ds, Item iA, Item iB) {

  double aAvgR = iA.getAverageRating();
  double bAvgR = iB.getAverageRating();

  Integer[] uid = Item.getSharedUserIds(iA, iB);

  n = uid.length;

  x = new double[n];
  y = new double[n];

  User u;

  double urA=0;
  double urB=0;

  for (int i=0; i<n; i++) {

    u = ds.getUser(uid[i]);

    urA = (double) u.getItemRating(iA.getId()).getRating();
    urB = (double) u.getItemRating(iB.getId()).getRating();

    x[i] = urA - aAvgR;
    y[i] = urB - bAvgR;
  }
}
```

Нормализация данных и использование корреляции Пирсона встроены в класс `PearsonCorrelation`, а применение этих методик инкапсулировано в классе `MovieLensItemSimilarity`. По этой причине класс `MovieLensDelphi` несколько отличается от других классов типа `Delphi`. Эти различия заметны в программном коде из листинга 3.21.

Листинг 3.21. Вычисление рейтинговой оценки включает перенормировку данных и повторное масштабирование

```
private double estimateItemBasedRating(User user, Item item) {

  double itemRating = item.getAverageRating();

  int itemId = item.getId();
  int userId = user.getId();
```

```
  double itemAvgRating = item.getAverageRating();
  double weightedDeltaSum = 0.0;

  int sumN=0;

  // Проверка: оценивал ли пользователь этот предмет раньше
  Rating existingRatingByUser = user.getItemRating(item.getId());

  if (existingRatingByUser != null) {

    itemRating = existingRatingByUser.getRating();

  } else {

          double similarityBetweenItems = 0;

          double weightedDelta = 0;
          double delta = 0;

          for (Item anotherItem : dataSet.getItems()) {

      // Учитываем только те предметы, которые были оценены пользователем
        Rating anotherItemRating =
➥    anotherItem.getUserRating(userId);

        if (anotherItemRating != null) {
            delta = itemAvgRating - anotherItemRating.getRating();

          similarityBetweenItems =
➥    itemSimilarityMatrix.getValue(itemId, anotherItem.getId());

             if (Math.abs(similarityBetweenItems) >
➥    similarityThreshold) {  ❶

             weightedDelta = similarityBetweenItems * delta;

             weightedDeltaSum += weightedDelta;

             sumN++;
             }
        }
      }

      if (sumN > 0) {
        itemRating = itemAvgRating -
➥    (weightedDeltaSum/(double) sumN)  ❷
      }
   }

   return itemRating;
}

public List<PredictedItemRating> getTopNRecommendations(
➥  List<PredictedItemRating> recommendations, int topN) {
  PredictedItemRating.sort(recommendations);
```

Цикл по всем предметам

Перенормировка данных

Получение сходства двух предметов

```
    double maxR = recommendations.get(0).getRating();
    double scaledR;

    List<PredictedItemRating> topRecommendations =
➥    new ArrayList<PredictedItemRating>();

      for(PredictedItemRating r : recommendations) {

       if( topRecommendations.size() >= topN ) {
         // Имеется достаточное количество рекомендаций.
         break;
       }

       scaledR = r.getRating() * (5/maxR);
       r.setRating(scaledR);

       topRecommendations.add(r);
    }

    return topRecommendations;
  }
```

Мы взвешиваем отклонение ❶ от среднего значения на основе сходства двух предметов и назначаем ❷ рейтинговую оценку, исходя из полученного для этого предмета среднего значения и суммы взвешенных отклонений.

Основанием для *перенормировки данных* (data renormalization) является тот факт, что полученные нами показатели сходства были сформированы с использованием средней рейтинговой оценки предмета, поэтому чтобы вычислить предсказанную рейтинговую оценку предмета, необходимо осуществить перенормировку из разностей (дельта) в реальные рейтинговые оценки. Один из недостатков подобной нормализации данных заключается в том, что максимальное значение предсказанной рейтинговой оценки может выпасть из диапазона приемлемых значений. Таким образом, требуется повторно масштабировать предсказанные рейтинговые оценки, как показано в методе `getTopNRecommendations`.

3.5. Масштабная реализация и вопросы оценки

Коммерческие системы выработки рекомендаций функционируют в жестких условиях. Число пользователей обычно составляет порядка миллионов, а число предметов – порядка сотен тысяч. Дополнительным требованием является способность предоставлять рекомендации в реальном времени (как правило, время отклика не должно превышать одну секунду), не жертвуя при этом качеством рекомендаций. Как мы видели, можно со временем повысить точность предсказаний путем накопления рейтинговых оценок, полученных от каждого пользователя. Но в реальной жизни крайне важно давать превосходные рекомендации новым пользователям, для которых у нас мало оценок по

определению. Другое жесткое требование к современным системам выработки рекомендаций – способность обновлять предсказания по мере поступления новых рейтинговых оценок. На крупных коммерческих сайтах за несколько часов могут быть выставлены тысячи рейтинговых оценок и приобретены тысячи товаров, а за весь день эти количества измеряются, вероятно, десятками тысяч. Важно, чтобы систему выработки рекомендаций можно было обновлять, внося в нее дополнительную информацию, и такое обновление должно происходить в рабочем режиме – без простоя.

Предположим, вы написали программу для рекомендателя и удовлетворены скоростью его работы и тем объемом данных, который он может обрабатывать. Хорош ли ваш рекомендатель? Что толку в наличии быстродействующего и масштабируемого рекомендателя, дающего плохие рекомендации! Поэтому давайте поговорим об оценке точности результатов работы системы выработки рекомендаций. Изучив соответствующую литературу, вы обнаружите десятки количественных показателей и несколько качественных методик, позволяющих оценить результаты работы систем выработки рекомендаций. Такое изобилие показателей и методик отражает сложность проведения целенаправленной, беспристрастной и точной оценки рекомендаций. Если вас интересует эта тема, много дополнительной информации можно найти в обзорной статье [Herlocker, J.L., J.A. Konstan, L.G. Terveen, and J.T. Riedl].

Мы написали класс, позволяющий оценить наши рекомендации, полученные на основе данных MovieLens, путем вычисления *среднеквадратического отклонения, или среднеквадратичной ошибки* (root mean square error, RMSE) предсказанных рейтинговых оценок. RMSE – простая, но надежная методика оценки точности рекомендаций. Этот показатель обладает двумя главными характеристиками: 1) он всегда возрастает (вы не прославитесь точным предсказанием рейтинговых оценок), и 2) при возведении разностей в квадрат большие разности (>1) увеличиваются, и неважно, заниженной или завышенной оказывается ваша оценка.

Мы можем утверждать, что методика RMSE, скорее всего, слишком наивна. Рассмотрим два случая. В первом случае мы рекомендовали фильм с 4 звездочками, пользователю этот фильм совсем не понравился (он дал рекомендованному фильму 2 звездочки); во втором случае мы рекомендовали фильм с 3 звездочками, а пользователю он понравился (пользователь дал ему 5 звездочек). В обоих случаях вклад ошибки RMSE одинаков, но пользователь, скорее всего, в первом случае больше неудовлетворен, чем во втором; а уж какую неудовлетворенность должны испытывать мы – кому и знать, как не нам!

Вы можете найти программный код для вычисления ошибки RMSE в классе `RMSEEstimator`. В листинге 3.22 показано, как можно оценить точность предсказаний нашего рекомендателя `MovieLensDelphi`.

Листинг 3.22. Вычисление среднеквадратичной ошибки для рекомендателя

```
MovieLensDataset ds = MovieLensData.createDataset(100000);  ❶

MovieLensDelphi delphi = new MovieLensDelphi(ds);

RMSEEstimator rmseEstimator = new RMSEEstimator();

rmseEstimator.calculateRMSE(delphi);
```

Мы создаем набор данных, для которого изымаем часть данных в объеме 100 Кбайт из большого набора данных MovieLens ❶, включающего миллион рейтинговых оценок. На оставшихся данных о рейтинговых оценках объемом 900 Кбайт рекомендатель будет тренироваться, а для оценки результатов его работы будут использованы изъятые данные в объеме 100 Кбайт; остальная часть сценария не требует пояснений. Если выполнить этот сценарий с тем программным кодом, описанном в этом разделе, то полученная вами ошибка RMSE должна быть равна 1,0256. Неплохой показатель для ошибки RMSE, но и не очень хороший. Мы настоятельно рекомендуем вам улучшить этот результат и задались целью получить значение ошибки RMSE меньше 1. Для сравнения сообщим, что у лучших команд, которые боролись за приз Netflix, показатель ошибки RMSE имел значение от 0,86 до 0,88. Так что хотя набор данных и отличается, не огорчайтесь, если ваши усовершенствования дадут показатель ошибки RMSE, примерно равный 0,9, – это был бы громадный успех для вас и для нас!

3.6. Заключение

Из этой главы вы узнали о понятиях расстояния и сходства между пользователями и предметами. Мы убедились в том, что нельзя подходить ко всему с одной меркой и что показатель сходства следует выбирать осмотрительно. Мы также представили вам несколько показателей – метрику Жаккарда, корреляцию Пирсона, а также наши варианты этих метрик. Формулы вычисления показателей сходства должны обеспечивать результаты, не противоречащие нескольким фундаментальным правилам, но в остальном мы вольны выбирать те из них, которые позволят получить результаты, наиболее подходящие для наших целей.

Рассмотрены две основные категории методик выработки рекомендаций – коллаборативная фильтрация и подход, основанный на сходстве контента. Мы прошли по пути построения музыкального интернет-магазина, подробно, но доходчиво демонстрируя базовые принципы. В процессе построения примеров мы создали инфраструктуру, на основе которой вы сможете написать систему выработки рекомендаций для собственного приложения.

Наконец, мы замахнулись на два примера более общего характера. Первый – гипотетический веб-сайт, где интерфейс Digg API использовался для того, чтобы извлечь контент пользователей и затем на его основе проанализировать сходство между ними, а также для того, чтобы рекомендовать пользователям статьи, которые они еще не видели. В этом примере мы обратили внимание на существование результатов второго порядка, и к тому же *высшего порядка* (higher-order effects), и предложили способ, позволяющий задействовать эти результаты, чтобы повысить точность рекомендаций. Наш второй пример был связан с рекомендациями по фильмам и позволил представить понятие нормализации данных, а также часто используемый коэффициент линейной корреляции (Пирсона). Что касается последнего, мы также предоставили класс, который обеспечивает оценку точности рекомендаций, основанную на величине среднеквадратичной ошибки.

В обоих примерах мы показали, что по мере возрастания сложности и масштаба задачи все более настоятельной становится потребность в применении комбинации методик, чтобы повысить эффективность и качество вырабатываемых рекомендаций. В связи с этим мы обсудили возможность повторного использования в примере с сайтом MyDiggSpace.com той информации, которую вы получили в результате анализа пользовательских переходов по ссылкам. Этот лейтмотив будет сопровождать нас на протяжении всей книги – объединение методик, охватывающих разные аспекты задачи, может привести – и часто приводит – к выработке более точных рекомендаций.

В следующей главе мы познакомим вас еще с одним семейством интеллектуальных алгоритмов – с алгоритмами кластеризации. И если раздел «Сделать» вами еще не проработан, то, пожалуй, самое время заняться им сейчас, пока у вас в голове еще не стихли отголоски материала по выработке рекомендаций.

3.7. Сделать

1. *Показатели сходства*. Реализуйте коэффициент Жаккарда для `MusicUsers`. Какие отличия вы наблюдаете? Вариантом коэффициента Жаккарда является *коэффициент Танимото* (Tanimoto metric), который в большей степени подходит для непрерывных величин. Коэффициент Танимото вычисляется как отношение пересечения двух множеств ($N_i = |X \cap Y|$) к объединению ($N_u = |X| + |Y|$) минус пересечение – $T = N_i / (N_u - N_i)$.

   Например, если X = {baseball, basketball, volleyball, tennis, golf} и Y = {baseball, basketball, cricket, running}, то значение коэффициента Танимото вычисляется как 2 / ((5 + 4) – 2), что приблизительно равно 0,2857. Выведите формулу для случая векторов (Java-массивы

`double[] x` и `double[] y`). Подсказка: пересечение соответствует внутреннему произведению двух векторов, а объединение – сумме их модулей.

Еще один интересный показатель сходства – *расстояние городских кварталов* (city block). Своим названием эта метрика обязана тому предположению, что значениями векторов X и Y являются координаты на многомерной ортогональной решетке. Если векторы двумерные, это напоминает способ, которым воспользовался бы водитель такси, чтобы объяснить вам, как проехать куда-то в городе: «Эмпайр-стейт-билдинг находится в двух кварталах к югу и в трех кварталах к востоку отсюда». Если вам нравится эта метрика или вы хотите проанализировать, в каких случаях лучше применять именно ее, подробное объяснение есть в книге «Taxicab Geometry: An Adventure in Non-Euclidean Geometry» [Krause, E. F.].

2. *Изменение диапазона предсказания*. Вас когда-нибудь интересовало, почему на разных веб-сайтах хотят, чтобы вы оценивали фильмы, музыкальные записи и другие товары, выбирая в качестве оценки одно целое значение из интервала от 1 до 5 (включительно)? Почему бы не взять значение из интервала от 1 до 10? Или даже от 1 до 100? Разве это не позволило бы вам точнее выразить степень своей удовлетворенности товаром? И почему бы не оценивать различные аспекты товара? В случае кинофильмов можно было бы оценить сценарий, игру актеров, саундтрек, а также визуальные эффекты. Вы можете дополнить программный код, который мы предоставили в этой главе, и поэкспериментировать в указанных направлениях. Можете ли вы выявить какие-либо потенциальные проблемы?

3. *Уточнение рекомендаций посредством создания ансамбля методов*. Методика, которая становится все более популярной, заключается в объединении независимых методик с целью повышения точности совместно вырабатываемых рекомендаций. Есть несколько серьезных теоретических причин для того, чтобы заниматься созданием ансамблей методов; если эта тема вас интересует, прочтите статью [Dietterich, T. G.]. Кроме теории есть эмпирическое доказательство того, что ансамбль методов способен обеспечить лучшие результаты, чем отдельные методики. В конкурсе на приз Netflix лидируют (ко времени написания этой книги) [Bell, R. M., and Y. Koren], их установка: «Мы не нашли совершенную модель. Вместо этого мы получили лучшие результаты, объединив предсказания моделей, дополняющих друг друга».

   Как насчет того, чтобы объединить некоторые рекомендатели – те, что мы вам предоставили в этой главе, а также те, что вы сможете придумать сами, – и сравнить результаты ансамбля рекомендателей с результатами каждого отдельного рекомендателя? Если первые

окажутся лучше – ваша «гремучая смесь» сработала! Если нет, проанализируйте работу использованных рекомендателей и выясните, насколько они охватывают различные аспекты проблемы.

4. *Минимизация ошибки округления*. Как вам, скорее всего, известно, стандартные числовые типы языка Java и большинства других языков программирования хранят значения с ограниченной точностью. Представление чисел типа `integer` или `long` является точным даже несмотря на то, что диапазон их значений конечен и определяется количеством битов, выделяемых под каждый из этих типов. Но стоит ввести арифметику с плавающей запятой (`float` и `double`), как возникает ряд вопросов, обусловленных погрешностью представления чисел. В лучшем случае вас это не волнует, а в худшем – для всех таких значений можно использовать тип `double`.

   Тем не менее интенсивное использование в интеллектуальных приложениях численных расчетов требует, чтобы вы знали о влиянии конечной точности реальных чисел на результат вычислений, особенно на результаты сложения или умножения очень маленьких или очень больших чисел. Рассмотрим ошибку округления, которую мы упомянули, рассматривая получение оценки среднеквадратического отклонения с помощью класса `PearsonCorrelation`. Наименьшее число с плавающей запятой, в результате сложения которого с величиной 1,0 будет получен результат, отличный от 1,0, называется *машинной точностью* (ε). Почти каждая арифметическая операция между числами с плавающей запятой вносит незначительную погрешность порядка величины ε. Эта ошибка называется *ошибкой округления*.

   Прочтите статью [Chan, T. F., G. H. Golub, and R. J. LeVeque] об улучшенном двухпроходном алгоритме и реализуйте соответствующее вычисление среднеквадратического отклонения. Краткое описание этого алгоритма можно также найти в монументальном труде «Numerical Recipes: The Art of Scientific Computing» [Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery]. Замечаете ли вы ощутимое отличие в результатах? Как вы думаете, что произойдет, если вы используете наборы данных гораздо большего размера, чем рассмотренные в этой книге? Обратите внимание: основные принципы этого алгоритма также вполне применимы при вычислении ошибки RMSE, с помощью которого мы оценивали точность наших рекомендаций.

5. *Непараметрическая, или ранговая корреляция*. Корреляции из этой категории полезны, когда у вас есть причина усомниться в обоснованности допущения о линейности, на котором основан показатель корреляции Пирсона. Вы можете создать новые классы для определения сходства, взяв за основу показатель такого типа, который

в обмен на некоторую информацию о данных позволяет гарантировать истинность корреляции между двумя наборами данных – в нашем случае речь идет о двух наборах рейтинговых оценок. Основная идея непараметрической корреляции – замещение значений некоторой величины рангом значения в наборе данных. Наиболее распространенные коэффициенты непараметрической корреляции – коэффициент ранговой корреляции Спирмана (rs) и тау Кендалла (τ). Все, что касается этих коэффициентов, вы можете прочесть в мастерски написанной книге «Numerical Recipes: The Art of Scientific Computing» [Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery].

В случае оценки фильмов по пятибалльной шкале от 1 до 5 вы получаете массу конфликтующих рангов величин; например, для множества фильмов значение будет равным 4. Но тем самым вам предоставляется возможность творчески подойти к применению таких корреляций. Что если вы задействуете время выставления оценок, чтобы распутать узелок с одинаковыми значениями? Реализуйте такой подход и сравните полученные результаты с теми, которые вы получаете с помощью обычной, бесхитростной корреляции Пирсона.

3.8. Ссылки

- Bell, R. M., and Y. Koren. «Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights». IEEE International Conference on Data Mining (ICDM'07), 2007. *http://www.research.att.com/~yehuda/pubs/BellKorIcdm07.pdf*.
- Chan, T. F., G. H. Golub, and R. J. LeVeque. «Algorithms for computing the sample variance: Algorithms and recommendations». American Statistician, vol. 37, pp. 242–247, 1983.
- Dietterich, T. G. «Ensemble methods in machine learning». Multiple Classifier Systems, (Editors: J. Kittler and F.Roli) volume 1857 of Lecture Notes in Computer Science, Cagliari, Italy. Springer, pp. 1–15, 2000. *http://citeseer.ist.psu.edu/dietterich00ensemble.html*.
- Estes, W. K. «Classification and Cognition». Oxford University Press, 1996.
- Herlocker, J. L., J. A. Konstan, L. G. Terveen, and J. T. Riedl (2004). «Evaluating Collaborative Filtering Recommender Systems». ACM Transactions on Information Systems, Vol 22, 5–53. ACM Press, 2004. *http://web.engr.oregonstate.edu/~herlock/papers/eval_tois.pdf*.
- James, W. «The Principles of Psychology». Henry Holt and Company, 1918.

- Krause, E. F. «Taxicab Geometry: An Adventure in Non-Euclidean Geometry». Dover Publications, Inc. 1986.
- Linden, G., B. Smith, and J. York. «Amazon.com recommendations: Item-to-item collaborative filtering». IEEE Internet Computing, January–February 2003, pp. 76–80.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. «Numerical Recipes: The Art of Scientific Computing» (3rd Edition). Cambridge University Press, 1997.
- Spertus E., Sahami M., Buyukkokten O. «Evaluating Similarity Measures: A Large-Scale in the Orkut Social». Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2005), 2005.

4

Кластеризация: объединение в группы

- Необходимость и значение кластеризации
- Обнаружение групп пользователей на типичном веб-сайте, а также выявление групп похожих новостных сообщений, записей в блогах или документов
- Алгоритмы кластеризации на основе связей и молниеносный алгоритм k-средних

Присущая нам, людям, способность накапливать и сохранять информацию в значительной степени опирается на нашу же способность систематизировать данные, которые мы в изобилии получаем через такие механизмы, как чувственное восприятие, разум, язык и эмоции. Без тех или иных ссылочных структур невозможно было бы справиться с переизбытком доступной информации. Умственные образы, которые упорядочивают все получаемые нами данные, помогают удерживать в памяти квинтэссенцию этих данных и осмысливать окружающую действительность.

Как правило, мы организуем наши представления в группы или категории. Интеллектуальные приложения следуют тем же принципам и добиваются таких же результатов с помощью двух обширных категорий алгоритмов – *кластеризации* (clustering) и *классификации* (classification). В этой главе рассматриваются алгоритмы кластеризации; в следующей – алгоритмы классификации.

Вообще говоря, термин «кластеризация» относится к процессу объединения похожих вещей в группы. Допустим, в вашей базе данных есть ряд записей с информацией о книгах. Например, пусть в этой базе дан-

ных для каждой книги хранятся ее идентификатор (ID), заглавие, номер ISBN, внешний ключ, ссылающийся на таблицу с данными об авторах (скажем, `author_ID`), а также другие уместные в данном случае поля. Выполнив команду `SELECT` языка запросов SQL с предложением `ORDER BY author_ID`, вы получите список книг, упорядоченный по идентификатору автора. Перебирая записи в этом списке, вы начнете с книг первого автора, за которыми последуют книги второго автора, и так далее. В сущности, книги в списке объединены в группы по их авторам. В контексте кластеризации такие группы книг называются *кластерами книг* (book clusters), а то, что мы только что описали, – это логически простой, но ограниченный алгоритм кластеризации ваших книг.

Кластеризация полезна во многих ситуациях, но не всегда ее можно выполнить посредством простых SQL-запросов. Элементы, используемые для идентификации искомых групп, часто не являются уникальными, и поэтому требуются методы, приспособленные для работы со случайными данными.

В предшествующих главах мы видели, что для самых разнообразных объектов можно определить понятие расстояния и связанное с ним понятие сходства. Наша способность определять расстояние между двумя произвольными объектами пригодится и в этой главе, поскольку любые два объекта будут принадлежать одному и тому же кластеру только в том случае, если они в достаточной мере «близки» друг к другу и достаточно «удалены» от членов других кластеров.

Начнем с примера, который иллюстрирует некоторые предпосылки для применения кластеризации. Поскольку тема кластеризации обширна и у нас нет возможности раскрыть ее во всей полноте, мы предлагаем обзор алгоритмов кластеризации в соответствии со структурой кластеров, типом и размером кластерных данных. В оставшейся части этой главы будут подробно рассмотрены несколько конкретных алгоритмов. Кроме того, один раздел мы посвятим более сложным вопросам кластеризации, таким как вычислительная сложность алгоритмов кластеризации, а также рассмотрим проблему большой размерности.

4.1. Необходимость кластеризации

В этом разделе демонстрируется такой распространенный случай применения кластеризации, как идентификация групп пользователей в веб-приложении. Такие группы помогли бы нацеливать рекламу на конкретную аудиторию, улучшать взаимодействие с отдельными пользователями (для каждого пользователя можно отображать сообщения от пользователей с похожими взглядами, облегчая создание социальной сети на вашем сайте) и так далее. Задача идентификации групп пользователей по природе своей идеально подходит для того, чтобы решать ее с применением методов кластеризации.

Наша цель – показать, что даже если бы вы понятия не имели о том, что такое кластеризация, вам пришлось бы ее изобрести, чтобы добиться удовлетворительного решения названной и подобных ей проблем. Другими словами, мы хотим предоставить вам набор простых способов, к которым вы сможете прибегнуть для решения таких задач, даже если прежде ничего не знали о кластеризации. Мы рассматриваем кластеризацию как общий случай сортировки записей с несколькими атрибутами, а также как меру упорядочения по этим атрибутам.

Для начала мы покажем, что прямолинейный подход на основе предложений языка запросов SQL приемлем лишь в нескольких случаях, и объясним, почему в общем случае решение с применением стандартных SQL-запросов является неполным и нецелесообразным. Мы вновь обращаемся к сортировке, чтобы показать, что хотя нам удалось добиться универсальности в смысле применения произвольной системы показателей упорядочения, мы по-прежнему не в состоянии эффективно обработать случаи с несколькими атрибутами. Следовательно, потребуются универсальные методы кластеризации.

4.1.1. Группы пользователей на веб-сайте (конкретный случай)

Сейчас мы представим простую ситуацию для анализа, которой будем пользоваться в этом разделе, чтобы проиллюстрировать кластеризацию. Будем исходить из того, что мы работаем для крупной платформы, такой как SourceForge.net, обеспечивающей нужды сообщества разработчиков ПО с открытым исходным кодом, и нам требуется выяснить, почему люди участвуют в подобных проектах. Группы пользователей можно было бы идентифицировать по данным пользовательских профилей, выполнив их кластерный анализ. Кроме имени пользователя (`Name`), в интересах нашего обсуждения будем учитывать следующие атрибуты:

- `Age` – *возраст* пользователей, который мы будем измерять числом прожитых лет.
- `IncomeRange` – *годовой доход* пользователей, который определяется с помощью «вилок», или диапазонов: например, доход 65 000–80 000 долл. попадает в диапазон 0, доход 80 000–95 000 долл. соответствует диапазону 1 и так далее. Все сведения о диапазонах годового дохода и их границах можно найти в файле *README* в папке *data/ch04*.
- `Edu` – *образовательный уровень* пользователей: средняя школа, колледж, аспирантура и так далее.
- `Skills` – оценка пользователями их участия в проектах с точки зрения возможности повышения *профессионального мастерства*; скажем, по пятибалльной шкале, от 1 до 5.
- `Social` – оценка пользователями их участия в проектах с точки зрения возможности установить *социальные связи с* теми, с кем есть

общие интересы; опять-таки, эту оценку можно измерять по пятибалльной шкале, от 1 до 5.

- `isPaid` – индикатор *оплачиваемого участия*, то есть получает ли отдельный человек плату за свое участие в проекте. Можно использовать для этого булевскую переменную или обеспечить более детальное представление об оплачиваемом участии, зафиксировав долю времени (в процентах), оплачиваемого третьей стороной, в суммарном времени, затраченном на работу в проекте.

Этот пример можно распространить на любое веб-приложение, располагающее структурой социальной сети, введя атрибуты, более уместные в вашем случае. Чтобы наш пример был более конкретным, мы смоделировали данные, представленные в табл. 4.1. Заголовки столбцов этой таблицы содержат только что перечисленные атрибуты. В каждой строке находятся значения атрибутов для каждого из 20 учитываемых нами пользователей.

Таблица 4.1. Моделированные данные для кластерного анализа пользователей —членов интернет-сообщества

Name	Age	IncomeRange	Edu	Skills	Social	isPaid
Albert	23	0	0	3	3	0
Alexandra	25	1	2	4	2	0
Athena	24	0	1	3	4	0
Aurora	23	1	2	5	2	0
Babis	21	0	0	3	4	0
Bill	31	1	2	4	2	0
Bob	32	1	1	3	1	1
Carl	30	0	2	4	2	0
Catherine	31	1	1	3	3	0
Charlie	30	1	2	3	2	0
Constantine	37	1	1	3	2	0
Dmitry	35	2	2	1	1	1
Elena	38	1	1	3	2	0
Eric	37	2	2	2	2	0
Frank	39	3	1	3	1	1
George	42	2	2	2	1	1
Jack	43	3	1	1	1	1
John	45	4	2	1	1	1

Name	Age	IncomeRange	Edu	Skills	Social	isPaid
Maria	43	2	1	3	1	0
Lukas	45	3	2	1	1	1

Наша цель проста: определить, если это возможно, группы из отдельных пользователей, принимающих участие в проектах с открытым исходным кодом, на основании значений их атрибутов. В двух следующих разделах мы представим два наивных подхода, в порядке возрастания их сложности и результативности, которые могут помочь нам добиться поставленной цели.

4.1.2. Нахождение групп с помощью SQL–предложения order by

Для достижения нашей цели проще всего было бы загрузить данные в таблицу – если они еще не хранятся в таблице базы данных, – и написать SQL-запрос для нахождения интересующих нас возможных групп пользователей (кластеров). Мы загрузили данные в СУБД MySQL, но вы можете использовать для воспроизведения полученных нами результатов любую базу данных по своему выбору; файл *README.txt* в папке *data/ch04* содержит предложения языка запросов SQL, обеспечивающие загрузку данных в СУБД MySQL.

На рис. 4.1 показаны результаты выполнения следующего запроса: `select * from sf_users order by IncomeRange, Education;`. Как видите, в случае одного атрибута стандартный SQL-запрос работает прекрасно. Мы легко можем идентифицировать пять групп по атрибуту `Income-Range` и получили следующие кластеры: (Albert, Babis, Athena, Bill, Carl), (Elena, Constantine, Catherine, Bob, Charlie, Aurora, Alexandra), (Maria, Dmitry, Eric, George) и так далее. Аналогичные результаты можно получить для любого другого атрибута. Но обратите внимание: как только мы добавляем в предложение `order by` еще какие-то атрибуты, нам уже не удается легко идентифицировать другие группы. Результаты запроса определяет первый атрибут, а дополнительно указанные атрибуты ведут к дальнейшей сегментации тех кластеров, которые обнаружены для атрибутов, указанных первыми по порядку их перечисления в SQL-предложении.

Если исходить из предположения о возможности идентификации подходящих кластеров исключительно путем наглядной проверки, необходимо ответить на следующий вопрос: в каком порядке следует указывать атрибуты, чтобы можно было выявить подходящие кластеры? На этот вопрос нет простого ответа. Что если данные содержат тысячи записей? И что произойдет, если надо учесть не два-три, а десять и больше атрибутов? В таких случаях, если только мы априори не располага-

```
mysql> select * from sf_users order by IncomeRange, Education;
+-------------+-----+--------+-----+--------+--------+--------+
| Name        | Age | Income | Edu | Skills | Social | isPaid |
|             |     | Range  |     |        |        |        |
+-------------+-----+--------+-----+--------+--------+--------+
| Albert      |  23 |    0   |  0  |    3   |   3    |   0    |
| Babis       |  21 |    0   |  0  |    3   |   4    |   0    |
| Athena      |  24 |    0   |  1  |    3   |   4    |   0    |
| Carl        |  30 |    0   |  2  |    4   |   2    |   0    |
| Elena       |  38 |    1   |  1  |    3   |   2    |   0    |
| Constantine |  37 |    1   |  1  |    3   |   2    |   0    |
| Catherine   |  31 |    1   |  1  |    3   |   3    |   0    |
| Bob         |  32 |    1   |  1  |    3   |   1    |   1    |
| Bill        |  31 |    1   |  2  |    4   |   2    |   0    |
| Charlie     |  30 |    1   |  2  |    3   |   2    |   0    |
| Aurora      |  23 |    1   |  2  |    5   |   2    |   0    |
| Alexandra   |  25 |    1   |  2  |    4   |   2    |   0    |
| Maria       |  43 |    2   |  1  |    3   |   1    |   0    |
| Dmitry      |  35 |    2   |  2  |    1   |   1    |   1    |
| George      |  42 |    2   |  2  |    2   |   1    |   1    |
| Eric        |  37 |    2   |  2  |    2   |   2    |   0    |
| Frank       |  39 |    3   |  1  |    3   |   1    |   1    |
| Jack        |  43 |    3   |  1  |    1   |   1    |   1    |
| Lukas       |  45 |    3   |  2  |    1   |   1    |   1    |
| John        |  45 |    4   |  2  |    1   |   1    |   1    |
+-------------+-----+--------+-----+--------+--------+--------+
20 rows in set (0.03 sec)
```

Рис. 4.1. *Использование SQL-запросов для идентификации кластеров*

ем сведениями о данных, задача окажется трудной для решения, если вообще решаемой. Поразмыслив над этим, вы осознаете, что на одних SQL-запросах тут далеко не уедешь.

Фундаментальная проблема подхода с применением языка запросов SQL заключается в том, что обнаружение кластеров с трудом поддается автоматизации, и реализовывать этот подход имеет смысл не более, чем для пары атрибутов. Идентификация кластеров облегчается в случае перечислимых данных, но усложняется для непрерывных величин, и эта задача оказывается почти не решаемой для необработанных текстовых данных. Более того, если использовать не один, а несколько атрибутов, идентификация групп затрудняется, поскольку получаемые результаты будут существенно различаться в зависимости от того, как упорядочены атрибуты в запросе. С точки зрения кластеризации, возможности стандартного подхода с применением языка запросов SQL довольно ограниченны.

Тем не менее объединение возможностей SQL-запросов с более развитыми алгоритмами может привести к жизнеспособным реализациям решений кластеризации, поскольку язык SQL позволяет эффективно выполнить ряд операций с большими наборами данных. См. описание и ссылки, относящиеся к алгоритму SQLEM, в разделе «Сделать».

4.1.3. Нахождение групп путем сортировки массива

Вы, может быть, подумали, что проблемы SQL-подхода могут исчезнуть, если мы в своем Java-коде загрузим данные и применим пользовательский компаратор для целенаправленного упорядочения исходных многомерных данных. Если данные находятся в некоторого рода массиве, оснащенном таким пользовательским компаратором, то у нас появится возможность отсортировать эти данные и выявить все кластеры, которые могут быть в них представлены, верно? Давайте сделаем это и посмотрим, что происходит. На рис. 4.2 показано, что получилось в результате пользовательской сортировки массива.

На вид – отличные результаты, но определение границ между кластерами остается упражнением для пользователя алгоритма. Мы могли бы добавить несколько строк программного кода, чтобы создать по одно-

```
bsh % SortedArrayClustering.cluster(ds.getData());
John       ([45.0, 4.0, 2.0, 1.0, 1.0, 1.0])
Lukas      ([45.0, 3.0, 2.0, 1.0, 1.0, 1.0])
Maria      ([43.0, 2.0, 1.0, 3.0, 1.0, 0.0])
Jack       ([43.0, 3.0, 1.0, 1.0, 1.0, 1.0])
George     ([42.0, 2.0, 2.0, 2.0, 1.0, 1.0])
Frank      ([39.0, 3.0, 1.0, 3.0, 1.0, 1.0])
Elena      ([38.0, 1.0, 1.0, 3.0, 2.0, 0.0])
Eric       ([37.0, 2.0, 2.0, 2.0, 2.0, 0.0])
Constantine([37.0, 1.0, 1.0, 3.0, 2.0, 0.0])
Dmitry     ([35.0, 2.0, 2.0, 1.0, 1.0, 1.0])
Bob        ([32.0, 1.0, 1.0, 3.0, 1.0, 1.0])
Bill       ([31.0, 1.0, 2.0, 4.0, 2.0, 0.0])
Catherine  ([31.0, 1.0, 1.0, 3.0, 3.0, 0.0])
Carl       ([30.0, 0.0, 2.0, 4.0, 2.0, 0.0])
Charlie    ([30.0, 1.0, 2.0, 3.0, 2.0, 0.0])
Alexandra  ([25.0, 1.0, 2.0, 4.0, 2.0, 0.0])
Athena     ([24.0, 0.0, 1.0, 3.0, 4.0, 0.0])
Aurora     ([23.0, 1.0, 2.0, 5.0, 2.0, 0.0])
Albert     ([23.0, 0.0, 0.0, 3.0, 3.0, 0.0])
Babis      ([21.0, 0.0, 0.0, 3.0, 4.0, 0.0])
```

Рис. 4.2. *Кластеризация данных путем сортировки элементов массива с помощью пользовательского класса Comparator*

му кластеру для каждых четырех имен, но это был бы обман. Люди из одной возрастной группы порой походят друг на друга, но было бы опрометчиво считать эти результаты отражающими то, что будет происходить в общем случае. Листинг 4.1 содержит те две строки, которые позволяют создать выходные данные, показанные на рис. 4.2. Первая команда обеспечивает загрузку данных из табл. 4.1. Во второй команде для идентификации кластеров в загруженных данных используется класс `SortedArrayClustering`.

Листинг 4.1. Идентификация кластеров путем сортировки массива DataPoints

```
SFDataset ds = SFData.createDataset();

SortedArrayClustering.cluster(ds.getData());
```

В листинге 4.2 представлено содержимое класса `SortedArrayClustering`. В принципе, оно похоже на подход с применением SQL-предложений, поскольку единственное, что мы делаем, – это сортируем данные и выводим их на экран. Вот только ответственность за упорядочение множества точек мы переложили с SQL-предложения `order by` на наше пользовательское определение класса `Comparator`. Итак, есть нечто фундаментальное, чего лишены оба этих подхода.

Сортировка является эффективной и уместной методикой для кластеризации, если мы имеем дело с одним измерением. Многомерная природа наших данных была скрыта обращением к методу `getR()`. Этот метод обеспечивает вычисление расстояния каждого элемента до источника значений всех атрибутов. Считайте, что расстояние – это стрелка, направленная из центра нашей системы координат (где значения всех атрибутов равны нулю) к каждой точке данных. Само значение расстояния получается в результате использования класса `EuclideanDistance`, который, как предполагает его имя, и обеспечивает реализацию вычислений для получения евклидова расстояния, введенного в главе 3.

Листинг 4.2. Класс SortedArrayClustering: сортировка массива точек данных и их печать

```
public class SortedArrayClustering {

  public static void cluster(DataPoint[] points) {

    Arrays.sort(points, new Comparator<DataPoint>() {   ← Сортировка в порядке
                                                          убывания
    public int compare(DataPoint p1, DataPoint p2) {

          int result = 0;
      // Сортировка по значению оценки
      if (p1.getR() < p2.getR()) {
          result = 1;
      } else if (p1.getR() > p2.getR()) {
```

```
            result = -1;
        } else {
            result = 0;
        }
        return result;
    }
    });

    for (int i=0; i < points.length; i++) {
      System.out.println(points[i].toString());
    }
  }
}
```

Поскольку атрибуты «улетучились» и у нас осталась лишь содержащая все элементы строка, мы должны разобраться с двумя главными вопросами. Во-первых, по-прежнему надо решить, сколько кластеров существует и что они собой представляют. Во-вторых, из-за отсутствия нормализации данных, при вычислении евклидова расстояния значение атрибута `age` доминирует над значениями всех остальных атрибутов. Этот нежелательный эффект можно смягчить путем нормализации значений атрибутов в наборе данных, но для произвольного набора данных сделать это было бы нелегко.

Мы рассматриваем алгоритмы кластеризации, которые можно было бы охарактеризовать как интеллектуальные. Если исходить из наличия интеллекта у людей, что подумал бы человек, посмотрев на этот набор данных? Какие кластеры идентифицировал бы в нем человек? В частности, давайте ограничимся группой людей старше 40 лет. Пользователям George, Jack, John, Maria и Lukas перевалило за 40, и значения большей части приписываемых им атрибутов идентичны или очень близки друг к другу. Но все эти люди, за исключением пользователя Maria, получают плату за участие в проектах с открытым исходным кодом, а главным мотивом участия пользователя Maria в таких проектах, по-видимому, является ее желание повысить свое профессиональное мастерство и, следовательно, доход. Пользователь Maria, скорее всего, не должен быть включен в один кластер с пользователями George, Jack, John и Lukas, но с нашей сортировкой так не получится!

Можно было бы вернуться и проделать какие-то манипуляции с использованным нами расстоянием, с тем чтобы добиться желаемого исключения пользователя Maria из этого кластера. Если вы поступите таким образом, то лишите рассматриваемый подход его простоты и, по всей вероятности, не сможете успешно использовать этот алгоритм для другого набора данных.

Принятие решения относительно того, какие кластеры являются «правильными» для данного набора данных, – серьезная проблема, в области которой ведутся активные научные изыскания. Два «наивных» подхода – применение SQL-запросов и сортировка массива, – так же как

и все те алгоритмы, которые мы представим в последующих разделах, имеют свои преимущества и недостатки. В конечном счете, успешность применения того или иного алгоритма определяется природой обрабатываемых данных.

Этот вывод относится не только к кластеризации, и проектируя интеллектуальные приложения, вы всегда должны о нем помнить. Кластеризация представляет собой труднейший случай из-за отсутствия явного показателя успеха, которым мог бы воспользоваться компьютер. Кластеры заранее не известны, а если бы были известны, то вам не нужен был бы алгоритм кластеризации! Именно поэтому кластеризация принадлежит к категории машинного обучения, называемой *обучением без учителя* (unsupervised learning).

В следующем разделе приведен обзор алгоритмов кластеризации и их категоризация по ряду критериев, таких как получаемая в результате структура кластеров, структура и тип обрабатываемых данных, а также размер данных, которые необходимо подвергнуть кластеризации.

4.2. Обзор алгоритмов кластеризации

Мы представили подходы, использующие возможности языка запросов SQL и сортировку массивов, в качестве вступления к теме кластеризации, и таким образом, вы можете рассматривать кластеризацию как обобщение сортировки. Так и есть! Сортируя список объектов, мы, по сути, выстраиваем все объекты в один ряд и с помощью компаратора выбраем в этом ряду первый объект, второй и так далее. В результате этого процесса для каждого объекта ряда мы идентифицируем его ближайших соседей.

Выполняя кластеризацию набора объектов, мы также идентифицируем ближайшее окружение объекта, но объекты при этом могут сохранять свою многомерную природу. Объектами могли бы быть точки на плоскости или в трехмерном пространстве, или точки более общего геометрического построения, в зависимости от числа учитываемых атрибутов и от понятия расстояния, которым мы хотим воспользоваться.

Задача алгоритмов кластеризации – идентифицировать группы способом, свободным от недостатков SQL-подхода или подхода с использованием простой сортировки массива, который, следовательно, можно распространить на несколько измерений и на пространство произвольных объектов. Члены кластера должны быть очень похожи друг на друга (на своих соседей) и иметь существенные отличия от членов любого другого кластера этого исходного набора. Кластеризация применима к широкому спектру задач, который простирается от биологии и медицины до финансов и маркетинга.

Алгоритмы кластеризации выступают в самых разных видах и формах, и классифицировать их по какому-то одному критерию трудно. Поэто-

му перед тем, как перейти к конкретным реализациям и отвлечься от общей картины, мы приведем общий обзор алгоритмов кластеризации. Начнем с классификации алгоритмов кластеризации на основе структуры кластеров. Ищет ли алгоритм иерархические связи между точками или просто делит области пространства на разные группы? Второй вид классификации алгоритмов кластеризации – по типу и структуре данных. Одни алгоритмы кластеризации лучше работают с числовыми данными, другие специализируются на категорийных данных. Для классификации третьего вида главное – создавался ли алгоритм в расчете на обработку больших наборов данных. А теперь перейдем к обзору алгоритмов кластеризации с этих точек зрения.

4.2.1. Классификация алгоритмов кластеризации по структуре кластеров

На рис. 4.3 отображены категории классификации различных алгоритмов кластеризации по структуре тех кластеров, которые являются результатом работы этих алгоритмов.

Рис. 4.3. *Классификация алгоритмов кластеризации по структуре кластеров*

Результатом применения иерархических алгоритмов является определение кластеров внутри кластеров. Применение иерархического алгоритма для кластеризации новостных статей могло бы обеспечить получение четырех больших групп, представляющих основные темы, такие как политика, спорт, бизнес и технологии, а также подгрупп этих основных групп: например, в группе спортивных новостей можно выделить новости баскетбола, бейсбола и так далее. В примере, который мы рассматриваем в этом разделе, иерархический алгоритм может разделить пользователей проектов с открытым исходным кодом на две большие группы: те, кто получает плату за участие, и те, кто участвует в проектах бесплатно. Две эти большие группы можно дополнительно разделить по возрасту или по диапазону годового дохода.

Значительная часть алгоритмов иерархической кластеризации допускает применение *параметра сдвига* (threshold parameter), задающего глубину, на которой алгоритм должен прекратить поиск более мелких подгрупп. Хотя по логике следовало бы задавать это значение на основе окончательной структуры кластеров, такие параметры сокращают

количество лишних вычислений. Разумеется, заранее итоговое число кластеров неизвестно, но оно зависит от конфигурационных настроек алгоритма, определяющих критерий прекращения действий по формированию иерархии кластеров.

Алгоритмы из категории *агломеративно-иерархических* (agglomerative hierarchical) являются «восходящими»: они начинают с отдельных элементов и формируют кластеры, объединяя их с другими элементами, действуя снизу вверх, по направлению к глобальному (супер)кластеру. Категория *дивизимно-иерархических* (divisive hierarchical) алгоритмов придерживается «нисходящего» принципа: эти алгоритмы начинают с глобального (супер)кластера и «спускаются» вниз, разделяя данные на более мелкие кластеры.

Плоские (partitional) алгоритмы создают фиксированное число кластеров. К этой категории принадлежит так называемый алгоритм кластеризации k-средних (k-means); мы используем его позже в этой главе. Алгоритмы *минимального остовного дерева* (minimum spanning tree, MST) и *ближайших соседей* (nearest neighbor) также имеют «плоские» версии. В рамках этой категории применяются два основных подхода: *концептуальное моделирование* (conceptual modeling) и *итеративная оптимизация*. Типичные представители первого подхода – алгоритмы на основе вероятностных моделей, типичный образец второго – алгоритм k-средних.

4.2.2. Классификация алгоритмов кластеризации по типу и структуре данных

На рис. 4.4 мы представили классификацию алгоритмов кластеризации по типу и по структуре данных. Если вы имеете дело только с числовыми данными – например, с географическими координатами на карте или с историческими данными о биржевом курсе, – наиболее подходящими для вашей работы могут оказаться алгоритмы на основе решеток. Алгоритмы из этой категории, основанные на *спектральных* (spectral) и *волновых* (wavelet) методах, обеспечивают значительные

Рис. 4.4. *Классификация алгоритмов кластеризации по типу и структуре данных*

преимущества. Результат работы алгоритма WaveCluster, предложенного в [Sheikholeslami, G., S. Chatterjee, and A. Zhang], – высококачественные кластеры, полученные в результате вычислений минимальной сложности.

Еще одна категория алгоритмов кластеризации специализируется на обработке категорийных данных. Главное отличие этих алгоритмов в том, что они используют показатели, основанные на принадлежности множеству, такие как коэффициент Жаккарда. Как правило, категорийные данные нуждаются в упорядочении, но нередко для них трудно найти подходящее числовое представление. Как можно представить с помощью чисел список имен? Какой бы способ вы ни предложили, он будет зависеть от контекста, а не от магического алгоритма, успешного во всех случаях. Если речь идет об именах людей, то лексикографическое упорядочение вполне приемлемо, но в случае названий юридических лиц этот способ может привести к смешению фирм, никак не связанных друг с другом. Таким образом, многие алгоритмы кластеризации, которые отлично работают с данными, определяемыми как числовые, не в состоянии справиться с категорийными данными.

Чтобы сделать эту мысль более понятной, вернемся к подходу, использованному нами в разделе 2.5. Там мы ранжировали несколько новостных статей, используя не гиперссылки между этими статьями, а набор характерных для данной статьи слов. Естественное представление этих данных было категорийным (не числовым), и мы использовали число общих термов как показатель степени связанности двух документов. Один из алгоритмов кластеризации, которые мы рассмотрим в этой главе, похож на методику из раздела 2.5 и хорошо работает с категорийными данными. Это алгоритм *ROCK* (Robust Clustering Using Links), принадлежащий к категории агломеративно-иерархических алгоритмов, – в нем для определения понятия соседства в случае новостных статей применяется показатель сходства Жаккарда (см. раздел 3.1.3).

Алгоритмы фильтрующей кластеризации применяются, если кластеры должны удовлетворять определенным ограничениям. Типичный случай здесь – точки кластеризации на двумерной поверхности, в присутствии помех (obstacles). Понятно, что кластеры, которые мы формируем, должны избегать помех. Иначе типичное евклидово расстояние будет работать плохо и для измерения расстояния между двумя точками потребуются более выразительные метрики. неплохой кандидат – длина кратчайшего пути между двумя точками; вычисление кратчайшего пути включает уклонение от помех. В [Tung, A. K. H., J. Hou, and J. Han] представлен алгоритм, который удовлетворительно справляется с этой проблемой. В нашей книге алгоритмы фильтрующей кластеризации не рассматриваются.

4.2.3. Классификация алгоритмов кластеризации по размеру обрабатываемых данных

На рис. 4.5 представлена категория алгоритмов кластеризации, предназначенных для больших наборов данных. Мы несколько иначе подходим к этой категории алгоритмов кластеризации. Пространственно-временная сложность многих алгоритмов кластеризации возрастает как квадрат числа точек данных, которые вы хотите подвергнуть кластеризации. Действуйте осторожно, иначе быстро исчерпаете имеющуюся оперативную память или будете вечно дожидаться завершения процесса кластеризации!

Поэтому [Bradley, P.S., U. Fayyad, and C. Reina] предложили концепцию, по которой алгоритм, предназначенный для интернет-приложений и имеющий дело с большими базами данных, должен обеспечивать выполнение следующих условий:

- По возможности, однократное сканирование базы данных
- Функционирование в онлайн-режиме по принципу «хороший ответ доступен в любое время»
- Возможность приостановить, остановить и продолжить его работу
- Возможность инкрементного обновления для учета новых данных
- Учет ограничений по ОЗУ, если они есть
- По возможности, поддержку разных режимов сканирования, например: последовательное сканирование, сканирование на основе индексов и выборочное сканирование
- Алгоритмы, работающие с однонаправленными курсорами, предпочтительнее тех, что работают с представлением базы данных, поскольку такое представление, как правило, – результат затратных по вычислениям операций слияния (joins)

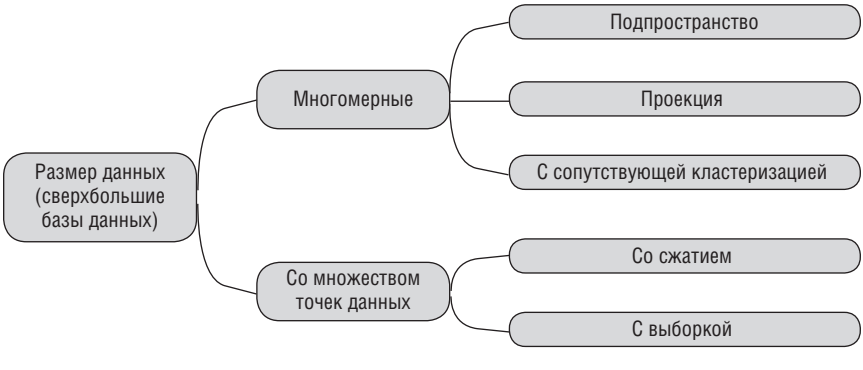

Рис. 4.5. *Классификация алгоритмов кластеризации по размеру обрабатываемых данных*

Разного рода алгоритмы, где по сути более ясные версии базовых алгоритмов смешиваются с эвристиками и другими методиками (такими как сжатие и дискретизация), то есть платящие сложностью за эффективность и производительность, – следствие соблюдения этих требований.

Нетрудно догадаться, что алгоритм может удовлетворять нескольким критериям, то есть один алгоритм может принадлежать нескольким категориям. Например, алгоритм *BIRCH* (balanced iterative reducing and clustering using hierarchies)[1] можно отнести и к алгоритмам кластеризации для сверхбольших баз данных (very large database, VLDB), и к иерархическим алгоритмам кластеризации.

Наш обзор несколько затянулся, но он показал, что идентификация групп похожих объектов, представлявшаяся довольно простой задачей, – интересная и весьма глубокая тема. Мы можем выбирать из множества алгоритмов кластеризации, и выбор зависит от многих факторов, таких как природа данных, требуемый тип результата и ограничения вычислений. В следующих разделах мы представим ряд алгоритмов кластеризации, охватив многое из того, что мы здесь обсудили, и подробно рассмотрим кластеризацию очень больших наборов данных. Что ж, засучим рукава – и за дело!

4.3. Алгоритмы связей

В этом разделе мы, также на основе данных, описанных в разделе 4.1, попытаемся выяснить, какого рода группы пользователей можно выявить в этом воображаемом хранилище открытого исходного кода. Начнем с описания такой структуры данных, как *дендрограмма* (dendrogram), которая полезна для кластеризации и задействована в программном коде по всей этой главе. Мы опишем идеи, на которых основаны алгоритмы связей, и подробно опишем три таких алгоритма. В частности, будут рассмотрены алгоритмы одной связи, средней связи и минимального остовного дерева.

4.3.1. Дендрограмма: базовая структура данных кластера

Реализация базовой структуры, которую мы будем использовать в процессе кластеризации, инкапсулирована в классе `Dendrogram`. Структура дендрограммы показана на рис. 4.6. Это древовидная структура данных[2], которая помогает отобразить иерархическую структуру класте-

---

[1] Сбалансированное итеративное сокращение и кластеризация с использованием иерархий. – *Прим. перев.*

[2] *«Дендро»* по-гречески означает «дерево». – *Прим. перев.*

ров. Можете считать, что дендрограмма – это набор упорядоченных триад: [*d*, *k*, {...}], где первый элемент – порог близости (*d*), второй элемент – число кластеров (*k*), а третий – набор кластеров.

На рис. 4.6 представлена дендрограмма с четырьмя уровнями; эту же дендрограмму можно представить как упорядоченное множество: *{[0, 5, {{A}, {B}, {C}, {D}, {E}}], [1, 3, {{A, B}, {C}, {D, E}}], [2, 2, {{A, B, C}, {D, E}}], [3, 1, {A, B, C, D, E}]}*. Таким образом, дендрограмма оснащена всем необходимым для отображения не одного, но множества кластеров. В свою очередь, дендрограмма позволяет нам отобразить формирование кластеров, поскольку они создаются объединением одиночных элементов в одну структуру. Все агломеративно-иерархические алгоритмы поступают так:

1. Определяют исходную дендрограмму, все элементы которой являются кластерами с одним элементом.
2. Скачкообразно наращивают пороговое значение для расстояния и решают, какие элементы должны образовать новые кластеры.
3. Берут все вновь созданные кластеры и добавляют в дендрограмму новый уровень.
4. Продолжают выполнять шаги 2 и 3 до тех пор, пока все элементы не будут принадлежать одному большому кластеру.

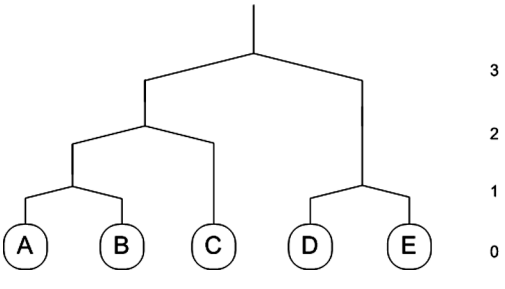

Рис. 4.6. *Визуализация иерархических кластеров: простая дендрограмма*

С точки зрения реализации, мы отобразили структуру дендрограммы с помощью двух связанных хеш-карт, как показано в листинге 4.3; в этом листинге не приводятся два вспомогательных метода, обеспечивающие печать результатов.

Листинг 4.3. Дендрограмма: главный класс, обеспечивающий инкапсуляцию реализации иерархических кластеров

```
public class Dendrogram {

  private Map<Integer, ClusterSet> entryMap;
  private Map<Integer, String> levelLabels;
```

```
  private Integer nextLevel;
  private String levelLabelName;

  public Dendrogram(String levelLabelName) {
    entryMap = new LinkedHashMap<Integer, ClusterSet>();
    levelLabels = new LinkedHashMap<Integer, String>();
    nextLevel = 1;
    this.levelLabelName = levelLabelName;
  }

  public int addLevel(String label, Cluster cluster) {
    List<Cluster> values = new ArrayList<Cluster>();
    values.add(cluster);
    return addLevel(label, values);
  }

  public int addLevel(String label, Collection<Cluster> clusters) {

    ClusterSet clusterSet = new ClusterSet();

    for(Cluster c : clusters) {
      // Перед добавлением копируем кластер - со временем
      // элементы кластера могут измениться, но мы хотим сохранить
      // текущее состояние дендрограммы.
      clusterSet.add(c.copy());
    }

    int level = nextLevel;

     entryMap.put(level, clusterSet);
     levelLabels.put(level, label);

     nextLevel++;
     return level;
   }

public void setLevel(int level, String label,
➥ Collection<Cluster> clusters) {

  ClusterSet clusterSet = new ClusterSet();

  for(Cluster c : clusters) {
     clusterSet.add(c.copy());
  }

  System.out.println("Setting cluster level: "+level);

  entryMap.put(level, clusterSet);
  levelLabels.put(level, label);

  if( level >= nextLevel ) {
    nextLevel = level + 1;
  }
}
```

Подводя итог сказанному, такая структура данных, как дендрограмма, может отобразить все возможные кластерные конфигурации набора данных, являются они иерархическими или нет. Эта структура данных предпочтительна для представления результатов кластеризации. А теперь познакомимся с семейством алгоритмов связей.

4.3.2. Знакомство с алгоритмами связей

В листинге 4.4 мы приводим сценарий для загрузки данных, подобных данным сайта SourceForge, и последовательной активизации алгоритмов.

Листинг 4.4. Агломеративно-иерархические алгоритмы кластеризации

```
SFDataset ds = SFData.createDataset();
DataPoint[] dps = ds.getData();                          Загрузка данных
double[][] adjMatrix = ds.getAdjacencyMatrix();

SingleLinkAlgorithm sla = new SingleLinkAlgorithm(dps,adjMatrix);   Кластеризация
Dendrogram dendroSLA = sla.cluster();                               одной связи
dendroSLA.print(4);

MSTSingleLinkAlgorithm sla2 =
➥ new MSTSingleLinkAlgorithm(dps,adjMatrix);            MST-кластеризация одной связи
Dendrogram dendroSLA2 = sla2.cluster();
dendroSLA2.print(4);

AverageLinkAlgorithm ala =
➥ new AverageLinkAlgorithm(dps,adjMatrix);              Кластеризация
Dendrogram dendroALA = ala.cluster();                    средней связи
dendroALA.print(4);
```

Класс `SFDataset` представляет наш набор данных из раздела 4.1. Три класса содержат реализации соответствующих алгоритмов в порядке их появления: `SingleLinkAlgorithm`, `MSTSingleLinkAlgorithm` и `AverageLinkAlgorithm`. Чтобы устранить эффекты, связанные с представлением данных и отсутствием нормализации, все алгоритмы используют в качестве отправной точки одинаковую информацию – исходные данные из табл. 4.1 (в виде массива `Datapoint[] dps`) и матрицу смежности (в виде массива `double[][] adjMatrix`), где зафиксирована относительная близость каждого пользователя с каждым другим пользователем в этом наборе данных.

Все наши алгоритмы связей инициализируют свои дендрограммы путем присваивания триады [0, N, {{$X1$}, {$X2$}, ..., {XN}}]. Если порог близо-

сти (первый элемент триады) равен 0, единственный элемент, который может быть близок к любому другому элементу, – это он сам, и следовательно, все элементы загружаются как отдельные кластеры.

Как мы говорили, все алгоритмы используют двумерный массив типа `double` для хранения *матрицы смежности* (adjacency matrix). Эта матрица содержит расстояния между любыми двумя элементами множества; можно считать матрицу близости аналогом матрицы сходства, которую мы использовали для пользователей и предметов в главе 3. Значения элементов матрицы смежности позволяют нам – с помощью порога близости – определить, как поступить с двумя элементами: объединить их и образовать новый кластер или оставить их разобщенными, в виде отдельных кластеров. Такие сравнения осуществляются многократно путем наращивания величины порога близости. После выполнения конечного числа шагов все элементы будут принадлежать одному кластеру, и алгоритм завершит работу.

Этот процесс изображен на рис. 4.7, где показаны три такие итерации; для наглядности мы ограничились двумя измерениями. В случае больших размерностей принцип остается прежним, просто пространства большой размерности труднее визуализировать. Черный кружок – это точка данных, которую мы хотим поместить в кластер, пунктирная окружность – порог близости для каждой точки данных на этом уровне итерации.

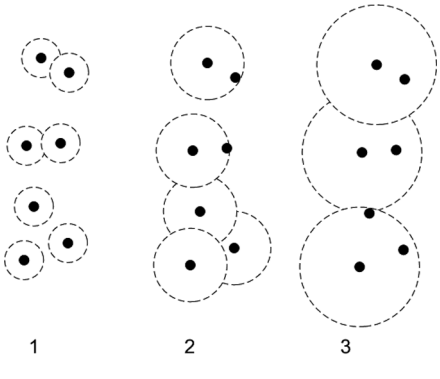

Рис. 4.7. *Алгоритм одной связи в действии (3 итерации)*

Первая итерация: каждая точка данных принадлежит собственному кластеру, всего 7 кластеров. Вторая итерация: образованы два кластера (вверху), всего 5 кластеров. Третья итерация: объединены три точки данных (внизу), всего 3 кластера. От итерации к итерации окружности растут, пока радиус первой окружности (порог близости) у первой точки

данных (в центре) не вырастет настолько, что эта окружность охватит весь набор данных. В этот момент итерации прекращаются.

Агломеративные алгоритмы отличаются двумя признаками:

- Применяемый ими подход к объединению кластеров на каждом шаге итерации
- Определение матрицы смежности

Алгоритмы одной связи, средней связи и минимального остовного дерева – это три популярных версии агломеративно-иерархической кластеризации, основанной на понятиях теории графов. В следующих трех разделах мы рассмотрим эти алгоритмы.

4.3.3. Алгоритм одной связи

Алгоритм одной связи (см. листинг 4.5) пытается найти наибольшее число связанных компонентов в графе. Этот алгоритм объединяет два кластера, если их соединяет хотя бы одно ребро; отсюда название – алгоритм *одной связи* (single link). Другими словами, если минимальное расстояние между любыми двумя точками не превышает значение *порога близости* (proximity threshold), то есть если точки данных находятся внутри пунктирной окружности, эти кластеры объединяются. Алгоритмически это показано в методах `cluster` и `buildClusters` в листинге 4.5.

Листинг 4.5. Объединение кластеров, имеющих хотя бы одну связь

```
public Dendrogram cluster() {
  Dendrogram dnd = new Dendrogram("Distance");
  double d = 0;

  List<Cluster> initialClusters = new ArrayList<Cluster>();
  for(DataPoint e : elements) {  ❶
    Cluster c = new Cluster(e);
    initialClusters.add(c);
  }

  dnd.addLevel(String.valueOf(d), initialClusters);
  d = 1.0;
  int k = initialClusters.size();

  while( k > 1 ) {  ❷
    int oldK = k;
    List<Cluster> clusters = buildClusters(d);
    k = clusters.size();
    if( oldK != k ) {
      dnd.addLevel(String.valueOf(d), clusters);
    }
    d = d + 1;
```

```
    }
    return dnd;
  }

  private List<Cluster> buildClusters(double distanceThreshold) {

    boolean[] usedElementFlags = new boolean[elements.length];

    List<Cluster> clusters = new ArrayList<Cluster>();

    for(int i = 0, n = a.length; i < n; i++) {

      List<DataPoint> clusterPoints = new ArrayList<DataPoint>();

      for(int j = i, k = a.length; j < k; j++) {    ❸

        if( a[i][j] <= distanceThreshold && usedElementFlags[j] == false ) {
          clusterPoints.add(elements[j]);
          usedElementFlags[j] = true;
        }
      }

      if( clusterPoints.size() > 0 ) {
        Cluster c = new Cluster(clusterPoints);
        clusters.add(c);
      }
    }
    return clusters;
  }
```

Изначально мы загружаем каждую точку данных в ее собственный кластер ❶. Итерации повторяются до тех пор, пока не получится единственный кластер, содержащий все точки данных ❷. На каждой итерации в методе `buildClusters` выполняется кластеризация и значение порога расстояния увеличивается на единицу.

Обратите внимание: хотя мы воспользовались симметрией матрицы смежности (второй цикл начинается с индекса `i`, а не с нуля) ❸, этот алгоритм требует выполнения нескольких операций, число которых возрастает как квадрат числа элементов, которые мы хотим объединить в кластеры. Будем говорить, что вычислительная сложность алгоритма, в пространстве и во времени, равна $O(N^2)$. Для небольших наборов данных это не имеет значения, но такая функциональная зависимость оказывается гибельной, когда мы подвергаем кластеризации реальные наборы данных. Подробнее мы будем говорить об этих аспектах кластеризации в реальных условиях в разделе 4.4. На рис. 4.8 показаны результаты, полученные для алгоритма одной связи, когда мы использовали примерный набор данных из табл. 4.1; вывод данных настроен так, что на печать выводятся кластеры уровня 4.

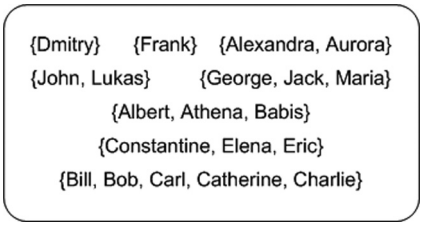

Рис. 4.8. *Результаты кластеризации, полученные с использованием алгоритма одной связи для уровня 4*

4.3.4. Алгоритм средней связи

Алгоритм средней связи, показанный в листинге 4.6, подобен алгоритму одной связи, но объединяет два кластера по другому условию. В частности, проверяется, не превышает ли *среднее расстояние* (average distance) между любыми двумя точками в двух целевых кластерах пороговое значение близости. Заметьте, что в этом алгоритме мы наращиваем порог близости на половину единицы (0,5), а не на целую единицу. Это приращение носит произвольный характер; вы можете варьировать его значение и наблюдать, как это сказывается на результатах.

Листинг 4.6. Объединение кластеров на основе среднего расстояния

```
public Dendrogram cluster() {

  Dendrogram dnd = new Dendrogram("Distance");   ← Инициализация
  double d = 0.0;

  for(DataPoint e : elements) {
     Cluster c = new Cluster(e);
     allClusters.add(c);
  }

  dnd.addLevel(String.valueOf(d), allClusters.getAllClusters());
  d = 1.0;

  while( allClusters.size() > 1 ) {   ← Цикл верхнего уровня для построения иерархии

    int K = allClusters.size();

    mergeClusters(d);

    // Может быть, что для текущего d нет кластеров для объединения.

    if( K > allClusters.size() ) {
       dnd.addLevel(String.valueOf(d),
    ➥  allClusters.getAllClusters());
       K = allClusters.size();
    }
    d = d + 0.5;
```

```
  }
  return dnd;
}

private void mergeClusters(double distanceThreshold) {

  int nClusters = allClusters.size();

  ObjectToIndexMapping<Cluster> idxMapping =
➥new ObjectToIndexMapping<Cluster>();

  double[][] clusterDistances = new double[nClusters][nClusters];

  for(int i = 0, n = a.length; i < n; i++) {
```

Суммирование расстояния всех связей для всех кластеров

```
     for(int j = i + 1, k = a.length; j < k; j++) {

         double d = a[i][j];

         if( d > 0 ) {
           DataPoint e1 = elements[i];
           DataPoint e2 = elements[j];

            Cluster c1 = allClusters.findClusterByElement(e1);
            Cluster c2 = allClusters.findClusterByElement(e2);

            if( !c1.equals(c2) ) {
               int ci = idxMapping.getIndex(c1);
               int cj = idxMapping.getIndex(c2);

               clusterDistances[ci][cj] += d;
               clusterDistances[cj][ci] += d;
            }
          }
       }
    }
    boolean[] merged = new boolean[clusterDistances.length];

    for(int i = 0, n = clusterDistances.length; i < n; i++) {

       for(int j = i+1, k = clusterDistances.length; j < k; j++) {

          Cluster ci = idxMapping.getObject(i);
          Cluster cj = idxMapping.getObject(j);
          int ni = ci.size();
          int nj = cj.size();

          clusterDistances[i][j] =
➥         clusterDistances[i][j] / (ni * nj);
```

Усредненное расстояние между кластерами

```
          clusterDistances[j][i] = clusterDistances[i][j];

          // Объединение кластеров,
          // если расстояние не превышает порогового значения
          if( merged[i] == false && merged[j] == false ) {
```

```
                if( clusterDistances[i][j] <= distanceThreshold) {
                   allClusters.remove(ci);
                   allClusters.remove(cj);
                   Cluster mergedCluster = new Cluster(ci, cj);
                   allClusters.add(mergedCluster);
                   merged[i] = true;
                   merged[j] = true;
                }
             }
          }
       }
    }
```

Как и прежде, дендрограмма инициализируется путем размещения каждого элемента набора в собственном кластере, и новые кластеры формируются до тех пор, пока все элементы не будут принадлежать одному кластеру. В отличие от алгоритма одной связи, нам необходимо найти суммарное расстояние всех связей между двумя кластерами. Алгоритм средней связи требует дополнительных вычислений. В первом цикле, в методе `mergeClusters` в листинге 4.6 складываются расстояния между любыми двумя элементами набора, чтобы получить суммарное расстояние для кластеров, которым принадлежат эти элементы. Во втором цикле, в этом же методе, итоговая сумма делится на число связей, и полученное среднее расстояние сравнивается с пороговым значением. Если среднее расстояние не превышает порогового значения, кластеры объединяются. Завершив все объединения и сбор данных для текущего уровня, алгоритм продолжает работу на следующем уровне дендрограммы, точно так же, как это делал алгоритм одной связи.

На рис. 4.9 показаны результаты, которые мы получили для алгоритма средней связи, когда использовали пример набора данных из табл. 4.1; вывод данных настроен так, что на печать выводятся кластеры для уровня 4, как мы сделали это на рис. 4.8. Обратите внимание, сейчас было получено меньшее число кластеров. Как вы думаете, что произошло? Почему наблюдается такое явное различие в результатах? Даже несмотря на то, что уровень остается прежним, отличается способ, с помощью

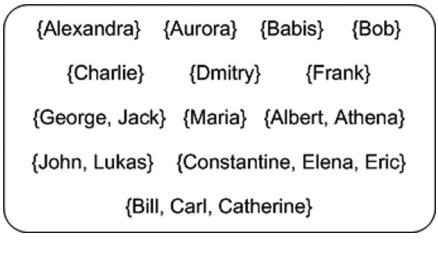

Рис. 4.9. *Результаты кластеризации, полученные с использованием алгоритма средней связи*

которого каждый алгоритм вычисляет расстояние. Соответственно, для одного и того же уровня используется разный порог близости. Другими словами, после четырех итераций алгоритм одной связи расширяет свои окружности близости (см. рис. 4.7) намного больше, чем это делает алгоритм средней связи. Поэтому, естественно, результаты кластеризации, полученные для алгоритма средней связи, демонстрируют меньшее число кластеров. Мораль истории такова: вы должны сравнивать алгоритмы такого типа по размеру окружностей близости, а не по уровню итерации.

4.3.5. Алгоритм минимального остовного дерева

Чтобы разобраться в третьем агломеративном алгоритме, нам надо будет поговорить о понятии *минимального остовного дерева* (minimum spanning tree, MST). Вообще говоря, для данного набора элементов можно построить дерево, соединяя любые две вершины с помощью только одного ребра (связи). *Остовное дерево* (spanning tree) соединяло бы все вершины данного набора, и понятно, что для этого есть много способов. Но в дереве MST вершины соединяются так, что для соединенных вершин *минимизируется сумма смежных значений*. Чтобы обеспечить получение такой матрицы смежности, в нашей реализации используется алгоритм Прима–Ярника (Prim–Jarnik) для идентификации минимального остовного дерева, и эта реализация включает $O(N^2)$ операций. Есть другие алгоритмы с почти линейной производительностью – $O(N \log(N))$. Если у вас возникло желание прочесть еще что-то об алгоритмах теории графов, относящихся к минимальному остовному дереву и расширению его возможностей, обратитесь к разделу «Сделать», а также к разделу ссылок.

MST-алгоритм одной связи, как предполагает его название, является вариантом алгоритма одной связи на основе минимального остовного дерева. Последнее получено из матрицы смежности и обеспечивает естественное упорядочение элементов набора. Если матрица смежности является массивом 5×5, дерево MST также представлено двумерным массивом 5×5, и кроме того, оно должно быть симметричным. Для обеих матриц мы присваиваем диагональным элементам значение –1, чтобы указать, что нас не интересуют случаи самосвязывания. Это единственное отличие MST-алгоритма от двух других агломеративных алгоритмов кластеризации, которые мы уже видели. Этот алгоритм использует данные из дерева MST, чтобы объединить кластеры на основе возрастания порядка следования их элементов в дереве.

Рассмотрим результаты, которые мы получаем при выполнении сценария из листинга 4.4. Алгоритм с одной связью на четвертом уровне формирует кластеры, которые были показаны на рис. 4.5. На рис. 4.10 показаны результаты, которые мы получили в результате применения MST-алгоритма с одной связью для примерного набора данных из табл. 4.1.

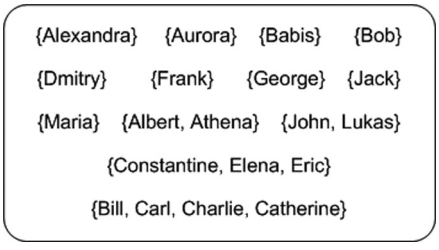

Рис. 4.10. *Кластеризация результатов, полученных с использованием MST-алгоритма одной связи*

Вывод данных настроен на печать кластеров для уровня 4, аналогично тому, как это делалось раньше для вывода результатов, показанных на рис. 4.8 и 4.9, для алгоритмов одной связи и средней связи, соответственно.

MST-алгоритм одной связи в результате формирует меньше кластеров, чем алгоритм одной связи, потому что, аналогично случаю алгоритма средней связи, окружности близости на четвертом уровне расширяются не так сильно, как в случае алгоритма одной связи. Если вы постепенно повышаете уровень, то можете наблюдать объединение различных одноточечных наборов – *синглтонов* (singletons) и кластеров в более крупные кластерные образования. Как и прежде, работа алгоритма завершается, когда все элементы набора данных оказываются принадлежащими одному кластеру. Итак, давайте рассмотрим программный код. В листинге 4.7 показан вспомогательный класс `MST`, предназначенный для создания минимального остовного дерева для данной матрицы смежности.

Листинг 4.7. Создание минимального остовного дерева на основе матрицы смежности

```
public class MST {

  public double[][] buildMST(double[][] adjM) {

    boolean[] allV = new boolean[a.length];
    allV[0] = true;

    double[][] mst = new double[adjM.length][adjM.length];
    for(int i = 0, n = mst.length; i < n; i++) {
       for(int j = 0; j < n; j++) {
           mst[i][j] = -1;
       }
    }

    Edge e = null;
    while( (e = findMinimumEdge(allV, adjM)) != null ) {
        allV[e.getJ()] = true;
```

Инициализация вектора для хранения узлов дерева MST

Инициализация матрицы MST

Повторение итерации до тех пор, пока не найдем минимум

```
            mst[e.getI()][e.getJ()] = e.getW();
            mst[e.getJ()][e.getI()] = e.getW();
        }
        return mst;
    }

    private Edge findMinimumEdge(boolean[] mstV, double[][] a) {
        Edge e = null;
        double minW = Double.POSITIVE_INFINITY;
        int minI = -1;
        int minJ = -1;

        for( int i = 0, n = a.length; i < n; i++ ) {
            if( mstV[i] == true ) {
                for(int j = 0, k = a.length; j < k; j++) {
                    if( mstV[j] == false ) {
                        if( minW > a[i][j]) {
                            minW = a[i][j];
                            minI = i;
                            minJ = j;
                        }
                    }
                }
            }
        }
        if( minI > -1 ) {
            e = new Edge(minI, minJ, minW);
        }
        return e;
    }
}
```

Чтобы сократить этот листинг, мы не включили в него внутренний класс с именем `Edge` , который является элементарным классом, инкапсулирующим реализацию вычислений ребер графа и их весов; подробности этой реализации ищите в полном исходном коде. Как видим, это простой алгоритм для нахождения минимального остовного дерева, и он известен как алгоритм Прима–Ярника (Prim–Jarnik). Этот алгоритм можно свести к следующим шагам:

1. Инициализируем вектор, который указывает, принадлежит ли элемент дереву MST (`allV`).
2. Инициализируем матрицу MST (переменная `mst`) некоторым принятым по умолчанию отрицательным значением (например, –1).
3. Начинаем с любого узла и находим ребро, которое исходит из этого узла и имеет минимальную длину в сравнении со всеми остальными ребрами, исходящими из этого узла.
4. Узел, находящийся на другом конце ребра с минимальной длиной, добавляется к узлам дерева MST.

5. Повторяем шаги 3 и 4 до тех пор, пока в дерево не будут включены все узлы; дерево должно перекрывать граф.

Другими словами, алгоритм Прима наращивает остовное дерево, начиная с произвольно выбранного узла, путем многократного добавления ребра с наименьшим весом, соединяющего узел, который уже является частью дерева MST, с узлом, который еще не принадлежит дереву MST, и этот процесс завершается, когда все узлы оказываются частью дерева MST. Дерево MST, полученное в результате одного выполнения алгоритма Прима, может отличаться от дерева MST, полученного в результате повторного выполнения этого алгоритма. Есть способ, позволяющий последовательно получать одно и то же дерево MST, независимо от того, какой узел был выбран в качестве исходного. Можете ли вы представить, при каких условиях это возможно? Конечно, этот алгоритм не является единственным алгоритмом, доступным для получения минимального остовного дерева. Хорошо известны еще два алгоритма – *алгоритм Крускала* (Kruskal's algorithm) и *алгоритм Борувки* (Boruvka's algorithm).

Временная сложность MST-алгоритма связи определяется как $O(N^2)$, поскольку это порядок числа вычислений, которые необходимо выполнить, чтобы получить дерево MST. Чтобы убедиться в этом, взгляните на метод `findMinimumEdge` и обратите внимание на двойной цикл размерности N. Это число операций мажорирует остальную часть алгоритма. Показатель временной сложности можно улучшить с помощью хеш-таблицы и сортировки наименьших ребер для каждого из узлов, которые мы уже проанализировали.

И наконец, мы должны упомянуть, что все алгоритмы одной связи пользуются дурной славой из-за так называемого *цепочечного эффекта* (chain effect), который может привести к объединению двух кластеров только потому, что в них случайно оказались две точки, близкие друг к другу, тогда как большинство остальных точек этих кластеров находятся на большом расстоянии друг от друга. Алгоритмы одной связи не имеют «лекарства» от этой проблемы, но остальные алгоритмы, которые мы будем рассматривать, не страдают этим недостатком.

4.4. Алгоритм k-средних

Все три рассмотренные в предыдущем разделе алгоритма связей относились к агломеративно-иерархическим алгоритмам кластеризации. Алгоритм k-средних (k-means) – это первый плоский алгоритм, который мы исследуем, и надо сказать, что он находит самое широкое применение на практике благодаря его превосходным характеристикам производительности.

4.4.1. Первое знакомство с алгоритмом k–средних

Начнем с выполнения алгоритма k-средних, чтобы получить какие-то кластеры. В листинге 4.8 показаны шаги, которые необходимо выполнить, чтобы загрузить данные из табл. 4.1 и выполнить предоставляемую нами реализацию алгоритма k-средних. Чтобы сравнить результаты работы алгоритма k-средних с результатами выполнения алгоритма одной связи, когда на четвертом уровне были идентифицированы восемь кластеров, мы выбираем $k = 8$.

Листинг 4.8. Алгоритм k-средних в действии

```
SFDataset ds = SFData.createDataset();

DataPoint[] dps = ds.getData();   ← Загрузка данных

KMeansAlgorithm kMeans = new KMeansAlgorithm(8, dps);   ← Инициализация алгоритма k-средних

kMeans.cluster();   ← Старт кластеризации

kMeans.print();
```

На рис. 4.11 показаны варианты кластеров, полученные в результате применения алгоритма k-средних; сравните эти кластеры с теми, которые были идентифицированы другими (иерархическими) алгоритмами, особенно с кластерами, показанными на рис. 4.8, где их опять-таки 8. Главным для идеи алгоритма k-средних является понятие *центроида* (centroid) кластера, который также называют *центром* (center) или *средним значением* (mean value). Смотрите на элементы, образующие кластер, как на тела, обладающие массой: центроид кластера был бы эквивалентен центру тяжести для этой системы тел.

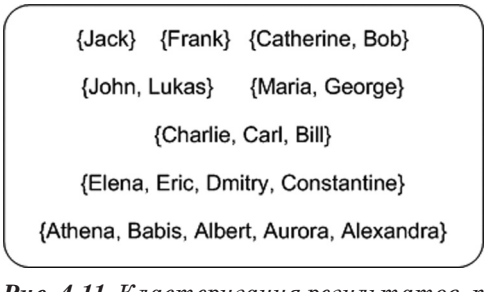

Рис. 4.11. *Кластеризация результатов, полученных при использовании алгоритма k-средних с $k = 8$*

На рис. 4.12 проиллюстрировано понятие центроида для кластера, точки которого (показаны кружочками черного цвета) занимают положение в вершинах шестиугольника. Центроид кластера (показан как пунк-

тирная окружность) находится в центре этого шестиугольника, из-за симметрии. Сам по себе центроид не обязан быть одной из точек данных, которые мы хотим объединить в кластеры. На самом деле, как показано на рис. 4.12, центроид чаще всего не будет принадлежать к точкам данных. Его роль – создать характерную опорную точку для набора точек, образующих этот кластер.

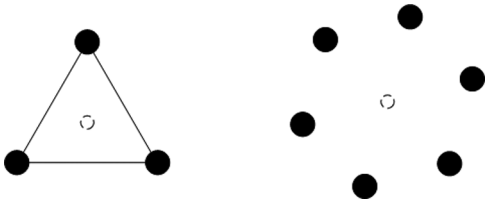

Рис. 4.12. *Центроиды (пунктирные окружности) для треугольного и шестиугольного кластеров точек (черные кружки)*

Есть вероятность, что варианты кластеров, которые вы получите при выполнении сценария из листинга 4.8, будут отличаться от тех, что показаны на рис. 4.11. Причина любых отличий кроется в инициализации местонахождения центроидов; это станет ясно в следующем разделе.

4.4.2. Внутренние механизмы работы алгоритма k-средних

Чтобы лучше понять внутреннюю работу алгоритма k-средних, рассмотрим его реализацию, представленную в листинге 4.9.

Листинг 4.9. KMeansAlgorithm: основной метод реализации алгоритма k-средних

```
public void cluster() {

   boolean centroidsChanged = true;

   while (centroidsChanged == true) {
      List<Set<DataPoint>> clusters = new ArrayList<Set<DataPoint>>(k);

      for (int i = 0; i < k; i++) {
        clusters.add(new HashSet<DataPoint>());
      }

      for (DataPoint p : allDataPoints) {
        int i = findClosestCentroid(allCentroids, p);
        clusters.get(i).add(p);
      }

      for (int i = 0; i < k; i++) {
        allClusters[i] = new Cluster(clusters.get(i));
```

Создание набора точек для каждого кластера

Назначение точки на основе расстояния

Создание кластеров

```
      }

      centroidsChanged = false;

     for (int i = 0; i < allClusters.length; i++) {

       if (clusters.get(i).size() > 0) {  ← Вычисление новых центроидов

         double[] newCentroidValues = findCentroid(allClusters[i]);

         double[] oldCentroidValues =
➥        allCentroids[i].getNumericAttrValues();

         if (!Arrays.equals(oldCentroidValues, newCentroidValues)) {

           allCentroids[i] =
➥           new DataPoint(allCentroids[i].getLabel(), newCentroidValues);

           centroidsChanged = true;
         }
       } else {
         // Центроид остается без изменений, если в кластере нет элементов
        }
      }
   }
}

public static DataPoint[] pickInitialCentroids(int k, DataPoint[] data) {

  Random randGen = new Random();

  DataPoint[] centroids = new DataPoint[k];

  Set<Integer> previouslyUsedIds = new HashSet<Integer>();

for (int i = 0; i < k; i++) {

    // Выбор точечного индекса, который мы еще не использовали
    int centroidId;
    do {
      centroidId = randGen.nextInt(data.length);
    }
    while( previouslyUsedIds.add(centroidId) == false );

    String label = “Mean-”+i+”(“+data[centroidId].getLabel()+”)”;

    double[] values = data[centroidId].getNumericAttrValues();

    String[] attrNames = data[centroidId].getAttributeNames();

    centroids[i] = new DataPoint(label,
➥Attributes.createAttributes(attrNames, values));
  }

  return centroids;
}
```

Алгоритм k-средних случайным образом выбирает (см. метод `pickInitialMeanValues`) k точек, которые представляют исходные центроиды вариантов кластеров. Затем вычисляются расстояния между этими центроидами и всеми точками набора, и каждая точка приписывается к тому кластеру, для которого расстояние между центроидом кластера и этой точкой оказывается минимальным. В результате этих назначений у нас сейчас изменилось местонахождение центроидов каждого кластера, поэтому мы повторно вычисляем новые центроиды до тех пор, пока не перестанет изменяться их местонахождение. Этот конкретный алгоритм k-средних приписывают Е. В. Форги (E. W. Forgy) и С. П. Ллойду (S. P. Lloyd), и у него есть следующие преимущества:

- Этот алгоритм хорошо работает с несколькими метриками.
- Легко получить версию этого алгоритма, которая выполняется параллельно, – когда данные разбиты, скажем, на n наборов и кластеризация осуществляется параллельно для каждого отдельного набора данных, на n различных вычислительных устройствах.
- Этот алгоритм не чувствителен к упорядочению данных.

На этом этапе вас может интересовать, что происходит, если алгоритм не прекратит работу. Не волнуйтесь! Гарантируется, что итерации будут прекращены через конечное число шагов. На практике этот алгоритм демонстрирует быструю *сходимость* (converges) (математический жаргон). Разумеется, мы всегда должны проявлять осторожность в отношении различных вариантов этого алгоритма. Если выбранная вами метрика не является евклидовым расстоянием, вы можете столкнуться с проблемами; см., например, статью о кластеризации очень больших коллекций документов [Dhillon, I.S., J. Fan, and Y. Guan], в которой применение косинусного сходства подчинено евклидову расстоянию. В другом случае некоторые из этих же авторов рассказывают о преимуществе использования отклонений Кульбака–Ляйблера (Kullback-Leibler divergences) (которые даже не являются расстояниями!) вместо возведенных в квадрат евклидовых расстояний.

Алгоритм k-средних работает быстро, особенно в сравнении с другими алгоритмами кластеризации. Как правило, его вычислительная сложность оценивается как $O(N)$, где N – число точек данных, которые мы хотим объединить в кластеры. Достаточно сказать, что имя процедуры, реализованной для алгоритма k-средних в коммерческом пакете SAS, – `FASTCLUS` (для быстрой кластеризации).

Обратите внимание: в отличие от агломеративных алгоритмов, алгоритм k-средних требует задания числа кластеров, которые должны быть сформированы, в качестве входного параметра. Естественно, возникает вопрос: каким должно быть значение k? Ответ (снова) зависит от ваших данных: вы должны выполнить алгоритм k-средних с разными значениями k и проанализировать полученные кластеры. Иногда, как в случае очень большого набора данных или когда требуется иерархи-

ческая кластеризация, полезно сначала выполнить алгоритм k-средних с маленьким значением k, а затем выполнять алгоритм иерархической кластеризации внутри крупных сегментов, образованных с помощью алгоритма k-средних. Такой подход естественным образом идеально подходит для распараллеливания, и вы можете извлечь выгоду из того диапазона дополнительных вычислительных возможностей, которые (и если) у вас имеются!

Заметим также, что алгоритм k-средних подходит для точек данных с числовыми атрибутами. Проблема использования алгоритма k-средних в случае категорийных данных (таких как строковые значения) сводится к нахождению соответствующего числового представления нечисловых значений атрибутов. В последнем случае важен также выбор метрики.

Вы должны знать, что выбор исходных центроидов крайне важен для быстрого завершения итераций, а также для формирования кластеров хорошего качества. С математической точки зрения, алгоритм k-средних пытается минимизировать среднеквадратичное расстояние между точками одного кластера. Так, если вы выбираете исходные центроиды в областях с высокой концентрацией точек данных, то представляется обоснованной возможность получить результаты быстрее и добиться высокого качества кластеров. Именно это предлагают [Arthur, D. and S. Vassilvitskii], описав в своей недавней статье то, что они называют алгоритмом k-средних++ (k-means++).

Итак, в предыдущих разделах был представлен ряд алгоритмов, позволяющих идентифицировать группы пользователей на веб-сайте. Конечно, проявив изобретательность, вы сможете применить эти же алгоритмы в иных обстоятельствах. Можно также комбинировать алгоритмы, иногда даже рекомендуется поступать таким образом. В промышленности предпочтение отдается алгоритму k-средних и его вариантам. Алгоритм k-средних является предпочтительным из-за его простоты (в реализации), скорости и способности выполняться на платформе, обеспечивающей параллельные вычисления.

4.5. Устойчивая кластеризация, использующая связи (ROCK)

В этом разделе мы продолжаем рассматривать алгоритмы кластеризации и анализируем алгоритм, который отличается от всего того, что мы видели до сих пор, в двух отношениях. Во-первых, этот алгоритм особенно хорошо подходит для категорийных данных, таких как ключевые слова, булевские атрибуты, перечисления и так далее. Во-вторых, этот алгоритм разработан так, что должен хорошо работать на очень больших наборах данных. Примером нам послужит коллекция данных с сайта Digg.com.

Для иллюстрации мы будем использовать фиксированный набор данных, который вы можете найти в каталоге *data/ch4*, файл *ch4_digg_stories.csv*. Эти данные были собраны с применением интерфейса Digg API, описанного в главе 3. Данные содержат предоставленные 10 случайными пользователями 49 новостей с сайта Digg с несколькими атрибутами. В этих данных мы скомпоновали 8 кластеров, которые легко идентифицируются человеком: откройте этот файл в своем любимом текстовом редакторе и просмотрите его данные. Можно ли идентифицировать эти кластеры, используя наши алгоритмы вместо глаз? Давайте посмотрим!

4.5.1. Введение в алгоритм ROCK

В листинге 4.10 выполняется загрузка данных, инициализируется алгоритм `ROCKAlgorithm`, а для отображения структуры кластеров используется уже знакомый вам класс `Dendrogram`.

Листинг 4.10. Кластеризация больших коллекций веб-новостей с помощью алгоритма ROCK

```
MyDiggSpaceDataset ds = MyDiggSpaceData.createDataset(15);

DataPoint[] dps = ds.getData();

ROCKAlgorithm rock = new ROCKAlgorithm(dps, 5, 0.2);

Dendrogram dnd = rock.cluster();

dnd.print(21);
```

Загрузка новости с сайта Digg, используются только первые 15 термов

Инициализация алгоритма ROCK для нахождения 5 заданных кластеров

В методе `print` класса `Dendrogram` мы ограничиваем выходные данные кластерами, в которых больше одного элемента. Другими словами, мы не показываем единичные элементы (в специальной терминологии – *синглтоны*), чтобы образование групп было более наглядным. На рис. 4.13 приведены результаты выполнения программного кода из листинга 4.10.

Обратите внимание: в качестве текста для идентификации похожих новостей с нашего форума алгоритм использует не просто заголовки, а скорее заголовки и описания. Эти описания могут существенно отличаться с точки зрения синтаксиса (см., например, новости, связанные с донорами крови и с сетью Facebook), что должно исключить возможность прямого сравнения строк их контента. Ключ в том, что коэффициент Жаккарда зависит не от порядка слов в тексте, но от количества общих термов в описаниях. Как можно видеть, на 21-м уровне 6 из 8 кластеров были идентифицированы правильно. Используйте собственные данные и посмотрите, какого рода кластеры вы получите для своих документов, новостей, статей и так далее – список можно продолжать до бесконечности! До тех пор пока вы размещаете свои данные в массиве класса `DataPoint`, вас ничто не сдерживает.

```
bsh % dnd.print(21);
Clusters for: level=21, Goodness=1.044451296741812
_____________________________________________________
{5619782:Lack Of Democracy on Digg and Wikipedia?,
 5611165:Lack Of Democracy on Digg and Wikipedia?}
_____________________________________________________
{5571841:Lack Of Democracy on Digg and Wikipedia?,
 5543643:Lack Of Democracy on Digg and Wikipedia?}
_____________________________________________________
{5142233:The Confederacy's Special Agent,
 5620839:The Confederacy's Special Agent,
 5586183:The Confederacy's Special Agent,
 5610584:The Confederacy's Special Agent,
 5598008:The Confederacy's Special Agent,
 5613383:The Confederacy's Special Agent,
 5613380:The Confederacy's Special Agent}
_____________________________________________________
{5585930:Microsoft, The Jekyll And Hyde Of Companies,
 5524441:Microsoft, The Jekyll And Hyde Of Companies,
 5609070:Microsoft, The Jekyll And Hyde Of Companies,
 5618201:Microsoft, The Jekyll And Hyde Of Companies,
 5620878:Microsoft, The Jekyll And Hyde Of Companies,
 5609797:Microsoft, The Jekyll And Hyde Of Companies}
_____________________________________________________  ________
{5607788:Recycle or go to Hell, warns Vatican -- part I,
 5592940:Recycle or go to Hell, warns Vatican -- part II,
 5618262:Recycle or go to Hell, warns Vatican -- part III,
 5595841:Recycle or go to Hell, warns Vatican --- part IV}
_____________________________________________________________
{5608052:Contract Free on AT&T,
 5620493:Contract Free on AT&T,
 5621623:Contract Free on AT&T,
 4955184:Contract Free on AT&T,
 5594161:Contract Free on AT&T}
```

Рис. 4.13. *Результаты кластеризации, полученные при выполнении кода из листинга 4.10*

4.5.2. Почему ROCK – это надежно?

Давайте внимательнее посмотрим на внутренние механизмы работы алгоритма ROCK. В листинге 4.11 показаны конструктор и главный метод `cluster` класса `ROCKAlgorithm`. Ключевая идея алгоритма ROCK состоит в том, чтобы использовать связи как меру сходства вместо показателя, основанного только на расстояниях. Разумеется, чтобы определить точки, которые «связаны» с любой другой точкой, нам все-таки потре-

буется использовать уже знакомые показатели расстояния. Цель – объединить в одном кластере точки, имеющие много общих связей.

Листинг 4.11. ROCKAlgorithm: метод cluster алгоритма устойчивой кластеризации на основе связей

```
public ROCKAlgorithm(DataPoint[] points, int k, double th) {

   this.points = points;   ← Точки данных для кластера

   this.k = k;   ← Минимальное число кластеров

   this.th = th;   ← Порог создания связей

   this.similarityMeasure = new JaccardCoefficient();   ← Матрица сходства

   this.linkMatrix =
➥ new LinkMatrix(points, similarityMeasure, th);   ← Матрица связей
}
public Dendrogram cluster() {
  List<Cluster> initialClusters = new ArrayList<Cluster>();
  for(int i = 0, n = points.length; i < n; i++) {   ❶
      Cluster cluster = new Cluster(points[i]);
      initialClusters.add(cluster);
  }

  double g = Double.POSITIVE_INFINITY;

  Dendrogram dnd = new Dendrogram("Goodness");

  dnd.addLevel(String.valueOf(g), initialClusters);

  MergeGoodnessMeasure goodnessMeasure = new MergeGoodnessMeasure(th);   ❷

  ROCKClusters allClusters = new ROCKClusters(initialClusters,
➥ linkMatrix, goodnessMeasure);   ❸

  int nClusters = allClusters.size();

while( nClusters > k ) {   ❹

      int nClustersBeforeMerge = nClusters;

      g = allClusters.mergeBestCandidates();

      nClusters = allClusters.size();

      if( nClusters == nClustersBeforeMerge ) {
        // Нет связанных кластеров для объединения
        break;
      }
      dnd.addLevel(String.valueOf(g), allClusters.getAllClusters());
  }
  return dnd;
}
```

Конструктор получает следующие аргументы.

- Точки данных, которые мы хотим объединить в кластер.
- Минимальное число кластеров, которое мы хотим получить; алгоритм ROCK является восходящим агломеративно-иерархическим алгоритмом: мы начинаем с того, что каждая точка находится в ее собственном кластере, и продолжаем объединение до тех пор, пока все точки не будут принадлежать единственному кластеру. Данный параметр позволяет остановить обработку до того, как все точки будут сгруппированы в единственный кластер, задавая минимальное число необходимых нам кластеров.
- Параметр, определяющий степень близости между двумя точками, которую требуется иметь, чтобы сформировать связь между этими точками.

В конструкторе мы создаем экземпляр класса, обеспечивающего вычисление Жаккардова сходства (`JaccardCoefficient`), и экземпляр нового класса (`LinkMatrix`), инкапсулирующего реализацию структуры связей между точками данных. Вы можете использовать другую меру расстояния, например `CosineSimilarity`, и проанализировать, какие будут получены кластеры: лучше, хуже или те же самые. Сможете ли вы объяснить случаи сходства и отличия в результатах? Поэкспериментировав, вы скоро поймете, что для каждой меры расстояния будет использовано свое, отличное от других пороговое значение, но в итоге ваши результаты будут в большой степени совпадать, поэтому мы и называем этот алгоритм «устойчивым».

Разумеется, этот класс не в состоянии полностью обслужить алгоритм ROCK. Он делегирует часть работы ряду других классов, которые мы скоро изучим. Далее перечислены шаги, реализованные в методе `cluster` из листинга 4.11:

❶ Этап инициализации, на котором мы создали новый кластер для каждой точки данных.

❷ На этом шаге создается *критерий качества* (goodness measure), который поможет оценить, должны мы объединить два кластера или нет. В каждом алгоритме кластеризации важно ответить на вопрос: «Какие кластеры являются лучшими?» Сумев определить «лучшие» кластеры, мы сможем разработать алгоритмы, нацеленные на «производство» таких кластеров. Алгоритм ROCK считает, что лучшие кластеры – те, для которых значение критерия качества максимально.

❸ Класс `ROCKClusters` инкапсулирует все релевантные данные и алгоритмы, необходимые для идентификации на основе критерия качества лучших кластеров, которые должны быть сформированы.

❹ На этом шаге повторяется процесс идентификации лучших кластеров и вводятся в действие два критерия завершения процесса. Во-

первых, работа алгоритма прекращается, если число уже сформированных кластеров равно заданному минимальному числу кластеров. Не забывайте, что если разрешить алгоритму выполняться без такого критерия, в результате все точки данных будут объединены в один кластер, что не очень информативно. Во-вторых, работа алгоритма завершается, если число кластеров между двумя итерациями не изменилось и продолжать их нет смысла.

Давайте повнимательнее рассмотрим класс `MergeGoodnessMeasure`. Как мы уже сказали, этот класс инкапсулирует критерий, используемый для оценки качества полученного кластера. У алгоритмов, основанных только на расстоянии сходства, нет возможности легко отличить один кластер от другого, если эти кластеры не являются «хорошо обособленными», поскольку ближайшими соседями могут оказаться точки данных, принадлежащие к разным кластерам. Таким образом, другие алгоритмы могут объединить два кластера только потому, что два принадлежащие этим кластерам элемента (по одному с каждой стороны) близки друг к другу, даже если у этих двух точек мало общих соседей.

Следовательно, первым делом мы хотим убедиться в том, что выбранный критерий качества для выявления лучших кластеров способен помочь нам эффективно разбираться с такими случаями. Для достижения этой цели алгоритм ROCK, как и предполагает его название, использует связи. Что такое связь? Мы говорим, что две точки данных связаны друг с другом, если у них есть общий сосед. В вопросе о том, надо ли объединять кластеры X и Y, нас интересует число связей между всеми парами точек из кластеров X и Y: одна точка такой пары берется из кластера X, а другая – из кластера Y. Большое число связей должно указывать на то, что эти две точки с высокой вероятностью принадлежат одному кластеру и должны обеспечить лучшие кластеры. Соответствующий программный код приведен в листинге 4.12.

Листинг 4.12. MergeGoodnessMeasure: критерий качества для идентификации лучших кластеров

```
public class MergeGoodnessMeasure {

    private double th;

    private double p;

    public MergeGoodnessMeasure(double th) {
        this.th = th;
        this.p = 1.0 + 2.0 * f(th);
    }

    public double g(int nLinks, int nX, int nY) {
        double a = Math.pow(nX + nY, p);
        double b = Math.pow(nX, p);
        double c = Math.pow(nY, p);
```

```
        return (double)nLinks / (a - b - c);
    }

    private double f(double th) {

        return (1.0 - th) / (1.0 + th);
    }
}
```

Самое важное обращение к методу — это `g(int nLinks, int nX, int nY)`, где `nLinks` — число связей между кластерами *X* и *Y*. Вы должны были ожидать (на основании сказанного нами об общих соседях и связях), что значение, возвращаемое методом `g`, будет зависеть от числа связей между любыми двумя точками двух кластеров. Но что означают другие аргументы? Почему формула так усложнилась? Давайте ответим на первый вопрос. Параметры `nX` и `nY` — это число точек данных, содержащихся в кластерах *X* и *Y*, соответственно. Ответ на второй вопрос не так прост, но гораздо более интересен.

Вы, вероятно, полагаете, что максимизация числа связей для всех пар точек двух кластеров должна обеспечить достаточный критерий качества, чтобы решить, объединять ли данные кластеры. Вспомните, однако, что мы осуществляем кластеризацию с двоякой целью. Нам требуется объединить в группу точки данных, принадлежащие одному кластеру, и отбросить точки, этому кластеру не принадлежащие. Даже несмотря на то, что максимизация числа связей должна была бы гарантировать попадание в один кластер точек с большим числом связей, это не помешает алгоритму поместить в один кластер все точки. Другими словами, использование только числа связей не поможет разнести по разным кластерам точки с малым числом связей между ними.

Формула ROCK позволяет оценить суммарное число связей в кластере *X* и запомнить эту сумму в переменной `b`, а суммарное число связей в кластере *Y* запоминается в переменной `c`. Метод `f` обеспечивает реализацию простой функции, обладающей следующим важным свойством: каждая точка, принадлежащая кластеру *X*, имеет приблизительно `Math.pow(nX,f(th))` соседей в кластере *X*. Следовательно, при вычислении этого критерия качества число связей в каждой паре точек делится на *ожидаемое* (expected) число связей, представленное как `(a - b - c)`. Это свойство критерия качества не позволяет назначить одному кластеру точки данных, не имеющие большого числа связей между ними. Переменная `th` вычисляется в данном варианте реализации метода `f` и получает значение от 0 до 1 включительно. Если значение этой переменной равно 0, то значение, возвращаемое методом `f`, равно 1, и все точки данных являются соседями. Если значение переменной `th` равно 1, то значение, возвращаемое методом `f`, равно 0, и единственным соседом точки является она сама. Как выяснилось, такая реализация метода

`f` хорошо работает с данными потребительской корзины, но в других случаях может оказаться непригодной. Условия, в которых функция `f` непригодна к применению, – более или менее однородные значения атрибутов для всех точек данных. Поэкспериментируйте с реализацией метода `f` в применении к нашим или собственным данным. Дополнительные сведения по этому вопросу можно найти в разделе «Сделать».

4.6. DBSCAN

В этом разделе описан улучшенный алгоритм пространственной кластеризации на основе плотности для приложений с шумами (Density-Based Spatial Clustering of Applications with Noise, DBSCAN), в котором вместо понятия связей или прямого расстояния между точками используется появившееся в последнее время понятие *плотности точек* (point density). Чтобы проиллюстрировать это понятие, предположим, что у вас есть неглубокое блюдо с исходно чистой водой и вы пролили в него несколько чернильных капель. Сразу же после попадания чернил в воду вам нетрудно идентифицировать область, содержащую чернила, поскольку свет по-разному отражается от чернил и от воды из-за их разной плотности. Какое отношение это имеет к кластеризации?

4.6.1. Первое знакомство с алгоритмами на основе плотности

Есть целый класс алгоритмов кластеризации, которые пытаются извлечь выгоду из этого простого будничного опыта. В частности, *алгоритмы на основе плотности* (density-based algorithms) являются следствием интуитивно понятной идеи о том, что при распознавании зрительных образов границы объектов определяются на основе градиентов плотности. Таким образом, распространив этот принцип на произвольные двух-, трех- и даже многомерные пространства, мы можем получить возможность идентифицировать кластеры (объекты) на основе понятия плотности точек в определенной области пространства. Большинство из нас, взглянув на левую часть рис. 4.14, зрительно выделят три кластера (показаны в правой части рис. 4.14). Как правило, точки, не принадлежащие к этим кластерам (семь белых кружков), мы рассматривали бы как шум.

Алгоритм DBSCAN, предложенный Мартином Эстером [Ester M., Kriegel H.-P., Jorg S., Xu X], разработан для обнаружения кластеров и шума в наборе данных. Прежде чем углубиться в детали, выполним сценарий из листинга 4.13, чтобы получить результаты кластеризации для тех же данных, которые были использованы в предыдущем разделе для алгоритма ROCK.

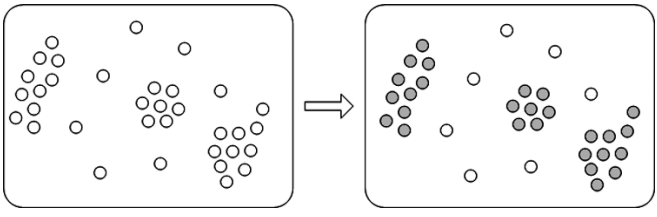

Рис. 4.14. *Кластеризация на основе плотности обусловлена нашей способностью зрительно распознавать формы и образы*

Листинг 4.13. Использование алгоритма DBSCAN

```
MyDiggSpaceDataset ds = MyDiggSpaceData.createDataset(15);

DataPoint[] dps = ds.getData();

CosineDistance cosD = new CosineDistance();

DBSCANAlgorithm dbscan = new DBSCANAlgorithm(dps,cosD,0.8,2,true);

dbscan.cluster();
```

Загрузка новости с сайта Digg и использование только первых 15

Инициализация алгоритма DBSCAN

Как можно видеть, применять алгоритм DBSCAN так же легко, как и все другие представленные нами алгоритмы. Единственный шаг, который предшествует созданию экземпляра класса `DBSCANAlgorithm`, помимо загрузки данных, – это определение соответствующей метрики расстояния. В данном случае мы использовали класс `CosineDistance`, но подошел бы и любой другой класс, реализующий интерфейс расстояния.

Конечно, мы могли бы интегрировать метрику расстояния в реализацию алгоритма, но, как оказывается, выбор метрики расстояния имеет значение. Следовательно, лучше определить эту метрику явно, как аргумент в конструкторе. Выбор метрики расстояния определяет форму (плоскую или объемную) «соседства», а соседство, в свою очередь, определяет различные параметры, относящиеся к плотности, которые мы сейчас исследуем. Но сначала посмотрим на результаты. Выполнение сценария из листинга 4.13 обеспечит выходные данные, приведенные на рис. 4.15.

Это довольно хорошие результаты! Обратите внимание: алгоритм корректно идентифицировал очевидные кластеры, но также выявил не столь очевидные кластеры статей (кластер 8). Кроме того, этот алгоритм определил как шум точки данных, не принадлежащие ни одному из данных кластеров. Но заметим также, что есть еще один кластер, который можно было бы извлечь из элементов шума, а новость с идентификатором `ID=5619818` можно было бы записать в первый кластер.

```
bsh % dbscan.cluster();
DBSCAN Clustering with NeighborThreshold=0.8 minPoints=2
Clusters:
1:
{5605887:A Facebook Application To Find Blood Donors Fast,
5611687:A Facebook Application To Find Blood Donors Fast,
5608576:A Facebook Application To Find Blood Donors Fast}
_____________________________________________________________
2:
{5142233:The Confederacy's Special Agent,
5613383:The Confederacy's Special Agent,
5620839:The Confederacy's Special Agent,
5598008:The Confederacy's Special Agent,
5586183:The Confederacy's Special Agent,
5610584:The Confederacy's Special Agent,
5613380:The Confederacy's Special Agent}
_____________________________________________________________
3:
{5620878:Microsoft, The Jekyll And Hyde Of Companies,
5618201:Microsoft, The Jekyll And Hyde Of Companies,
5585930:Microsoft, The Jekyll And Hyde Of Companies,
5609797:Microsoft, The Jekyll And Hyde Of Companies,
5609070:Microsoft, The Jekyll And Hyde Of Companies,
5524441:Microsoft, The Jekyll And Hyde Of Companies}
_____________________________________________________________
4:
{5594161:Contract Free on AT&T,
4955184:Contract Free on AT&T,
5608052:Contract Free on AT&T,
5621623:Contract Free on AT&T,
5579109:Contract Free on AT&T,
5620493:Contract Free on AT&T}
_____________________________________________________________
5:
{5607863:Lack Of Democracy on Digg and Wikipedia?,
 5571841:Lack Of Democracy on Digg and Wikipedia?,
 5619782:Lack Of Democracy on Digg and Wikipedia?,
 5611165:Lack Of Democracy on Digg and Wikipedia?,
 5543643:Lack Of Democracy on Digg and Wikipedia?}
_____________________________________________________________
6:
{5481876:How Traffic Jams Occur : Simulation,
 5613023:How Traffic Jams Occur : Simulation}
_____________________________________________________________
7:
{5617459:Robotic drumstick keeps novices on the beat,
 5619693:Robotic drumstick keeps novices on the beat}
_____________________________________________________________
```

```
8:
{5617998:Obama: “”I Am NOT Running for Vice President””,
 5625315:Obama Accuses Clinton of Using “”Republican Tactics””}
_____________________________________________________________
9:
{5607788:Recycle or go to Hell, warns Vatican -- part I,
 5595841:Recycle or go to Hell, warns Vatican --- part IV,
 5618262:Recycle or go to Hell, warns Vatican -- part III,
 5592940:Recycle or go to Hell, warns Vatican -- part II}
_____________________________________________________________
Noise Elements:
 {5610213:Senate panel critiques prewar claims by White House,
  5619818:A Facebook Application To Find Blood Donors Fast,
  5612810:Super Mario Bros Inspired Wii with USB base [ Pics ],
  5522983:Smoking Monkey[flash],
  5609833:NSA’s Domestic Spying Grows As Agency Sweeps Up Data,
  5625339:Lawmaker’s Attempt to Criminalize Anonymous Posting
Doomed,
  5610081:Digg’s Algo Change Cut Promotions by 38%,
  5604438:Archaeologists Unveil Finds in Rome Digs,
  5614085:House Files Contempt Lawsuit Against Bush Officials,
  5592473:Maryland police officers refuse to pay speeding
tickets,
  5622802:House Democrats Defy White House on Spying Program}
```

Рис. 4.15. *Результаты кластеризации, полученные при выполнении кода из листинга 4.13*

4.6.2. Внутренние механизмы алгоритма DBSCAN

Итак, подробности! Во-первых, нам необходимо определить аргументы, которые получает конструктор класса `DBSCANAlgorithm`. Этот конструктор имеет следующую сигнатуру:

```
public DBSCANAlgorithm(DataPoint[] points,
                       Distance distance,
                       double eps,
                       int minPoints,
                       boolean useTermFrequencies)
```

К этому моменту вам уже должен быть знаком массив `DataPoint`: именно здесь мы храним данные. Интерфейс `Distance` позволяет передать то расстояние, которое мы считаем наиболее подходящим для наших данных. Веселье начинается с переменной `eps`, что скорее всего означает эпсилон – букву греческого алфавита, которой обычно обозначают малое положительное число. Значение эпсилон помогает определить *эпсилон-окрестность* (epsilon neighborhood) для любой данной точки `DataPoint` *p*

как множество точек `DataPoints (q)`, расстояние от которых до *p* меньше или равно эпсилон. Итак, определение эпсилон-окрестности является достаточно простым и не должно противоречить вашим ожиданиям. Все усложняется для следующих нескольких определений, поэтому давайте рассмотрим рис. 4.16, напоминающий рисунок из оригинальной статьи [Ester M., Kriegel H.-P., Jorg S., Xu X].

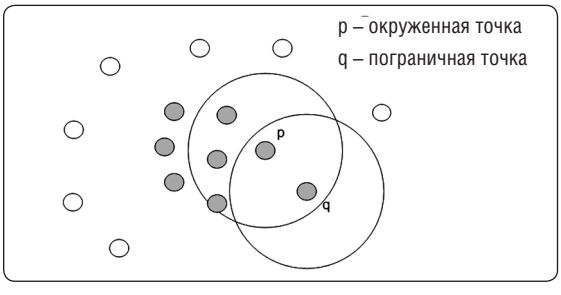

Рис. 4.16. *Окруженные и пограничные точки в алгоритме DBSCAN*

Большие окружности на рис. 4.16 являются эпсилон-окрестностями для точек данных *p* и *q* – одна окружность с центром в точке *p*, а другая с центром в точке *q*. Радиус окружности равен эпсилон (`eps`), а в переменной `minPoints` хранится минимальное число точек, которые должны находиться внутри соответствующей окружности, чтобы эта точка данных рассматривалась как *окруженная точка* (core point). Точки, которые принадлежат кластеру, но не являются окруженными, называются *пограничными точками* (border points). Согласно этой терминологии, точка данных *p* является окруженной, а точка данных *q* является пограничной. Мы говорим, что точка данных *p* *непосредственно плотно-достижима* (directly density-reachable) из точки данных *q*, с учетом значений переменных `eps` и `minPoints`, если выполняются следующие два условия:

- Точка *p* находится внутри эпсилон-окрестности точки *q*
- Внутри эпсилон-окрестности точки *q* находятся больше чем `minpoints` точек данных

На рис. 4.16 точка *q* является непосредственно плотно-достижимой из точки *p*, но точка *p* не является непосредственно плотно-достижимой из точки *q*. Это те базовые понятия, которые необходимы для понимания программного кода класса `DBSCANAlgorithm`, главные методы которого показаны в листинге 4.14.

Листинг 4.14. DBSCANAlgorithm: два главных метода нашей реализации

```
public List<Cluster> cluster() {
  int clusterId = getNextClusterId();
```

```
  for(DataPoint p : points) {
     if( isUnclassified(p) ) {

       boolean isClusterCreated = createCluster(p, clusterId); ❶

       if( isClusterCreated ) {
          clusterId = getNextClusterId();
       }
     }
  }

  List<Cluster> allClusters = new ArrayList<Cluster>();
  for(Map.Entry<Integer, Set<DataPoint>> e : clusters.entrySet()) {
    String label = String.valueOf(e.getKey());
    Set<DataPoint> points = e.getValue();
    if( points != null && !points.isEmpty() ) {
      Cluster cluster = new Cluster(label, e.getValue());
      allClusters.add(cluster);
    }
  }
  return allClusters;
}

private boolean createCluster(DataPoint p, Integer clusterId) {

   boolean isClusterCreated = false;

   Set<DataPoint> nPoints = findNeighbors(p, eps);  ❷

   if( nPoints.size() < minPoints ) {  ❸
      assignPointToCluster(p, CLUSTER_ID_NOISE);
      isClusterCreated = false;
    } else {

      assignPointToCluster(nPoints, clusterId);  ❹
      nPoints.remove(p);

      while( nPoints.size() > 0 ) {  ❺
        DataPoint nPoint = nPoints.iterator().next();

        Set<DataPoint> nnPoints = findNeighbors(nPoint, eps);  ❻

        if( nnPoints.size() >= minPoints ) {
           for(DataPoint nnPoint : nnPoints ) {
               if( isNoise(nnPoint) ) {
                 assignPointToCluster(nnPoint, clusterId);  ❼
               } else if( isUnclassified(nnPoint) ){
                 nPoints.add(nnPoint);  ❽
                 assignPointToCluster(nnPoint, clusterId);
               }
           }
        }
      nPoints.remove(nPoint);    ❾
    }
```

```
    isClusterCreated = true;
  }
  return isClusterCreated;
}
```

❶ Для каждой точки в наборе данных, который не был кластеризован, создаем кластер, содержащий этот набор; все остальное в методе `cluster` – это просто механика.

❷ Находим эпсилон-окрестность точки данных p для данного параметра `eps`.

❸ Если в эпсилон-окрестности точки `p` недостаточно точек данных, то это или шум или пограничная точка. Временно принимаем эту точку за шум, а если это пограничная точка, мы изменим ее статус позже. На этом этапе для точки `p` не создано ни одного кластера.

❹ Если число точек данных в эпсилон-окрестности больше или равно минимальному числу точек данных `minPoints`, можем продолжать. Но сначала необходимо удалить данную точку данных (`p`) из ее собственного набора точек кластера.

❺ Итеративный перебор всех точек данных, кроме `p`.

❻ Находим их эпсилон-окрестности и выясняем, является ли текущая точка непосредственно плотно-достижимой.

❼ Если текущая точка является пограничной, следует записать ее в кластер.

❽ Текущая точка не является шумом, но надо проанализировать, является ли она окруженной, поэтому добавляем эту точку в исходный список, чтобы найти ее эпсилон-окрестность, и записываем ее в анализируемый кластер.

❾ Прежде чем переходить к обработке следующей точки данных из окрестности точки `p`, удаляем точку данных, которую только что проанализировали.

Как можно догадаться по этому описанию, важно выбрать соответствующие значения `eps` и `minPoints`. Лучший способ, позволяющий определить хорошие значения для этих двух параметров, – экспериментировать со своими данными для каких-то известных кластерных образований. Если вы имеете дело с двумерным набором данных (точка данных имеет два атрибута), целесообразно выбрать для `minPoints` значение 4. Вы можете использовать это значение как отправную точку, а затем взять еще несколько значений и посмотреть, улучшатся ли результаты кластеризации. Случай многомерных наборов данных более сложен, потому что свойства многомерных пространств довольно сильно отличаются от свойств пространств меньшей размерности (мы обсудим это

в приложении C). Выбрать значение eps легче, если задействовать следующие факторы.

- Нормализация данных. По возможности, следует всегда использовать нормализацию данных, особенно если измерений больше двух.
- Доступность статистических данных о расстояниях между точками.
- Размерность набора данных – сколько атрибутов должны иметь ваши точки данных?

Задайте такое исходное значение для переменной eps, чтобы расстояние между двумя точками в наборе воспринималось как «близкое». Затем создайте несколько приращений в геометрической прогрессии, скажем, со знаменателем 2, и снова выполните кластеризацию. Как меняются результаты при изменении значения переменной eps? Если результаты существенно не меняются, дело сделано: в качестве значения переменной eps выберите среднее значение. Если результаты меняются, возможны две ситуации:

- По мере возрастания значения переменной eps результаты либо последовательно улучшаются, либо последовательно ухудшаются.
- Характер изменения результатов не является последовательным: например, вы удваиваете значение переменной eps – результаты ухудшаются; вы снова удваиваете значение – результаты улучшаются!

Первая ситуация не вызывает затруднений. Если результаты кластеризации улучшаются по мере возрастания значения переменной, то выберите максимальное значение или продолжайте наращивать значение переменной (если у вас есть время и ресурсы), до тех пор пока не будут получены хорошие результаты.

Вторая ситуация встречается редко; как правило, она возникает при кластеризации данных большой размерности и обычно связана с неудачно выбранной метрикой расстояния. Следовательно, первым делом проанализируйте, действительно ли вам нужны все атрибуты. Во-вторых, попробуйте использовать другой показатель расстояния, в большей степени соответствующий вашим данным. Это основные идеи и основные шаги реализации самого алгоритма DBSCAN. В оригинальной статье [Ester M., Kriegel H.-P., Jorg S., Xu X] этот алгоритм описан гораздо подробнее, а этот раздел поможет в ней разобраться.

Все алгоритмы, которые мы описывали до сих пор, будут хорошо работать с большинством наборов данных. Как мы убедились, в зависимости от природы ваших данных одни алгоритмы работают лучше других. Тем не менее качество результатов – не единственный фактор, о котором нам необходимо позаботиться. В следующем разделе более подробно рассмотрены некоторые проблемы кластеризации, общие для очень больших наборов данных.

4.7. Вопросы кластеризации очень больших наборов данных

Есть две основных категории вопросов, встающих при обработке очень больших наборов данных. Первый вопрос – число точек данных, которые мы хотим подвергнуть кластеризации. Этот вопрос ведет нас к рассмотрению *вычислительной сложности* (computational complexity) алгоритмов кластеризации и ее влияния на их производительность. Другими словами, нам надо знать число вычислительных операций, которые необходимо выполнить, как функцию числа точек данных, которые мы хотим иметь в кластере. Второй вопрос – это число атрибутов (размерностей), которые могут быть важны для нашей кластеризации. Привычный для нас мир имеет три измерения, поэтому наша интуиция приспособлена для трех и менее измерений. В случае больших размерностей природа геометрических объектов и понятие близости отличаются. Этот факт проявляется двояко. Во-первых, с большим числом размерностей необходимо проделать больше вычислений, что, в свою очередь, понизит быстродействие. Во-вторых, возникает ряд вопросов, связанных с особой природой многомерного пространства и в совокупности известных как *проклятие размерности* (curse of dimensionality). Рассмотрим эти вопросы по отдельности.

4.7.1. Вычислительная сложность

Важно знать характеристики производительности алгоритмов кластеризации как функцию числа точек данных, которые мы хотим объединить в кластер. Есть огромная разница между попытками обнаружить кластеры пользователей на сайте MySpace (O (10^8) зарегистрированных пользователей) и в базе данных какой-то местной газеты (несколько сотен зарегистрированных пользователей) или муниципального колледжа (несколько тысяч студентов). Если мы хотим измерить влияние числа точек данных (n), то важно знать пространственно-временную *вычислительную сложность* алгоритмов. То есть размер оперативной памяти и, соответственно, количество операций, необходимых для работы конкретного алгоритма кластеризации. В табл. 4.2 приведены обе эти метрики для алгоритмов, которые мы реализовали в этой главе; здесь k означает число кластеров, а t – число итераций (в случае алгоритма k-средних).

Обратите внимание на преобладание фактора n^2. Именно эта квадратичная зависимость от числа точек – причина возникновения проблемы вычислительной сложности в случае многих алгоритмов кластеризации. В книге «Data Mining» [Dunham, M. H.] (см. раздел 5.9.2) предлагается более подробное сравнение алгоритмов кластеризации в такой форме. С точки зрения эффективности, алгоритм k-средних является безусловным победителем и, вероятно, из-за своей эффективности находит ши-

Таблица 4.2. Пространственно-временная сложность наших алгоритмов кластеризации

Алгоритм	**Пространственная сложность**	**Временная сложность**
Одной связи	$O(n^2)$	$O(k\ n^2)$
Средней связи	$O(n^2)$	$O(k\ n^2)$
MST одной связи	$O(n^2)$	$O(n^2)$
k-средних	$O(n^2)$	$O(t\ k\ n)$
ROCK	$O(n^2)$	$O(n^2 \log(n))$ или $O(n \log(n))$ с пространственными индексами
DBSCAN	$O(n^2)$	$O(n^2)$

рокое применение на практике. Но не забудьте, что этот алгоритм не обрабатывает категорийные данные. Временную сложность алгоритма DBSCAN можно уменьшить, как было указано, путем использования пространственных индексов точек данных; поскольку мы имеем дело с плотностью в метрических пространствах, естественно рассматривать значения атрибутов как координаты точек данных. Как правило, для пространственных индексов используются R-деревья, и большинство коммерческих баз данных предлагают реализации R-деревьев для пространственных данных. Тем не менее вы должны понимать трудности, сопряженные с индексацией пространственных данных в случае больших размерностей. Это область активных научных изысканий, и хотя опубликовано уже много хороших идей, последнее слово в решении этой проблемы (эффективная индексация данных с большой размерностью) еще не сказано.

Разумеется, для решения этих вопросов эффективности разработано много других алгоритмов кластеризации. Хорошо изучен и довольно популярен алгоритмом кластеризации BIRCH, специально разработанный для больших наборов данных. Его сложность, как пространственная, так и временная, оценивается линейно – $O(n)$, и для работы ему требуется однократное сканирование базы данных. Этот алгоритм принадлежит к алгоритмам, основанным на *сжатии данных* (data squashing). Иными словами, они создают структуры данных, в которых хранят сжатую информацию о данных. Описание таких алгоритмов выходит за рамки темы, рассматриваемой в этой книге. Для тех,

кто хочет больше узнать об алгоритмах сжатия данных, в конце главы есть раздел ссылок.

4.7.2. Большая размерность

Вторая обширная категория вопросов, связанных с очень большими наборами данных, – вопросы большой размерности данных. В очень больших наборах данных точки данных могут иметь много атрибутов, и если не пренебречь некоторыми из них с самого начала, метрическое пространство может распространиться в нескольких измерениях – иногда даже в сотнях измерений! Мы уже обращали ваше внимание на большую размерность, но не давали точного определения понятию «большая». Как правило, большая размерность предполагает, что мы имеем дело больше чем с 16 атрибутами – здесь мы сохраняем основание 2, – следовательно, чтобы на решаемую нами задачу повлияла большая размерность, требуемое число измерений должно быть *O (10)*. Вы, наверное, удивитесь: почему это так важно? Формулы, которые мы видели до сих пор, не ограничивают нас пространствами малой размерности. Так в чем же дело?

Есть две фундаментальные проблемы, связанные с большой размерностью, которые особенно важны для кластеризации (хотя большую часть того, что мы рассмотрим, следует также отнести к алгоритмам классификации). Первая проблема заключается в том, что большое число измерений приводит к увеличению размеров пространства, доступного для «рассеивания» точек данных. То есть если вы сохраняете фиксированное число точек данных и увеличиваете число атрибутов, предназначенных для описания этих точек, плотность точек в пространстве возрастает в геометрической прогрессии! Поэтому можно долго бродить вокруг да около, не в состоянии определить то образование (кластер), которому можно было бы отдать предпочтение в сравнении с другими.

Вторая фундаментальная проблема носит устрашающее название – *проклятие размерности* (curse of dimensionality). Говоря проще, это означает, что если есть *любой* набор точек с большой размерностью и вы используете *любую* метрику расстояния между этими точками, окажется, что все точки удалены друг от друга примерно на равное расстояние! Чтобы проиллюстрировать это важное следствие размерности, рассмотрим следующий простой случай, который представлен на рис. 4.17.

Если рассматривать рис. 4.17 слева направо, для каждого изображения размерность возрастает на 1. Мы начинаем с 8 точек в одном измерении (ось X), однородно распределенных, скажем, в интервале от 0 до 1. Как видим, минимальное расстояние, которое необходимо пройти от любой данной точки до тех пор, пока мы встретим другую точку: $\min(D) = 0{,}125$, тогда как максимальное расстояние: $max(D) = 1$. Таким образом, $min(D) / max(D) = 0{,}125$. В двух измерениях эти 8 то-

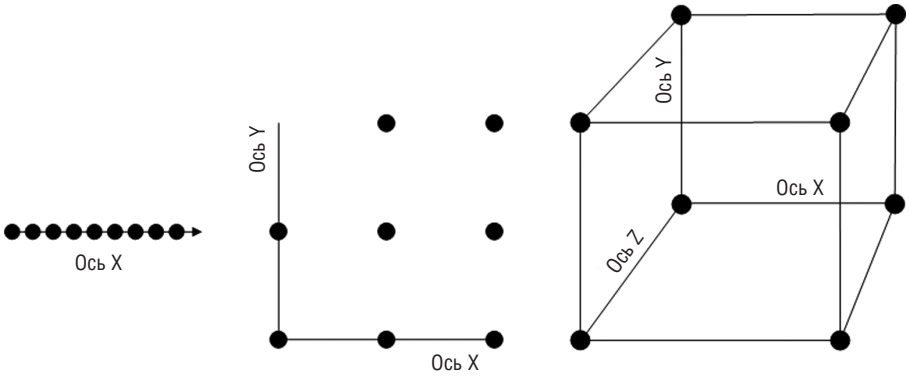

Рис. 4.17. *Проклятие размерности: каждая точка стремится быть равноудаленной от любой другой точки*

чек данных снова имеют однородное распределение, но теперь у нас *min(D) = 0,5* и *max(D) = 1,414* (вдоль главной диагонали); таким образом, *min(D) / max(D) = 0,354*. В трех измерениях: *min(D) = 1* и *max (D) = 1,732*; таким образом, *min(D) / max(D) = 0,577*. Другими словами, по мере возрастания числа измерений отношение минимального расстояния к максимальному расстоянию стремится к 1. То есть неважно, в каком направлении вы смотрите и какое расстояние измеряете, – все выглядит одинаково!

В результате все алгоритмы, основанные на вычислении расстояния, по мере возрастания числа измерений довольно быстро столкнутся с трудностями. В этом примере мы использовали стандартное евклидово расстояние, но вы можете использовать любую (подходящую) метрику расстояния, какую пожелаете, и убедиться в том, что эта проблема сохраняется. Некоторые [Beyer, K., R. Ramakrishnan, U. Shaft, J. Goldstein] даже задавались вопросом, имеет ли смысл говорить о понятии ближайших соседей в данных большой размерности. Интересный подход к решению этой проблемы описан в статье [Aggarwal, C. C.].

4.8. Заключение

Алгоритмы кластеризации полезны как инструмент исследования данных. Можно построить для наших данных иерархическую структуру из нескольких уровней кластеров или сформировать заранее заданное число кластеров. Есть несколько прикладных областей, где можно применить кластеризацию. Теоретически, для кластеризации пригоден любой набор данных, состоящий из объектов, которые можно определить в терминах значений атрибутов. Но требуется внимательно подходить к выбору меры расстояний между объектами и соответствующего алгоритма.

В этой главе мы рассмотрели объединение в группы сообщений, оставленных на форуме, и выявление похожих пользователей этого веб-сайта. Сложность рассмотренных алгоритмов варьируется от простых предложений языка запросов SQL до довольно развитых математических методов. Мы представили общий обзор типов кластеризации и полную реализацию шести алгоритмов: одной связи, средней связи, MST-алгоритм одной связи, алгоритм k-средних, алгоритмы ROCK и DBSCAN.

Алгоритмы одной связи, средней связи и MST-алгоритм одной связи являются агломеративно-иерархическими алгоритмами и исходят из предположения, что к моменту вычислений в наличии имеются все данные. По вычислительной сложности, как пространственной, так и временной, эти алгоритмы не очень благоприятны, поскольку их соответствующие показатели возрастают как квадрат числа точек данных. Таким образом, хотя эти алгоритмы легко реализовать, они не будут хорошо работать на больших наборах данных. Исключение здесь составляет только алгоритм одной связи на базе дерева MST. Мы можем улучшить временную сложность MST-алгоритма одной связи и сделать ее почти пропорциональной числу объектов данных.

Итеративный плоский алгоритм k-средних очень эффективен и часто обеспечивает хорошие результаты. Но он плохо обрабатывает категорийные данные, поскольку полагается на геометрическое понятие центроида, которое вряд ли легко применимо в случае категорийных данных. Другой недостаток этого алгоритма – его неспособность обрабатывать *отклоняющиеся значения* (outliers) – точки, находящиеся далеко от основных кластеров.

Алгоритм ROCK особенно хорошо подходит для булевских и категорийных данных, поскольку полагается на число связей, а не на расстояния. Это агломеративно-иерархический алгоритм, пространственная сложность которого неблагоприятна – $O(n^2)$, а временная еще хуже – $O(\log(n)\, n^2)$.

Алгоритм DBSCAN использует понятие плотности и неявно различает окруженные и пограничные точки кластера. Этот алгоритм хорошо обрабатывает отклоняющиеся значения, и хотя его пространственно-временная сложность оценивается как $O(n^2)$, его можно использовать в сочетании с алгоритмом k-средних, что обеспечит хороший и быстрый способ кластеризации (см. соответствующее задание в разделе «Сделать»).

Наше рассмотрение не является ни исчерпывающим, ни всеобъемлющим. Есть еще много других алгоритмов, описанных в литературе, и много вариантов алгоритмов, уже рассмотренных нами. Втиснуть все их в один раздел этой книги невозможно. Но вы уже познакомились со всеми основными понятиями и располагаете несколькими полностью

реализованными алгоритмами кластеризации, с которыми можете экспериментировать и которые можете дополнить так, чтобы они соответствовали вашей задаче.

4.9. Сделать

1. *Алгоритм SQLEM* – основанный на языке запросов SQL вариант алгоритма максимизации ожиданий (expectation-maximization, EM). Это хорошо известный в статистике алгоритм, который использует два шага, *E-шаг* и *M-шаг*. Теория, на которой основан этот алгоритм, требует углубленных познаний в математике, наличия которых у вас мы не предполагаем. Но можно, не усложняя, рассматривать кластеризацию k-средних (см. раздел 4.4) как алгоритм EM; E-шаг – это назначение, а M-шаг – это обновление значений центроидов.

   Поскольку подавляющее большинство приложений так или иначе полагаются на реляционные базы данных, вы можете реализовать этот алгоритм и сравнить результаты его работы с теми, которые мы предоставили в этой главе. Подробно этот алгоритм описан в статье [Ordonez, C. and Cereghini, P.].

2. *Алгоритмы минимального остовного дерева (MST).* Мы представили реализацию алгоритма Прима–Ярника. Минимальные остовные деревья применимы не только к кластеризации, но и к задачам организации сетей и нахождения маршрута. В 1926 году Отакар Борувка (Otakar Boruvka) представил первый из известных алгоритмов вычисления дерева MST, в контексте оценки эффективности покрытия Моравии электрическими сетями!

   Учебник [Eisner, J.], доступный в Интернете, объясняет классические алгоритмы и представляет улучшенный подход Гарольда Габова (Harold Gabow), Цви Галила (Zvi Galil) и Томаса Спенсера (Thomas H. Spencer), а также рандомизированный алгоритм Дэвида Каргера (David R. Karger), Филиппа Кляйна (Philip N. Klein) и Роберта Тарьяна (Robert E. Tarjan). Последний выполняет только $O(m)$ вычислительных операций, где (m) – число ребер. Прочтите учебник [Eisner, J.] и дополните класс `MST` так, чтобы он поддерживал более эффективные алгоритмы, представленные в учебнике.

3. *ROCK: оценка ожидаемого числа связей.* Мы уже говорили, что алгоритм ROCK особенно хорош, когда дело касается категорийных и булевских значений атрибутов, и что он использует понятие связей вместо того, чтобы сравнивать непосредственно расстояния. Этот алгоритм находит «лучшие» кластеры, максимизируя значение *критерия качества*. Критерий качества выбирается произвольно; основная идея заключается в том, что мы используем для сравнительного отбора лучших кластеров связи, но конкретный вариант

реализации может иметь свои отличия. Мы в реализации использовали эвристику, которую предложили Раманатан Гуха (Ramanathan V. Guha) и др. Эта эвристика вычисляет критерий качества как отношение числа общих связей между двумя кластерами к ожидаемому числу общих связей между ними.

Что, по-вашему, является логическим обоснованием этого варианта? Какие еще оценки можем мы сформулировать для ожидаемого числа связей между двумя кластерами? Проанализируйте различные идеи и сравните результаты. Что произойдет, если мы просто примем, что критерий качества пропорционален числу общих связей между двумя кластерами?

4. Кластеризация больших наборов данных. В случае больших наборов данных часто желательно объединить рассмотренные в этом разделе методы, чтобы сбалансировать эффективность и высокое качество кластеризации. Одна из возможных гибридных схем – объединение алгоритма k-средних с алгоритмом ROCK (если в данных преобладают категорийные или булевские атрибуты) или с алгоритмом DBSCAN (если данные ссылаются на пространственные или другие метрические координаты, где имеет смысл и эффективен показатель расстояния).

   Что предпочли бы вы? Не забудьте, что алгоритм k-средних имеет лучшую производительность в смысле пространства и времени. Следовательно, имея большие мощности для обработки данных, можно прибегнуть к распараллеливанию. То есть вы могли бы использовать алгоритм k-средних для нескольких итераций и для получения небольшого числа высокоуровневых кластеров, которые затем обрабатывали бы алгоритмами ROCK или DBSCAN. Напишите реализацию для этой задачи и протестируйте полученные результаты на большом наборе данных (сотни тысяч точек данных). Можно задействовать документы на своем персональном компьютере или копию большой рабочей базы данных.

   Рассмотрите альтернативный вариант выборки из большого набора данных и ее кластеризации с помощью мощных алгоритмов, таких как ROCK или DBSCAN. Затем используйте число полученных кластеров в качестве значения для k и выберите центроиды кластеров выборки, чтобы начать итерации алгоритма k-средних. Сравните эти два подхода: какой из них обеспечил вам получение лучших кластеров (исследуйте это опытным путем, визуально анализируя данные)? Какой подход оказался более эффективным? Можете ли вы объяснить свои результаты с теоретической точки зрения?

4.10. Ссылки

- Aggarwal, C. C. «Towards Meaningful High-Dimensional Nearest Neighbor Search by Human-Computer Interaction», ICDE, 2002. *http://citeseer.ist.psu.edu/aggarwal02towards.html*.
- Arthur, D. and S. Vassilvitskii. «k-Means++: The advantages of careful seeding» Symposium on Discrete Algorithms (SODA)», 2007. *http://www.stanford.edu/~darthur/kMeansPlusPlus.pdf*.
- Beyer, K., R. Ramakrishnan, U. Shaft, J. Goldstein. «When is nearest neighbor meaningful?» ICDT Conference 1999.
- Bradley, P. S., U. Fayyad, and C. Reina. «Scaling clustering algorithms to large databases», Proc. 4th International Conference on Knowledge Discovery and Data Mining (KDD-98). AAAI Press, pp. 9–15.
- Dhillon, I. S., J. Fan, and Y. Guan. «Efficient clustering of very large document collections», Data Mining for Scientific and Engineering Applications, 2001. Kluwer Academic Publishers.
- Dhillon, I. S., S. Mallela, and R. Kumar. «A Divisive Information-Theoretic Feature Clustering Algorithm for Text Classification», Journal of Machine Learning Research 3 (March 2003).
- Eisner, J. «State-of-the-art algorithms for minimum spanning trees – A tutorial discussion», 1997. *http://citeseer.ist.psu.edu/eisner97stateart.html*.
- Kruskal, J. B. «On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem», Proceedings of the American Mathematical Society, Vol 7 (1), pp. 48–50. 1956.
- Ordonez, C. and Cereghini, P. «SQLEM: Fast clustering in SQL using the EM algorithm», ACM SIGMOD Rec. 29 (2), pp. 559–570, 2000.
- Prim, R. C. «Shortest connection networks and some generalizations», Bell System Tech. J. 1957.
- Sheikholeslami, G., S. Chatterjee, and A. Zhang. «WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases». In Proceedings of the 24th VLDB Conference. 1998.
- Tung, A. K. H., J. Hou, and J. Han. «Spatial clustering in the presence of obstacles». In Proceedings of the 17th ICDE, 359-367, Heidelberg, Germany. 2001.
- Ester M., Kriegel H.-P., Jorg S., Xu X. «A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise». In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996.

5

Классификация: размещение по принадлежности

- Основные понятия методов классификации – вероятностных и на основе правил
- Автоматическая классификация сообщений электронной почты
- Выявление мошеннических финансовых транзакций с помощью нейронных сетей

«Что это?» – вот вопрос, который дети задают чаще прочих. В популярности этого вопроса у детей, чья природная любознательность столь же замечательна, сколь и неистребима, нет ничего неожиданного. Чтобы понять окружающий мир, мы организуем свои представления в группы и категории (группы, снабженные меткой, возможно, структурированные). В предыдущей главе мы привели несколько алгоритмов *кластеризации*, помогающих объединить точки данных в группу. В этой главе мы познакомим вас с рядом алгоритмов *классификации*, помогающих определить для каждой точки данных соответствующую категорию, также называемую *классом* (отсюда и термин *классификация*). Классификация отвечает на детский вопрос предложением в форме: «это – лодка», «это – дерево», «это – дом» и так далее. Классификация опирается на априорные ссылочные структуры, позволяющие разбить пространство всех возможных точек данных на множество классов, которые обычно (но не обязательно) не пересекаются. Сопоставьте это с произвольной природой кластеров, рассмотренных в предыдущей главе.

Мы могли бы утверждать, что кластеризация как часть нашего мыслительного процесса предшествует классификации, поскольку ссылочные структуры, необходимые для классификации, значительно более содержательно представляют знания, нежели утверждение из разряда «*X* принадлежит той же группе, что и *Y*». Термин *онтология* (ontology) используется, как правило, для ссылочной структуры, которая является понятийным представлением о мире или о его части, интересующей наше приложение. К практической стороне классификации, которую обычно не рассматривают, относится обслуживание онтологии. Есть книги, посвященные исключительно вопросам *инженерии* (ontology engineering) и *управления онтологией* (ontology management): [Staab, S., and R. Studer], [Gómez-Pérez, A., M. Fernández-López, and O. Corcho].

В разделе 5.1, есть несколько примеров из реальной жизни, в которых классификация играет важную роль. Также там дано определение абстрактной онтологической структуры и проведена аналогия между онтологическими структурами и структурами Java-кода. Получив достаточную мотивацию, мы переходим к разделу 5.2, где даем обзор классификаторов. Понятно, что невозможно охватить в этой книге все известные классификаторы, но наш обзор поможет вам сориентироваться в соответствующей литературе.

В разделе 5.3 вы изучите наивный (naïve) байесовский алгоритм – один из самых выдающихся и известных алгоритмов классификации всех времен. Мы рассмотрим конкретный случай фильтрации сообщений электронной почты, содержащих спам, наряду с более общим случаем распределения сообщений электронной почты по нескольким целевым папкам. Этот пример хорошо иллюстрирует классификацию неформатированного текста с помощью статистического алгоритма классификации.

Однако наиболее распространенными алгоритмами классификации сообщений электронной почты являются алгоритмы на основе правил. В разделе 5.3.2 рассматривается классификация электронной почты с точки зрения *процессора правил* (rules engine). Мы знакомим вас со всеми важными понятиями и демонстрируем применение правил с помощью процессора правил Drools (JBoss). В разделе 5.4 задача обнаружения случаев мошенничества решается как задача классификации. В этом контексте будет представлен еще один широко используемый классификационный подход – классификация посредством *нейронных сетей*.

Как понять, что мы назначили точке данных самый подходящий класс? Откуда нам знать, что классификатор *A* лучше классификатора *B*? Если вам когда-либо доводилось читать брошюры об инструментах бизнес-анализа, вы, возможно, уже знакомы с утверждениями вроде следующего: «Правильность нашего классификатора – 75%». В чем смысл этого

утверждения? Полезно ли оно? Эти вопросы получат ответ в разделе 5.5. Мы рассмотрим классификацию больших объемов точек данных, классификацию с учетом очень больших онтологических структур и быструю классификацию в оперативном режиме. Каждая из этих трех категорий, не являющихся взаимоисключающими, требует особого внимания и нередко встречается в приложениях на практике.

Начнем с обсуждения возможностей применения классификации и представления технических терминов, которые будут сопутствовать нам и дальше. Итак, для чего нужна классификация? Какие практические задачи она для нас решает?

5.1. Необходимость классификации

Осознанно или нет, мы сталкиваемся с классификацией ежедневно. Типичный пример из жизни – ресторанное меню, где блюда классифицированы по категориям: салаты, закуски, фирменные блюда, блюда из макарон, морепродукты и так далее. Статьи в газете или в интернет-конференции классифицированы по темам: политика, спорт, бизнес, международные новости, развлечения и так далее.

На книгах в библиотеке указывается *шифр* (call number), состоящий из двух номеров: *номера классификации Дьюи* (Dewey classification number) и *номера Каттера* (Cutter number). Категориями верхнего уровня в этой системе являются общие понятия, такие как религия, естественные науки и математика и так далее. Для организации и упорядочения коллекции книг библиотеки Конгресса Соединенных Штатов используется собственная система классификации, разработанная в конце XIX – начале XX века.

На протяжении XX века система библиотеки Конгресса была также адаптирована для других библиотек, особенно для крупных академических библиотек Соединенных Штатов. Мы упоминаем эти две системы классификации книг, потому что система библиотеки Конгресса не является столь строго иерархической, как система Дьюи, где иерархические связи между темами отражаются в классификационных номерах. Как мы увидим, важно различать иерархические и неиерархические ссылочные структуры.

Есть немало медицинских классификационных систем, применяемых для диагностики травм или заболеваний. Например, рентгенологи и хирурги-ортопеды классифицируют переломы верхней суставной поверхности большеберцовой кости (сложное повреждение колена) с помощью системы классификации Шацкера. Аналогично, есть системы классификации повреждений спинного мозга, коматозных состояний, сотрясений, травматических повреждений головного мозга и так далее.

Справочник «Классификация производственных травм и заболеваний» (Occupational Injury and Illness Classification, OIIC) обеспечивает систему

классификации для кодирования признаков травматических повреждений и заболеваний, а также смертельных случаев в отчете о производственных травмах и заболеваниях (Survey of Occupational Injuries and Illnesses, SOII) и в докладе о наиболее опасных профессиях (Census of Fatal Occupational Injuries, CFOI), одобренных правительством США. Документ ICD-10[1] Всемирной организации здравоохранения (World Health Organization, WHO) был одобрен на 43-й Всемирной ассамблее здравоохранения в мае 1990 года, а с 1994 года вступил в действие в государствах-членах. Он предназначен для классификации заболеваний и других проблем, связанных со здоровьем, которые регистрируются в разного рода отчетах о состоянии здоровья и жизни, включая свидетельства о смерти и карты стационарных больных. После того как вы побывали в кабинете у врача, именно к этому документу обращается ваша страховая компания, чтобы определить сумму страховки. На верхнем уровне это такие категории, как некоторые инфекционные и паразитарные заболевания, новообразования, заболевания эндокринной системы, диетологические и заболевания, связанные с обменом веществ, и так далее. В биологии система Линнея использует два атрибута для классификации всех живых существ – *род* (genus) и *вид* (species). Вам, скорее всего, знакомо выражение «Homo sapiens»[2], где Homo – это наш род, а sapiens – вид. Эта классификация может, и обычно дополняется другими атрибутами, такими как *семейство* (family), *отряд* (order), *класс* (class), *тип* (phylum) и так далее.

Отвлечемся, чтобы предупредить вас об опасности, связанной с количеством атрибутов. Вообще говоря, чем больше атрибутов вы используете, тем точнее должна быть классификация. «Обширная» номенклатура атрибутов обычно вещь хорошая, но из этого правила есть исключения. Один из печально известных симптомов того, что вы имеете дело с очень большим числом атрибутов, – *проклятие размерности*, которое рассматривалось в разделе 4.6.2. Обычно о большом числе атрибутов говорят, если их больше шестнадцати.

Как вы, возможно, помните, проклятием размерности называют явление, когда по мере возрастания числа атрибутов пространство становится все более и более однородным. Другими словами, какие бы точки вы ни выбрали и какую бы метрику ни применили для измерения расстояний, расстояние между любыми двумя точками остается примерно тем же. А раз так, то становится все труднее отличить, какая категория располагается «ближе» всех к конкретной точке данных, поскольку, где бы вы ни «стояли» в нашем пространстве, все представляется

---

[1] International Statistical Classification of Diseases and Related Health Problems, IDC-10 – в России этот документ называется «Международная статистическая классификация болезней и проблем, связанных со здоровьем». В настоящее время действует «Международная классификация болезней Десятого пересмотра» (МКБ-10). – *Прим. перев.*

[2] Человек разумный. – *Прим. перев.*

равноудаленным! Вы всегда можете добавить в свою онтологию атрибуты, с тем чтобы получить возможность сосредоточить все знания вашей предметной области в одном месте, а затем выбрать атрибуты, которые должны использоваться для классификации.

Из рассмотренных примеров должно быть понятно, что *плоские ссылочные структуры* (flat reference structures) не так «богаты», как *иерархические ссылочные структуры* (hierarchical reference structures). В свою очередь, иерархические ссылочные структуры менее содержательны, нежели структуры, которые являются иерархическими *и* обогащены семантикой. Это наблюдение опять-таки укладывается в рамки обсуждения онтологий. Мы не даем четкое определение понятия «онтология», поскольку непохоже, что по этому вопросу достигнуто единое мнение.

В настоящей книге под онтологией понимается совокупность трех составляющих: *концептов* (concepts), *объектов* (instances) и *атрибутов* (attributes).

На рис. 5.1 мы изобразили очень небольшой сегмент (элементарной) общей онтологии, уделив основное внимание концепту «транспортное средство». Концепты отображаются эллипсами, объекты – прямоугольниками, а атрибуты – прямоугольниками с закругленными углами. Обратите внимание на свойство, состоящее в наследовании назначенных атрибутов. Если атрибут 1 назначен корневому узлу дерева концептов, то он каскадом «спускается» до листовых узлов концепта. Так, значения для атрибута 1 могут быть назначены объектам «лодка» и «автомобиль». Но только объект «автомобиль» может иметь значение для атрибута 2. Атрибутом 1 мог бы быть атрибут «Марка», который всегда нужен по практическим соображениям, а атрибутом 2 мог бы быть атрибут «Количество колес». Атрибуты определяются на уровне концептов, но только объекты имеют конкретные и уникальные значения, поскольку только объекты представляют реальные «вещи».

Считайте, что концепты – это аналоги классов языка программирования Java, объекты являются аналогами экземпляров Java-классов, а атрибуты – это переменные Java-классов. Понятно, что база исходного программного кода, использующая пакеты, чтобы сгруппировать классы по функциональному признаку, или компонент, который использует наследование, чтобы абстрагировать общие структуру и поведение, и надлежащим образом применяет инкапсуляцию, занимают более высокую ступень, нежели база исходного кода, не обладающая названными качествами. По аналогии с атрибутами в данном нами определении онтологии, определяя класс, вы указываете типы данных для переменных, но не присваиваете переменной значение (если только она не является константой). Это хорошее рабочее определение, которое успешно послужит вам в 80–90% случаев. Если у вас когда-либо возникнут сомнения, обращение к данной аналогии поможет вам прояснить свою структуру.

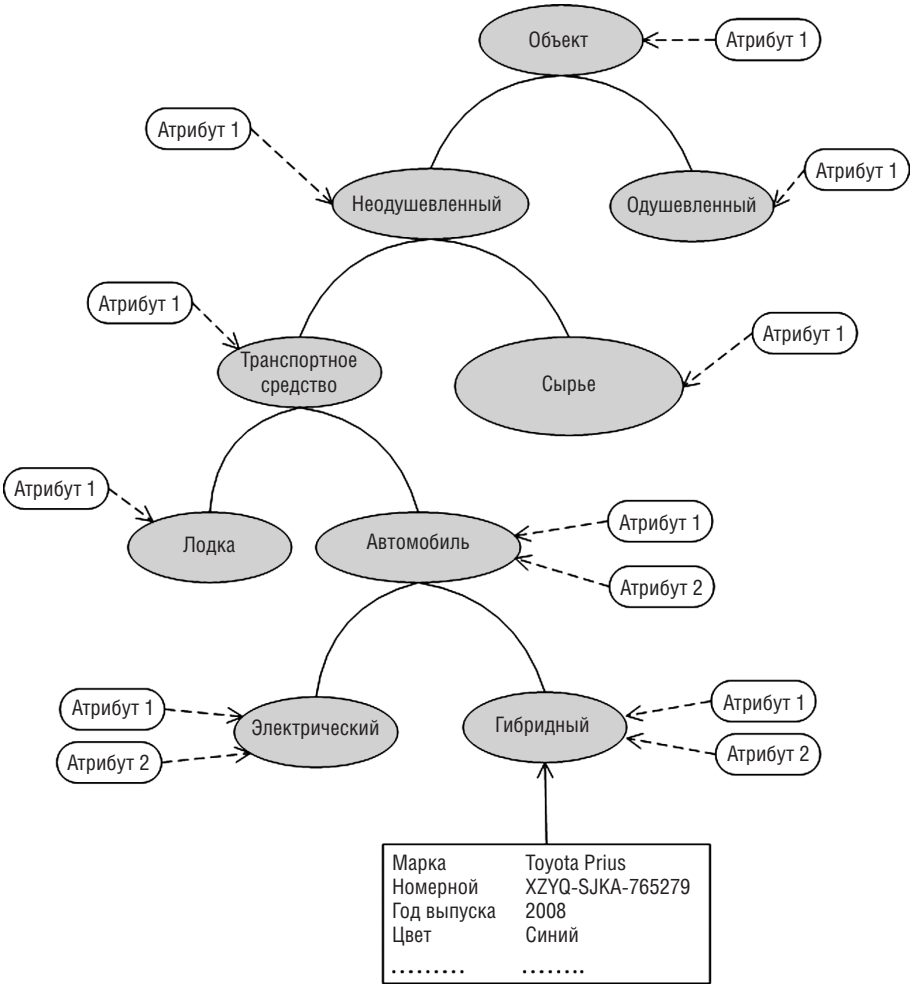

Рис. 5.1. *Пример, демонстрирующий базовые элементы ссылочной структуры (рудиментарная онтология)*

Безусловно, мы могли бы продолжить приводить примеры систем классификации – они вездесущи. По сути классификация данных равнозначна их структуризации или организации. Система классификации улучшает взаимодействие, сокращая ошибки, возникающие из-за неопределенности. Она также помогает организовать мысли и планировать свои действия. Ссылочная структура, применяемая для организации данных, может быть простой, как набор меток, или развитой, как *семантическая онтология* (semantic ontology). Вы слыхали о «Семантической паутине» (Semantic Web)? Ядро «Семантической паутины» [Antoniou, G., and F. van Harmelen] образуют технологии и формальные

спецификации, обеспечивающие создание, использование и обслуживание семантических онтологий. Онтологии полезны также в архитектурах, управляемых моделями [Gasevic, D., D. Djuric, V. Devedzic], – это проектная инициатива в области разработки программного обеспечения рабочей группы, которая занимается объектно-ориентированными технологиями и стандартами (Object Management Group, OMG) (*http://www.omg.org/*).

Взгляните на свою базу данных. Ваше приложение могло бы обслуживать интернет-магазин, систему управления документами в сети интранет, гибридный интернет-портал с неоднородным контентом или быть веб-приложением любого другого рода. Осмыслив свои данные и возможные способы их организации, вы поймете значение системы классификации для вашего приложения. Начиная с раздела 5.3 и далее, включая большую часть главы 6, мы будем внедрять механизмы классификации в гипотетические веб-приложения, демонстрируя применение алгоритмов классификации и те проблемы, которые могут при этом возникнуть. Но для начала сделаем обзор систем классификации. Если хотите поскорее перейти к действию, можете пропустить следующий раздел.

5.2. Обзор классификаторов

Чтобы представить множество всех систем классификации, можно рассмотреть эти системы с точки зрения используемых ими ссылочных структур. На верхнем уровне такого построения все системы классификации можно разбить на две главные категории – *двоичные* (binary) и *многоклассовые* (multiclass) системы. Двоичные системы классификации, как предполагает их название, отвечают только «да» или «нет» на вопрос: принадлежит ли эта точка данных классу X? Система медицинской диагностики может ответить на вопрос, есть ли у пациента раковое заболевание. А классификационная система миграционной службы может ответить, является ли некий человек террористом. Многоклассовые системы классификации приписывают точку данных к одному конкретному классу из нескольких, например, относят новостную статью к категории новостей.

Многоклассовые системы классификации можно было бы далее сгруппировать по двум критериям: являются ли разнообразные классы дискретными или непрерывными либо являются ли они «плоскими» (просто список меток) или имеют иерархическую структуру. Рассмотренные в предыдущем разделе схемы Дьюи и ICD-10 – это примеры систем классификации, включающих несколько дискретных конечных классов. Но результатом классификации может быть и непрерывная величина, как, например, в случае классификации предсказаний, процесс получения которых еще называют *прогнозированием* (forecasting). Если вы предоставляете в качестве входных данных стоимость акций

в понедельник, вторник, среду и четверг и хотите выяснить, сколько они будут стоить в пятницу, можно рассматривать эту задачу как случай многоклассовой классификации, дискретной или непрерывной. Дискретная версия могла бы предсказывать, возрастет ли стоимость акций, останется прежней или упадет в пятницу. Непрерывная версия могла бы предсказывать фактическую стоимость акций.

Категоризация систем классификации с учетом базовой методики не так прозрачна или общепринята. Но мы могли бы сказать, что есть две основные категории, которые весьма широко применяются в производстве. Первая категория включает *статистические алгоритмы* (statistical algorithms), а вторая – *структурные алгоритмы* (structural algorithms), как показано на рис. 5.2.

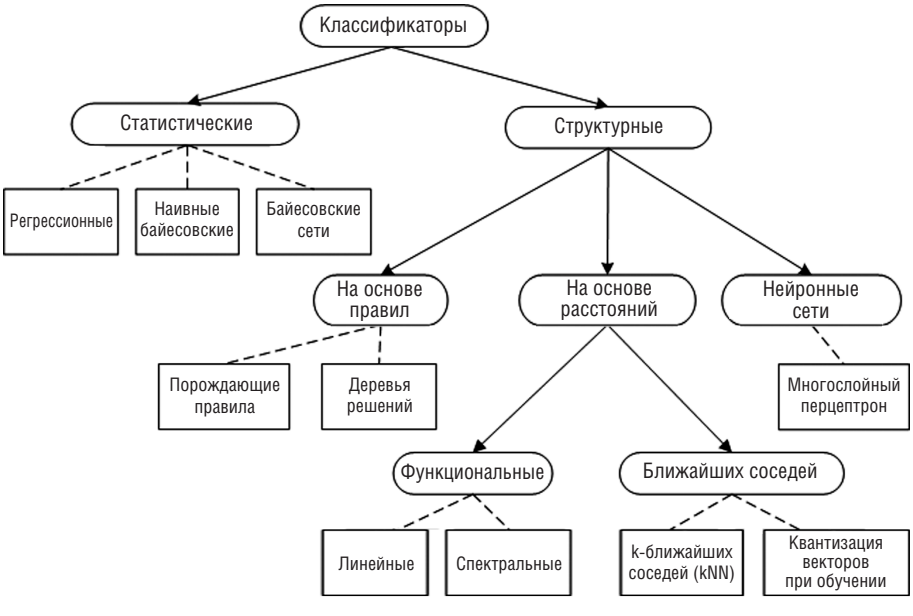

Рис. 5.2. *Обзор алгоритмов классификации по их устройству*

Статистические алгоритмы представлены тремя разновидностями. *Регрессионные алгоритмы* (regression algorithms) особенно подходят для прогнозирования – предсказания значения непрерывной величины. Регрессионные алгоритмы предполагают, что достаточно приспособить наши данные к конкретной модели; зачастую такой моделью является линейная функция имеющихся переменных. Отправной точкой для другого рода статистических алгоритмов классификации является теорема Байеса, с которой мы вкратце познакомились в главе 2. Довольно успешный и современный статистический подход объединяет теорему Байеса с вероятностной сетевой структурой, которая отображает зави-

симость между различными атрибутами рассматриваемой проблемы классификации.

Структурные алгоритмы имеют три главных ответвления: алгоритмы *на основе правил* (rule-based), которые включают правила типа «если–то» и деревья решений; алгоритмы *на основе расстояний* (distance-based), которые обычно подразделяют на функциональные схемы и схемы ближайших соседей; и *нейронные сети* (neural networks, NN). Нейронные сети образуют отдельную категорию, хотя мы должны упомянуть, что установлена и хорошо изучена эквивалентность конкретных нейронных сетей некоторым развитым статистическим алгоритмам (гауссовым процессам). В следующих разделах мы вкратце охарактеризуем каждую из этих категорий классификаторов.

5.2.1. Алгоритмы структурной классификации

Как показано на рис. 5.2, структурные алгоритмы на основе правил делятся на *порождающие правила* (production rules) (операторы `if-then`) и алгоритмы, использующие *деревья решений* (decision tree, DT). Порождающие правила могут быть получены вручную, у экспертов-людей, или выведены с помощью деревьев решений. Алгоритмы на основе правил обычно реализуются как системы, порождающие *прямую цепочку рассуждений* (forward chaining), – жуткий термин, по-моему! Лучший алгоритм из этой категории называется Rete [Russell, S., and P. Norvig]; *rete* по-латыни означает «сеть». Этот алгоритм является основой известных библиотек, таких как CLIPS, Jess и Soar.

В книге мы будем использовать объектно-ориентированную реализацию алгоритма Rete, которую предлагает проект JBoss. Речь идет о библиотеке JBoss Rules, известной также как Drools (оригинальное название проекта этого процессора правил). Это стабильный проект, он подробно документирован и его базовый программный код в равной степени пригоден как для изучения, так и для разработки. В этой связи следует заметить, что нам так же понравилось работать с интерфейсами Drools API, как и с интерфейсами Lucene API, рассмотренными в главе 2. Эти два проекта действительно готовы к применению в производстве.

Алгоритмы, использующие деревья решений, основаны на простой, но эффективной идее. Вы читали повесть Чарльза Диккенса «Рождественская песнь»? В ней Диккенс описывает игру («Да и нет»), в которой племянник Скруджа что-то задумывает, а остальные угадывают, что он задумал, задавая вопросы, на которые можно ответить только «да» или «нет». Варианты этой игры есть во многих культурах; она довольно популярна среди детей в испано-говорящих странах, где ее называют *veo, veo*[1]. Как и в этой известной игре, основная идея большинства

[1] Вижу, вижу (исп.). – *Прим. перев.*

DT-алгоритмов заключается в том, чтобы задавать вопросы, ответы на которые позволят отсечь максимально возможное число кандидатов, исходя из предоставленной информации. У алгоритмов на основе деревьев решений есть преимущества, такие как легкость использования и эффективность вычислений. Недостатки этих алгоритмов обычно бросаются в глаза, когда мы имеем дело с непрерывными переменными, поскольку для того, чтобы построить дерево, приходится выполнить дискретизацию – разбить континуум значений на конечное множество дискретных элементов. В общем случае алгоритмы деревьев решений не обладают хорошими способностями к обобщению и в результате плохо справляются с данными, которые ранее не встречались. Из этой категории широко используются алгоритмы C5.0 (на компьютерах с ОС UNIX) и See5 (на компьютерах с ОС Microsoft Windows). Эти алгоритмы можно обнаружить в нескольких коммерческих продуктах, например в системах анализа и обработки данных Clementine (*http://www.spss.com/clementine/*) и RuleQuest (*http://www.rulequest.com/*).

Вторую ветвь структурных алгоритмов образуют алгоритмы на основе расстояния. В предыдущих главах мы представили и интенсивно использовали понятия меры сходства и обобщенного расстояния. Структурные алгоритмы, до некоторой степени, интуитивно понятны, но их легко можно использовать не по назначению и получить в итоге неверные результаты классификации, поскольку множество атрибутов точек данных не связаны напрямую друг с другом. Одной только меры сходства недостаточно, чтобы адекватно отобразить различия в способах измерения атрибутов; для успешной работы алгоритмов на основе расстояния крайне важны тщательная нормализация и анализ пространства атрибутов. Тем не менее во многих случаях с малой размерностью и низким уровнем сложности эти алгоритмы хорошо работают и довольно легко реализуемы. Продолжив, можно разбить алгоритмы на основе расстояния на функциональные и алгоритмы ближайших соседей.

Функциональные классификаторы, как и предполагает их название, аппроксимируют данные с помощью функции. Благодаря использованию функции есть сходство с регрессией, но мы различаем эти два случая исходя из логического обоснования применения функции. В регрессии мы используем функцию как модель вероятностного распределения [Dunham, M. H.], в случае функциональных классификаторов мы заинтересованы только в численной аппроксимации данных. На практике трудно (и скорее всего бессмысленно) отделить линейную регрессию от линейной аппроксимации посредством минимизации квадратичной ошибки.

Алгоритмы ближайших соседей пытаются найти для каждой точки данных ближайший к ней класс. Используя формулы, которые мы уже приводили в связи с обобщенными расстояниями, можно вычислить, на каком расстоянии находится каждая точка данных от каждого до-

ступного класса. Тот класс, который находится ближе всех к объекту, назначается этому объекту. Пожалуй, самым распространенным алгоритмом классификации этого типа является алгоритм k-ближайших соседей (kNN), хотя другой алгоритм, известный как квантизация векторов при обучении (learning vector quantization, LVQ), также хорошо изучен и широко применяется.

Алгоритмы нейронных сетей (Neural Networks, NN) принадлежат к подкатегории структурных алгоритмов как таковых. Надлежащее представление этих алгоритмов требует серьезной математической подготовки. Мы сделаем все, что в наших силах, чтобы представить их с точки зрения вычислений, не обращаясь к математике. Основная идея, образующая фундамент для этого семейства алгоритмов классификации, – это построение искусственной сети вычислительных узлов, аналогичной биологической структуре человеческого мозга, которая, по сути, состоит из *нейронов* (neurons) и связывающих их *синапсов* (synapses).

Алгоритмы нейронных сетей хорошо работают при решении различных задач. У нейронных сетей есть два главных недостатка: отсутствие единой методологии проектирования, которую можно было бы применить для решения широкого спектра задач; результаты классификации, полученные с помощью нейронных сетей, трудно интерпретировать, поскольку классификатор по неустановленным причинам может совершить несколько ошибок. Вот почему мы рассматриваем нейронные сети как методику «черного ящика», в противоположность алгоритмам на основе деревьев решений или правил, где результат классификации для конкретной точки данных может быть легко интерпретирован.

5.2.2. Статистические алгоритмы классификации

Основная идея регрессионных алгоритмов состоит в нахождении данных, наиболее соответствующих формуле; общепринятой формулой является линейная функция входных значений [Hastie, T., R. Tibshirani, and J. Friedman]. Обычно регрессионные алгоритмы применяются, когда точки данных по сути являются числовыми величинами (такими как размеры объекта, вес конкретного человека или температура воздуха), но в отличие от алгоритмов Байеса, они не очень подходят для категорийных данных (таких как статус служащего или описание кредитного балла). Кроме того, если модель, описывающая данные, является линейной, трудно оправдать прилагательное «статистические»: в сущности, линейная регрессия не отличается от старого доброго упражнения из курса средней школы, когда требуется построить график для скопления точек (x, y).

Гораздо интереснее и не лишен статистического подхода случай так называемой *логистической регрессии* (logistic regression). В этом случае модель (логистическая функция) получает значения из интервала от 0 до 1, которые могут быть интерпретированы как вероятность принад-

лежности к классу и хорошо работать в случае двоичной классификации [Dunham, M. H.].

В большинстве методик, применяемых в алгоритмах из категории статистических, используется теорема из теории вероятностей, известная как *правило* (Bayes rule), или *теорема Байеса* (Bayes theorem) [Papoulis, A., and S. U. Pillai]. Мы встречались с правилом Байеса в главе 2, в контексте обучения на основе переходов пользователя по ссылкам. В статистических алгоритмах классификации такого рода наименьшим общим знаменателем является предположение о независимости атрибутов задачи друг от друга, достаточно явно выражаемой количественно. Прелесть байесовых алгоритмов в том, что они, по-видимому, хорошо работают, даже если предположение о независимости с очевидностью нарушается! В разделе 5.3 мы будем изучать самый знаменитый алгоритм, где применяется этот подход, – наивный байесовский алгоритм классификации.

Байесовские сети – довольно современный подход к машинному обучению, который пытается объединить широкие возможности теоремы Байеса с преимуществами структурных подходов, таких как на основе деревьев решений. Наивные байесовские классификаторы и родственные им алгоритмы могут предоставить простые вероятностные распределения, но неспособны отобразить вероятностную структуру данных, если таковая имеется. Вероятностные связи атрибутов можно графически отобразить с помощью мощного представления, обеспечиваемого *направленными ациклическими графами* (directed acyclic graphs, DAG). В этой книге мы не рассматриваем байесовские сети; для более глубокого изучения этой темы обратитесь к книге [Neapolitan, R. E.].

5.2.3. Жизненный цикл классификатора

Какого бы типа классификатор вы ни выбрали для своего приложения, его жизненный цикл будет соответствовать общей схеме, показанной на рис. 5.3. Жизненный цикл классификатора включает три этапа: обучение, тестирование и эксплуатация.

На этапе обучения классификатору предоставляются точки данных, которым мы уже назначили соответствующий класс. Каждый классификатор содержит несколько параметров, которые должны быть определены перед его использованием. Назначение этапа обучения – определить эти различные параметры; мы поместили внутрь звездочки знак вопроса, поскольку первостепенной задачей является определение параметров. На этапе тестирования мы проверяем классификатор, чтобы перед тем, как внедрять классификатор в действующую систему, убедиться в том, что нам удалось достичь определенного уровня доверия к получаемым результатам. Буква E в окружности символизирует то, что главной задачей является определение ошибки (error) классификатора, при этом уровень качества может и должен быть измерен

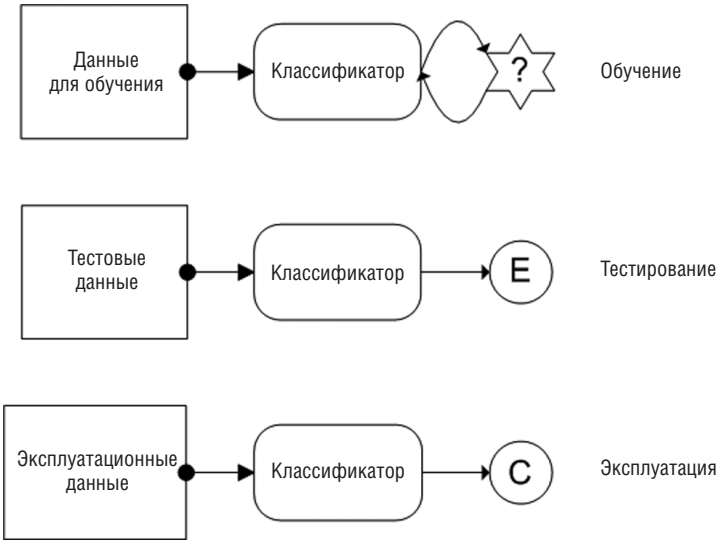

Рис. 5.3. *Жизненный цикл классификатора: обучение, тестирование, эксплуатация*

с помощью различных метрик (вопросы степени доверия и стоимости классификации обсуждаются в разделе 5.5). Данные, используемые на этапе тестирования (тестовые данные), должны отличаться от использованных на этапе обучения (данные для обучения).

Прежде чем классификатор перейдет на этап эксплуатации, можно неоднократно повторять этапы обучения и тестирования, поскольку могут иметься конфигурационные параметры, не идентифицируемые в процессе обучения, но предоставляемые в качестве входных данных на этапе проектирования классификатора. Эта важная мысль означает, что мы можем написать программное обеспечение, которое бы служило оболочкой для данного классификатора и его конфигурационных параметров, чтобы автоматизировать тестирование и проверку множества проектных решений для классификатора. В таком тестировании могли бы участвовать даже классификаторы принципиально иной природы, например наивный байесовский алгоритм, нейронная сеть и деревья решений. Мы можем либо выбрать лучший классификатор, выявленный по показателям качества на этапе тестирования, либо объединить все классификаторы в то, что можно было бы назвать схемой *метаклассификатора* (metaclassifier). Этот подход, успешно применяемый в производстве, неизменно обеспечивал лучшие результаты широкому спектру приложений. Объединение классификаторов рассматривается в главе 6.

На этапе эксплуатации мы применяем классификатор в действующей системе для проведения классификации в оперативном режиме. Как правило, на этапе эксплуатации параметры классификатора не меняются. Но есть возможность повысить качество результатов работы классификатора за счет встраивания (на этапе эксплуатации) краткосрочного этапа обучения, основанного на обратной связи с участвующим в цикле человеком. Эти три шага повторяются по мере получения новых данных от рабочей системы и предложения более удачных идей для нашего классификатора.

Теперь у нас есть общее представление об алгоритмах классификации. В литературе можно обнаружить множество других обзоров этих алгоритмов [например, Holmström, L., P. Koistinen, J. Laaksonen, and E. Oja]. В следующем разделе мы представим один из самых знаменитых статистических алгоритмов классификации – наивный байесовский классификатор. В частности, мы покажем, как применять классификацию для распределения допустимых сообщений электронной почты по категориям и для фильтрации спама.

5.3. Автоматическая категоризация почтовых сообщений и фильтрация спама

В этом разделе мы преследуем две цели. Первая – «крупноблочная»: получить возможность отличать допустимые сообщения электронной почты от спама, что является примером *двоичной классификации* (binary classification). Вторая цель – добиться большей степени детализации при сортировке сообщений электронной почты; мы хотим уточнить результаты классификации и разбить сообщения электронной почты, не являющиеся спамом, по следующим категориям: бизнес, международная почта, США и спорт. Эта задача является примером *многоклассовой классификации* (multiclass classification).

Электронная почта не требует особого представления. Это одно из первых приложений, появившихся в эпоху Интернета, и, пожалуй, самое распространенное сегодня. У большинства пользователей все сообщения помещаются прямо в папку *Входящие* (inbox) системы электронной почты. Было бы неплохо создавать собственные папки и чтобы сообщения электронной почты (автоматически) «раскладывались» бы по надлежащим папкам, верно? Можно было бы завести папку со значком «геенна огненная» и прямиком направлять в нее все сообщения, содержащие спам, называемый также *несанкционированной массовой рассылкой* (unsolicited bulk email).

Ваше почтовое клиентское приложение, скорее всего, уже делает это. Большинство нынешних почтовых клиентов реализуют – по крайней мере, в некоторой форме – классификацию на основе правил. По этой

причине большинство таких клиентов не очень хорошо обучаются, то есть не обобщают «ранее виденные» вручную распределенные по категориям сообщения электронной почты. И здесь у веб-клиентов электронной почты есть масса возможностей, поскольку алгоритмы, способные к обобщениям, могут быстро охватить намного более широкий диапазон сообщений электронной почты, например новые разновидности сообщений со спамом.

Алгоритмы, которые мы здесь рассмотрим, применимы к произвольной коллекции документов. Вы можете использовать их в приложениях, позволяющих пользователю загружать свои Word- и PDF-файлы с возможностью автоматической категоризации (еще один маркетинговый термин для классификации) этих документов по списку категорий, предоставленному пользователем.

В нашем примере для этого раздела коллекция сообщений электронной почты формируется из тех же файлов веб-страниц, которые мы использовали в главе 2 для поиска. Для каждой веб-страницы из главы 2 мы создали соответствующее ей сообщение электронной почты. Если вы прочли главу 2, то этот контент вам уже знаком. Если вы ее еще не читали, мы должны сказать, что наши сообщения электронной почты включают (выбор контента носил случайный характер, временной контекст – ноябрь 2006 года):

- Семь сообщений относятся к новостям бизнеса: три связаны с проникновением фирмы Google в сферу газетной рекламы, еще в трех обсуждаются, главным образом, акции фирмы NVidia, а одно сообщение содержит информацию о стоимости акций и динамике индексов.
- Три сообщения касаются попытки Лэнса Армстронга пробежать марафон в Нью-Йорке.
- Четыре сообщения связаны с политикой США , в частности с выборами в конгресс.
- Пять сообщений относятся к международным новостям: в четырех говорится о победе Ортеги на выборах в Никарагуа и в одном – о глобальном потеплении.
- Четыре сообщения являются спамом.

Классификация электронной почты представляет интерес во многих отношениях. Одна из особенностей здесь – наличие «кого-то», кто пытается переиграть ваш классификатор. Фактически, спамеры могут использовать описанные в этой книге методики, чтобы победить вашу схему классификации электронной почты, так что будьте начеку в этом противоборстве! Мы намеренно используем термин *схема* (scheme). Вряд ли, создавая приложение для фильтрации электронной почты или компьютерного «секретаря» электронной почты, вы ограничитесь только классификатором. Вот список дополнений к классификатору:

- Тесты заголовков
- Белый и черный списки автоматически сохраняемых адресов электронной почты
- Белый и черный списки адресов электронной почты, вводимых вручную
- Совместно используемые базы данных для идентификации спама
- Списки-«карцеры» реального времени (real-time blackhole lists, rbl)
- Набор символов и идентификация местонахождения

Нас, в основном, привлекают интеллектуальные аспекты систем классификации электронной почты, поэтому мы не будем рассматривать остальные упомянутые методики. Пример полезного модуля – проект ApacheSpam Assassin, расположенный по адресу *http://spamassassin.apache.org/*. Мы представим два метода классификации, которые вы можете использовать для создания подобных систем. Первый метод классификации опирается на наивный байесовский классификатор, представленный в главе 2. Второй метод классификации основан на правилах, мы используем процессор правил Drools.

5.3.1. Наивная байесовская классификация

В этом разделе мы применим статистический классификатор, инкапсулированный в классе `EmailClassifier`. Как уже говорилось, этот классификатор использует так называемый наивный байесовский алгоритм, расширяя возможности класса `NaiveBayes`, который является универсальной реализацией этого алгоритма. В общем случае классификаторы не зависят от объектов классификации: их интересуют только концепты `Concept`, образцы `Instance` и атрибуты `Attribute`. Работа классификатора заключается в назначении концепта образцу – вот и все, что он делает. Чтобы понять, какой концепт следует назначить конкретному образцу, классификатор считывает `TrainingSet` – набор образцов, которым уже назначены концепты. После загрузки образцов классификатор самостоятельно *обучается* (trains), иначе говоря *узнает* (learns), как отображать концепт на образец с помощью назначений, представленных в наборе `TrainingSet`. Способ, используемый каждым классификатором для обучения, зависит от самого классификатора. В этой главе мы используем понятия концепта, класса и категории как взаимозаменяемые.

Выборка сообщений электронной почты и обучение классификатора

Начнем с того, что продемонстрируем, как можно загрузить сообщения электронной почты и обучить соответствующий классификатор. В листинге 5.1 приведен сценарий среды BeanShell, который вы можете

выполнить, чтобы загрузить сообщения электронной почты и провести обучение классификатора.

Листинг 5.1. Загрузка обучающего набора сообщений электронной почты и обучение классификатора NaiveBayes

```
EmailDataset trainEmailDS = EmailData.createTrainingDataset();  ❶

trainEmailDS.printEmail("biz-04.html");
trainEmailDS.printEmail("spam-biz-03.html");  ❷

EmailClassifier emailFilter = new EmailClassifier(trainEmailDS, 10);  ❸

emailFilter.train();  ❹
```

❶ Класс `EmailData` отвечает за загрузку HTML-файлов, которые мы использовали в главе 2, и их трансляцию в экземпляры класса `Email` – простого класса, инкапсулирующего сообщение электронной почты на основе атрибутов `from`, `to`, `subject` и `textBody`. Метод `createTrainingDataset()` обеспечивает загрузку списка документов, которые мы хотим использовать для обучения классификатора. Этот список хранится в двумерном массиве `TRAINING_DATA` типа `String`; набор данных для тестирования хранится в двумерном массиве `TEST_DATA` типа `String`. Вы можете изменить содержимое этих списков и наблюдать, как это изменение повлияет на результаты классификации. Чтобы оценка была честной, для обучения и для тестирования следует использовать разные наборы файлов.

❷ На этом шаге выполняется печать содержимого двух сообщений электронной почты – одного допустимого и одного со спамом – просто для того, чтобы перед тем, как продолжать обработку, убедиться в том, что были загружены надлежащие данные, и оценить, с какого рода содержимым мы работаем.

❸ Здесь мы создаем экземпляр класса `EmailClassifier`, передавая ему в качестве параметров объектную ссылку на набор данных `EmailDataset` и число термов, которые должны учитываться при анализе сообщений электронной почты. Для каждого сообщения мы анализируем его содержимое и сохраняем первые 10 (в данном примере) наиболее часто встречающихся термов.

❹ Обучаем классификатор. Эта «санитарная» проверка нужна для того, чтобы удостовериться в наличии образцов, на которых можно обучать классификатор. В рамках этой проверки указываются атрибуты, на которых мы хотим обучать классификатор; вызывается метод `train()` родительского класса `NaiveBayes`; и задается случайный уровень вероятности для значений атрибутов, которые нам еще не встречались.

Классификатор электронной почты в действии

Получив обученный классификатор, можно его протестировать. Листинг 5.2 является продолжением листинга 5.1; следовательно, вам надо

выполнить его в том же самом сеансе оболочки (shell). Обратите внимание, как легко использовать классификатор на этом уровне. Для этого требуются буквально две строки программного кода!

Листинг 5.2. Использование наивного байесовского классификатора для обнаружения сообщений электронной почты, содержащих спам

```
EmailDataset testEmailDS =
➥  EmailData.createTestDataset();

email = testEmailDS.findEmailById("biz-01.html");
emailFilter.classify(email);

email = testEmailDS.findEmailById("sport-01.html");
emailFilter.classify(email);

email = testEmailDS.findEmailById("usa-01.html");
emailFilter.classify(email);

email = testEmailDS.findEmailById("world-01.html");
emailFilter.classify(email);

email = testEmailDS.findEmailById("spam-biz-01.html");
emailFilter.classify(email);
```

Загрузка сообщения электронной почты из набора тестовых данных

Извлечение электронной почты по имени файла

Классификация допустимой электронной почты

Результаты показаны на рис. 5.4. Обратите внимание: все сообщения электронной почты классифицированы правильно. Это создает базу для экспериментов с настройками и позволяет сравнить, как меняется правильность результатов классификатора по мере того, как вы наращиваете или сокращаете набор для обучения.

Заметим, что успешно классифицировать сообщения электронной почты легко, если все ранее не встречавшиеся сообщения похожи на те, которые имеются в наборе для обучения. В общем случае, если набор для обучения очень похож на набор для тестирования, можно получить высокий уровень правильности результатов. Как правило, этим мы обязаны *переобучению* (overfitting); подробнее о компромиссе между специализацией и обобщением говорится в п. 3 раздела «Сделать».

Тут может быть полезно повторить указанные шаги, изменив число часто встречающихся термов (см. листинг 5.1, шаг 3) и повторив обучение классификатора, и понаблюдать, как такое изменение влияет на результаты классификации. Сразу все шаги процесса классификации, представленные в листинге 5.2, можно выполнить, обратившись к методу `sample()` класса `EmailClassifier`.

Как мы сказали, работа классификатора заключается в назначении концепта образцу; для классификатора `EmailClassifier` концептами являются `SPAM` и `NOT SPAM` в случае фильтрации электронной почты (двоичная классификация) и названия категорий электронной почты в случае категоризации сообщений (многоклассовая классификация). Образцы со-

```
*** Classifying instance: biz-01.html
P(NOT SPAM|biz-01.html) = 0.944444444444445
P(SPAM|biz-01.html) = 0.055555555555556

Classified biz-01.html as NOT SPAM

*** Classifying instance: sport-01.html
P(NOT SPAM|sport-01.html) = 0.894736842105263
P(SPAM|sport-01.html) = 0.105263157894737

Classified sport-01.html as NOT SPAM

*** Classifying instance: usa-01.html
P(NOT SPAM|usa-01.html) = 0.882352941176471
P(SPAM|usa-01.html) = 0.117647058823529

Classified usa-01.html as NOT SPAM

*** Classifying instance: world-01.html
P(NOT SPAM|world-01.html) = 0.962264150943396
P(SPAM|world-01.html) = 0.037735849056604

Classified world-01.html as NOT SPAM

*** Classifying instance: spam-biz-01.html
P(NOT SPAM|spam-biz-01.html) = 0.468750000000000
P(SPAM|spam-biz-01.html) = 0.531250000000000

Classified spam-biz-01.html as SPAM
```

Рис. 5.4. *Результаты фильтрации сообщений электронной почты со спамом (двоичная классификация) для классификатора на основе наивного байесовского алгоритма*

общений электронной почты инкапсулированы в классе `EmailInstance`, дополняющем класс `BaseInstance`. Этот пример демонстрирует специализацию классов предоставляемой нами общей базы с конкретной целью (классификация сообщений электронной почты).

Класс `EmailClassifier` получает свой набор `TrainingSet` посредством метода `getTrainingSet` экземпляра класса `EmailDataset`. После загрузки этих образцов классификатор *обучается* (*узнает, как*) отображать концепт на образец исходя из назначений, представленных в наборе `TrainingSet`. Класс `EmailClassifier` использует для обучения классификатора не всю информацию электронной почты, а единственный атрибут, значение которого вычисляется на стадии создания экземпляра класса `EmailInstance`, как показано в листинге 5.3.

Листинг 5.3. Создание экземпляра класса EmailInstance

```
public EmailInstance(String emailCategory, Email email, int topNTerms) {

  super();
  this.id = email.getId();
  this.setConcept(new BaseConcept(emailCategory));

  String text = email.getSubject()+" "+email.getTextBody();
  Content content = new Content(email.getId(), text, topNTerms);

  Map<String, Integer> tfMap = content.getTFMap();

  attributes = new StringAttribute[1];
  String attrName = "Email_Text_Attribute";
  String attrValue = "";

  for(Map.Entry<String, Integer> tfEntry : tfMap.entrySet()) {
      attrValue = attrValue + " " + tfEntry.getKey();
  }

  attributes[0] = new StringAttribute(attrName, attrValue);
}
```

Сначала мы выполняем конкатенацию текста из строки с темой и из тела сообщения электронной почты. Затем анализируем результат конкатенации текста и создаем список N первых часто встречающихся термов с помощью класса `Content` (с которым мы встречались в главе 3). Текстуальный анализ обеспечивает пользовательский анализатор, дополняющий класс `StandardAnalyzer` из библиотеки Lucene; класс `PorterStemFilter` используется для разметки (tokenizing) строк. Оба эти класса из библиотеки Lucene можно найти в пакете `org.apache.lucene.analysis`.

Как видим, единственным атрибутом образца (`Email_Text_Attribute`) служит значение конкатенации N первых часто встречающихся термов. Разумеется, это допущение для упрощения моделирования. Несмотря на простоту, такой подход во многих случаях может обеспечить хорошие результаты. Не забудьте, что проектируя (или выбирая) интеллектуальный алгоритм для реального приложения, вы всегда должны начинать с простейшей возможной модели, которая является работоспособной. Это эквивалентно максимальному уклонению от ранней оптимизации программного кода, если вы привыкли оперировать такими понятиями.

Даже несмотря на то, что простое решение может не оказаться тем, которое вы в конце концов используете, оно позволит разобраться в природе ваших данных и в связанных с вашей задачей трудностях, не усложняя ситуацию изначально. Есть масса других вариантов. Вы можете выбрать два атрибута: одно значение атрибута – для темы сообщения, а другое – для тела сообщения. Можно также добавить сюда атрибут `from`. Если ваша электронная почта имеет временную отметку, можно учитывать время отправления сообщения – в рабочие часы или поздним вечером.

В разделе «Сделать» мы приглашаем вас исследовать эти и другие альтернативы (не стесняйтесь проявлять творческий подход) и сравнить результаты, а также изменения в сложности классификации, по мере того как увеличивается объем информации о сообщениях электронной почты, учитываемой при обучении классификатора.

Более подробное знакомство с наивным байесовским классификатором

Пора ближе познакомиться с реализацией наивного байесовского алгоритма. В листинге 5.4 приведен класс `NaiveBayes`, без своего простого по логике конструктора, комментариев Javadoc, вывода некоторых протокольных данных и пары простейших методов для получения значений свойств (getters). В остальном он здесь целиком – всего лишь на двух страницах программного кода, – один из самых надежных, успешных и широко используемых алгоритмов классификации всех времен!

Напоминаем, что классификатор узнаёт о том, как надо назначать классы образцам, анализируя пример образцов, предоставленных для обучения, и в результате определяет класс, назначенный данному образцу. Естественно, интерфейс `Classifier` требует, чтобы каждый классификатор реализовал методы `boolean train()` и `Concept classify(Instance instance)`. Разумеется, каждый классификатор реализует эти методы по-своему, поэтому давайте посмотрим, как это делается в классе `NaiveBayes`.

Листинг 5.4. Класс NaiveBayes: универсальный байесовский классификатор

```
public class NaiveBayes implements Classifier {

  private String name;

  protected TrainingSet tSet;

  protected Map<Concept,Double> conceptPriors;   ❶

  protected Map<Concept, Map<Attribute, AttributeValue>> p;   ❷

  protected ArrayList<String> attributeList;   ❸

  public boolean train() {   ❹
    boolean hasTrained = false;

    if ( attributeList == null || attributeList.size() == 0) {
       System.out.print("Can't train the classifier
➥      without attributes for training!");
       System.out.print("Use the method --> trainOnAttribute(Attribute a)");
    } else {
      calculateConceptPriors();

      calculateConditionalProbabilities();
```

```
    hasTrained = true;
  }
  return hasTrained;
}

public void trainOnAttribute(String aName) {

  if (attributeList ==null) {
    attributeList = new ArrayList<String>();
  }
  attributeList.add(aName);
}

private void calculateConceptPriors() {   ❺

  for (Concept c : tSet.getConceptSet()) {

    int totalConceptCount=0;

    for (Instance i : tSet.getInstances().values()) {

      if (i.getConcept().equals(c)) {
        totalConceptCount++;
      }
    }
    conceptPriors.put(c, new Double(totalConceptCount));
  }
}

protected void calculateConditionalProbabilities() {   ❻
  p = new HashMap<Concept, Map<Attribute, AttributeValue>>();

  for (Instance i : tSet.getInstances().values()) {

    for (Attribute a: i.getAtrributes()) {

      if (a != null && attributeList.contains(a.getName())) {

        if ( p.get(i.getConcept())== null ) {
          p.put(i.getConcept(), new HashMap<Attribute, AttributeValue>());
        }
        Map<Attribute, AttributeValue> aMap = p.get(i.getConcept());
        AttributeValue aV = aMap.get(a);
        if ( aV == null ) {

          aV = new AttributeValue(a.getValue());
          aMap.put(a, aV);
        } else {
          aV.count();
        }
      }
    }
  }
}
```

```
 public double getProbability(Instance i, Concept c) {   ❼
   double cP=1;

   for (Attribute a : i.getAtrributes()) {

     if ( a != null && attributeList.contains(a.getName()) ) {

       Map<Attribute, AttributeValue> aMap = p.get(c);
       AttributeValue aV = aMap.get(a);
       if ( aV == null) {
         cP *= ((double) 1 / (tSet.getSize()+1));
       } else {
         cP *= (double)(aV.getCount()/conceptPriors.get(c));
       }
     }
   }
   return (cP == 1) ? (double)1/tSet.getNumberOfConcepts() : cP;
 }

public double getProbability(Concept c, Instance i) {   ❽
   double cP=0;

   if (tSet.getConceptSet().contains(c)) {

     cP = (getProbability(i,c)*getProbability(c))/getProbability(i);

   } else {
     cP = 1/(tSet.getNumberOfConcepts()+1.0);
   }
   return cP;
 }

 public double getProbability(Instance i) {
   double cP=0;

   for (Concept c : getTset().getConceptSet()) {

     cP += getProbability(i,c)*getProbability(c);
   }
   return (cP == 0) ? (double)1/tSet.getSize() : cP;
 }

 public double getProbability(Concept c) {   ❾
     Double trInstanceCount = conceptPriors.get(c);
     if( trInstanceCount == null ) {
         trInstanceCount = 0.0;
     }
     return trInstanceCount/tSet.getSize();
 }

 public Concept classify(Instance instance) {   ❿
   Concept bestConcept = null;
   double bestP = 0.0;
```

```
        for (Concept c : tSet.getConceptSet()) {
            double p = getProbability(c, instance);
            if( p >= bestP ) {
                bestConcept = c;
                bestP = p;
            }
        }
        return bestConcept;
    }
}
```

Длинный листинг. Но прежде чем рассмотреть его подробно, вспомним, что мы узнали в главе 2. Наивный байесовский алгоритм оценивает то, что называется *условной вероятностью* (conditional probability) X при условии Y. То есть позволяет узнать, с какой вероятностью можно наблюдать концепт X при условии, что мы уже видели образец Y. В частности, этот классификатор использует в качестве входных данных следующее:

- Вероятность наблюдения концепта X в общем случае, также называемую *априорной* (prior) вероятностью; обозначается $p(X)$
- Вероятность наблюдения образца Y при условии, что мы случайным образом выбираем образец из концепта X, также называемую *правдоподобием* (likelihood); обозначается $p(y \mid X)$
- Вероятность наблюдения образца y в общем случае, также называемую *подтверждением* (evidence); обозначается $p(Y)$

На выходе классификатора – вычисленная вероятность того, что образец Y принадлежит концепту X, также называемая *апостериорной вероятностью* (posterior probability); обозначается $p(X \mid Y)$. Это вычисление выполняется по следующей формуле (ее называют теоремой Байеса):

$$p(X \mid Y) = \frac{p(Y \mid X)p(X)}{p(Y)}$$

До сих пор мы старались избегать математических формул в явном виде. Но несмотря на простую внешность, это очень мощная формула, фундамент множества классификаторов, от применяющих наивный байесовский алгоритм до реализаций на основе гауссовых процессов и байесовских доверительных сетей [MacKay, D. J. C.]. Если вы не против запомнить одну формулу, хорошенько заучите эту!

До тех пор пока мы заняты собственно классификацией, оценка подтверждения $p(Y)$ не требуется, потому что значение этой вероятности не меняется для разных классов. Классификатор работает, вычисляя апостериорные вероятности $p(X \mid Y)$ для всех классов и выбирая класс с наибольшей апостериорной вероятностью. Выполним мы деление на $p(Y)$ или нет, порядок результата не изменится. Поскольку, с точки

зрения вычислений, дешевле это деление не выполнять, в реализации можно избегать деления на $p(Y)$.

Давайте проанализируем один за другим главные моменты листинга 5.4. Прежде всего, мы задаем имя для экземпляра классификатора `NaiveBayes`. Если вы используете единственный классификатор, в этом нет необходимости. Но, как вы увидите в главе 6, довольно часто приходится создавать ансамбли классификаторов, объединяя их, чтобы улучшить результаты. Сохранение идентификатора для этого классификатора окажется полезным позже. Разумеется, каждому классификатору нужен набор для обучения. Имя классификатора и его набор для обучения намеренно задаются на этапе создания экземпляра класса. После того как экземпляр классификатора `NaiveBayes` создан, вы не можете заменить его набор для обучения `TrainingSet` другим, однако всегда можно получить ссылку на этот набор и добавить в него новые образцы.

❶ Карта `conceptPriors` позволяет хранить счетчик для каждого концепта из набора для обучения. Мы могли бы использовать эту карту для хранения не только счетчиков, но и априорных вероятностей. Однако счетчики надо использовать повторно, поэтому ради эффективности вычислений мы и храним именно их; априорные вероятности можно получить простым делением.

❷ В переменной `p` хранятся условные вероятности – вероятность наблюдения концепта X при условии, что мы наблюдали образец Y, или, в случае переходов пользователя по ссылкам, вероятность того, что пользователь A хочет видеть URL-адрес X при условии, что он предложил запрос Q.

❸ Это список атрибутов, которые должен учитывать классификатор при обучении. Образцы из набора для обучения могут иметь несколько атрибутов, и только некоторые из них могут быть значимыми, поэтому мы следим за тем, какие атрибуты следует использовать. Для заполнения этого списка служит метод `trainOnAttribute(String)`.

❹ Метод `train()` отвечает за обучение классификатора. Быстро удостоверившись в том, что у нас есть хотя бы один атрибут для обучения, этот метод вычисляет априорные вероятности концептов и условные вероятности, следуя формуле теоремы Байеса. Если все идет хорошо, этот метод вернет значение `true`; в противном случае будет возвращено значение `false`.

❺ Это первая составляющая процесса обучения, где вычисляются априорные вероятности $p(X)$. Для всех образцов из набора для обучения мы подсчитываем, сколько раз встретился каждый концепт. В данной реализации обеспечивается хранение состояния счетчика. Реальные априорные вероятности концептов – это значения счетчиков, деленные на общее число образцов в наборе для обучения.

❻ Вторая составляющая процесса обучения, где подсчитывается, сколько раз данное конкретное значение атрибута встречается в концепте. Это число необходимо для вычисления условных вероятностей $p(Y \mid X)$, выполняемого в методе `getProbability(Instance I, Concept c)`. Для каждого образца из набора для обучения и для каждого атрибута из списка атрибутов для обучения мы подсчитываем, сколько раз было встречено конкретное значение в данном концепте.

❼ Вычисляем условные вероятности $p(Y \mid X)$. Своим происхождением термин *наивный* (naïve) обязан этому методу. Обратите внимание: мы ищем вероятности наблюдения конкретных образцов при условии конкретного концепта. Но каждый образец уникально определяется уникальными значениями его атрибутов. Условная вероятность образца является, по сути, объединенной вероятностью всех условных вероятностей значений атрибутов. Каждая условная вероятность значения атрибута определяется выражением `(aV.getCount()/conceptPriors.get(c))`. В предыдущей реализации предполагалось, что все эти значения атрибутов статистически независимы, поэтому объединенная вероятность – это просто произведение отдельных вероятностей, полученных для каждого значения атрибутов. Это и есть «наивная» часть. В общем случае, без предположения о статистической независимости атрибутов, объединенная вероятность не была бы равна этому произведению.

Мы взяли в кавычки слово «наивный», потому что, как оказалось, наивный байесовский алгоритм очень надежен и широко применяется даже для решения таких задач, в рамках которых предположение о независимости атрибутов явно нарушается. Фактически, можно показать, что наивный байесовский алгоритм является оптимальным в диаметрально противоположном случае – когда между атрибутами есть полностью детерминированная зависимость [Rish, I.].

В случае значений атрибутов, которые не были подсчитаны ранее, мы назначаем произвольную условную вероятность, которая равна величине, обратной числу образцов в наборе для обучения, плюс 1. Эта аппроксимация носит произвольный характер; вы можете прибавить 2 или 3, или вычислить вероятность пропущенных значений атрибутов каким-то совершенно иным способом. Не преуменьшайте влияние этой аппроксимации на результаты классификации, особенно если набор для обучения невелик. Каким, по-вашему, должно быть это значение для небольшого набора для обучения?

❽ Этот метод позволяет вычислить апостериорную вероятность класса. Апостериорная вероятность – это выходные данные формулы теоремы Байеса. По сути, классификация данного образца сводится к многократному обращению к данному методу: по одному обращению для каждого класса. Здесь возможна оптимизация, смысл которой – избежать обращения к методу `getProbability(i)` и последующего деления,

поскольку, как уже говорилось, оценка подтверждения (выражение $p(Y)$ в формуле теоремы Байеса) для классификации не требуется. Можно было бы игнорировать и сам метод `getProbability(Instance)`; мы включили его здесь для полноты представления.

В методе `getProbability(Concept, Instance)` выполняется проверка, позволяющая выяснить, встречался ли уже конкретный концепт. Если бы для классификации был использован класс `NaiveBayes` с фиксированным набором концептов, этот шаг был бы не нужен. Но вспомните, этот же класс мы использовали в главе 2, в контексте обучения на основе переходов пользователя по ссылкам, где существовала возможность передачи концепта, не включенного в набор для обучения.

❾ С помощью этого метода вычисляется априорная вероятность $p(X)$ класса X как отношение числа образцов, назначенных данному классу, к общему числу образцов в наборе для обучения.

❿ Метод `classify(Instance)` обеспечивает классификацию данного образца, возвращая класс, для которого получена наибольшая вероятность наблюдения. Можно было бы использовать массив, чтобы сохранить полученные для каждого класса значения вероятностей. Впоследствии этот массив можно было бы отсортировать и вернуть три (или пять) лучших классов в случае многоклассовой классификации. В реальной системе предпочтение следовало бы отдать именно такому решению, поскольку вероятности могут иметь очень близкие друг другу значения, и тогда приложение могло бы показать конечному пользователю диапазон вариантов для выбора, вместо того чтобы автоматически назначать какой-то один.

Универсальный классификатор электронной почты

Итак, давайте более внимательно рассмотрим класс `EmailClassifier`. В листинге 5.5 приведен программный код этого класса, за исключением определения объектных переменных, конструктора и методов классификации, которые очень просты. Поскольку мы только что подробно объяснили классификатор `NaiveBayes`, сосредоточимся на переопределенных методах этого класса.

Листинг 5.5. Классификатор электронной почты на основе универсального класса NaiveBayes

```
public class EmailClassifier extends NaiveBayes {

    public boolean train() {   ❶
        if( emailDataset.getSize() == 0) {
            System.out.println("Can't train classifier -
➥   training dataset is empty.");
            return false;
        }
        for(String attrName : getTset().getAttributeNameSet()) {
```

```
            trainOnAttribute(attrName);
        }
        super.train();
        return true;
    }

    protected void calculateConditionalProbabilities() {

        p = new HashMap<Concept, Map<Attribute, AttributeValue>>();

        for (Instance i : tSet.getInstances().values()) {

            Attribute a = i.getAtrributes()[0];   ❷

            Map<Attribute, AttributeValue> aMap = p.get(i.getConcept());
            if ( aMap == null ) {
                aMap = new HashMap<Attribute, AttributeValue>();
                p.put(i.getConcept(), aMap);
            }

            AttributeValue bestAttributeValue =
➥           findBestAttributeValue(aMap, a)   ❸

            if (bestAttributeValue != null ) {
                bestAttributeValue.count();
            } else {
                AttributeValue aV = new AttributeValue(a.getValue());
                aMap.put(a, aV);
            }
        }
    }

public double getProbability(Instance i, Concept c) {
      double cP=1;
      for (Attribute a : i.getAtrributes()) {

        if ( a != null && attributeList.contains(a.getName()) ) {

          Map<Attribute, AttributeValue> aMap = p.get(c);
          Attribute bestAttributeValue = findBestAttributeValue (aMap, a);
          if (bestAttributeValue == null) {
             cP *= ((double) 1 / (tSet.getSize()+1));   ❹
          } else {
             cP *= (double)(bestAttributeValue.getCount()/conceptPriors.
get(c));
          }
        }
     }
     return (cP == 1) ? (double)1/tSet.getNumberOfConcepts() : cP;
 }

private Attribute findBestAttributeValue(Map<Attribute,
➥  AttributeValue> aMap, Attribute a) {
```

```
    JaccardCoefficient jaccardCoeff = new JaccardCoefficient();   ❺

    String aValue = (String)a.getValue();
    String[] aTerms = aValue.split(" ");
    Attribute bestMatch = null;
    double bestSim = 0.0;
    for(Attribute attr : aMap.keySet()) {
      String attrValue = (String)attr.getValue();
      String[] attrTerms = attrValue.split(" ");
      double sim = jaccardCoeff.similarity(aTerms, attrTerms);
      if( sim > jaccardThreshold && sim > bestSim) {   ❻
          bestSim = sim;
          bestMatch = attr;
      }
    }
    return bestMatch;
  }
}
```

❶ Этот метод гарантирует загрузку каких-то данных для обучения, назначает соответствующие атрибуты для обучения из набора данных для обучения и инициирует метод `train()` класса `NaiveBayes` для собственно обучения классификатора.

❷ Это предложение оправданно только в данной конкретной реализации. В общем случае у вас был бы не один, а несколько атрибутов. В одном из пунктов раздела «Сделать» вам предлагается самостоятельно проанализировать случай классификации электронной почты, для чего вы должны ввести большее число атрибутов. Если вы работаете над этим заданием, придется пересмотреть указанную часть реализации.

❸ Этот шаг необходим, потому что мы используем чисто текстовое представление для сообщений электронной почты. Данная методика является общей, ее можно применять всегда, когда вы имеете дело с текстом. В нашей реализации единственный используемый атрибут получает свое значение в результате конкатенации строк темы сообщения и тела сообщения; напоминаем, что для этого мы использовали пользовательский анализатор, чтобы уменьшить шум и извлечь максимум возможной информации. Если мы учитываем строгое равенство строк при сравнении значений атрибутов, каждое сообщение электронной почты из предоставленного нами в качестве примера набора данных будет иметь собственное значение атрибута. Вместо этого мы считаем, что два значения атрибута эквивалентны, если они совпадают согласно алгоритму, реализованному в методе `findBestAttributeValue`.

❹ В случае если атрибуты имеют ранее не встречавшиеся значения, наша оценка совпадает с оценкой, реализованной в классе `NaiveBayes`. Мы назначаем произвольную условную вероятность, равную вели-

чине, обратной числу образцов в наборе для обучения, плюс 1. Не преуменьшайте влияние этой аппроксимации на результаты классификации, особенно если набор для обучения невелик. Не забывайте, что речь идет об условной вероятности наблюдения конкретного значения атрибута. В определенных случаях эти значения предоставляются людьми – экспертами в предметной области, использующими свой опыт для создания оценки, которая может быть больше (или меньше), чем оценка, полученная исходя из размера набора для обучения. Эту оценку можно заменить небольшой константой (как правило, вполне подходит малая величина порядка 10^{-4} или 10^{-5}) и посмотреть, как эта замена скажется на результатах классификации.

❺ Возможно, вы еще помните коэффициент Жаккарда из главы 3 (раздел «Сделать») или из главы 4 (он был использован в реализации алгоритма ROCK). Это мера сходства, основанная на отношении размеров пересечения и объединения двух множеств. В данном случае этими двумя множествами являются лексемы, полученные в результате разбиения значений атрибута на отдельные термы. Можно было бы использовать одну из множества других мер сходства, с которыми мы уже встречались. Полезно использовать класс `CosineSimilarity` вместо класса `JaccardCoefficient` и сравнить результаты классификации.

❻ `jaccardThreshold` – это объектная переменная, с которой связаны методы `get` и `set`. По умолчанию, этой переменной присвоено значение 0,25, но вы можете в оперативном режиме заменить его любым другим, по своему выбору. Присвоить этой переменной значение 0,3 в среде BeanShell можно с помощью команды `emailFilter.setJaccardCoefficient(0.3);`. Значение данной переменной – это минимальное значение показателя сходства, которым, согласно нашему желанию, должны обладать два значения атрибута, чтобы мы признали их эквивалентными.

Вот и все! Мы рассмотрели реализацию вероятностного классификатора электронной почты. Способности нашего классификатора были продемонстрированы только для случая фильтрации сообщений. Поэтому давайте рассмотрим его применение для обобщенного (многоклассового) случая. В листинге 5.6 показаны необходимые шаги. Единственное отличие заключается в обращении к методу `setBinary(false)` класса `EmailDataset`. Поэтому на этапе создания набора данных назначаются разные классы (или категории электронной почты, если угодно). В обоих этих случаях, двоичной и многоклассовой классификации, классификатор работает одинаково.

Листинг 5.6. Загрузка набора сообщений для обучения и классификация электронной почты

```
EmailDataset trainEmailDS = EmailData.createTrainingDataset();

trainEmailDS.setBinary(false);   ← Использование всех категорий электронной почты
```

```
EmailClassifier emailFilter = new EmailClassifier(trainEmailDS, 10);

emailFilter.train();

emailFilter.sample();
```

Тестирование путем классификации нескольких сообщений электронной почты

Результаты показаны на рис. 5.5, где можно видеть, что только одно из сообщений электронной почты (usa-01) было классифицировано не-

```
*** Classifying instance: biz-01.html
P(WORLD|biz-01.html) = 0.085106382978723
P(BIZ|biz-01.html) = 0.765957446808511
P(USA|biz-01.html) = 0.063829787234043
P(SPAM|biz-01.html) = 0.042553191489362
P(SPORT|biz-01.html) = 0.042553191489362
Classified biz-01.html as BIZ

*** Classifying instance: sport-01.html
P(WORLD|sport-01.html) = 0.121212121212121
P(BIZ|sport-01.html) = 0.181818181818182
P(USA|sport-01.html) = 0.090909090909091
P(SPAM|sport-01.html) = 0.060606060606061
P(SPORT|sport-01.html) = 0.545454545454546
Classified sport-01.html as SPORT

*** Classifying instance: usa-01.html
P(WORLD|usa-01.html) = 0.235294117647059
P(BIZ|usa-01.html) = 0.352941176470588
P(USA|usa-01.html) = 0.176470588235294
P(SPAM|usa-01.html) = 0.117647058823529
P(SPORT|usa-01.html) = 0.117647058823529
Classified usa-01.html as BIZ

*** Classifying instance: world-01.html
P(WORLD|world-01.html) = 0.805970149253731
P(BIZ|world-01.html) = 0.089552238805970
P(USA|world-01.html) = 0.044776119402985
P(SPAM|world-01.html) = 0.029850746268657
P(SPORT|world-01.html) = 0.029850746268657
Classified world-01.html as WORLD

*** Classifying instance: spam-biz-01.html
P(WORLD|spam-biz-01.html) = 0.121212121212121
P(BIZ|spam-biz-01.html) = 0.181818181818182
P(USA|spam-biz-01.html) = 0.090909090909091
```

Рис. 5.5. *Использование классификатора для многоклассовой классификации сообщений электронной почты*

верно. Вероятности можно интерпретировать как меру доверия к тому факту, что это сообщение электронной почты принадлежит конкретному классу. В реальном приложении многоклассовой классификации, если уровень доверия ниже определенного значения (скажем, 0,7), система должна выбрать первые три или пять классов и предоставить их конечному пользователю в качестве кандидатур. Такие разновидности проектов с включением человека в рабочий цикл часто встречаются в интеллектуальных приложениях и действительно необходимы, чтобы непрерывно повышать производительность классификатора.

Как мы сказали, реальный фильтр электронной почты включает далеко не один только вероятностный классификатор. В следующем разделе будет представлена реализация процессора правил – методики, удачно дополняющей вероятностный классификатор. Многие возможности хороших фильтров спама опираются на правила, например белый и черный списки, совместно используемые базы данных для идентификации спама и так далее.

Таким образом, с помощью классификатора `NaiveBayes` можно отделять сообщения электронной почты, содержащие спам, от допустимых сообщений, а также распределять сообщения электронной почты по нескольким выбранным вами категориям. Разумеется, все, что мы говорили по поводу сообщений электронной почты, можно применить к любому другому документу, текстуальное представление которого вы в состоянии получить, – это могут быть документы текстового процессора Microsoft Word, XML-документы, HTML-документы с веб-сайтов, документы в формате PDF и так далее.

5.3.2. Классификация по правилам

В этом разделе мы проанализируем другой подход к классификации – *классификацию на основе правил*. Итак, что такое правила? И чем они отличаются от байесовского классификатора? Чтобы ответить на эти два вопроса, потребуется более общее представление о парадигмах программирования. Сегодня используются несколько парадигм программирования. Типичное приложение Java/J2EE характеризуется элементами процедурного, объектно-ориентированного и, возможно, аспектно-ориентированного программирования. В частности, процедурное программирование означает, что мы говорим компьютеру, *что* надо делать и *как*. Эта парадигма имеет самое широкое распространение, и именно ее мы используем для написания нашего ПО. Но есть и другая парадигма, известная как *декларативное* программирование, которая смещает акцент на то, что надо делать, а часть «как делать» делегирует механизму выполнения.

Пример декларативного программирования – рассуждения, основанные на прецедентах (rule-based reasoning). Составляющими системы классификации на основе правил являются *факты*, *процессор правил* и (разу-

мается) *правила*. Факты – это просто данные о мире. Правила являются условными предложениями, которые говорят нам, что надо делать, если данные удовлетворяют определенным условиям; другими словами, они эквивалентны условным операторам `if-then` языков программирования. Процессор правил отвечает за выполнение правил, ассоциированных с фактами. Процессор правил, в сравнении с вероятностным классификатором, существенно отличается тем, что накапливает и отображает знания. В случае вероятностного классификатора, такого как наивный байесовский классификатор, который мы изучили в предыдущем разделе, для представления знаний используются понятия априорных вероятностей концептов и условных вероятностей наблюдения, получаемых на основе данных из набора для обучения. «Ручное» вмешательство с целью накопления такого знания (вероятностей) отсутствует; если есть произвольный, правильно сформированный набор для обучения, классификатор извлечет из него информационный контент (знания), необходимый ему для выполнения поставленных перед ним задач классификации. Классификатор на основе правил накапливает знания в виде правил, это правила являются знанием системы, что порождает вопрос о том, как мы их получаем. Правила вводятся в систему вручную или полуавтоматически, когда эксперты-люди вводят правила с помощью удобных рабочих экранных форм.

У систем на основе правил есть два основных режима работы. Первый – это *прямая цепочка рассуждений* (forward chaining); этот режим управляется данными, в том смысле, что мы предоставляем данные и хотим выяснить, какие правила к ним применимы. Второй режим – это *обратная цепочка рассуждений* (backward chaining); этот режим управляется целями, то есть мы начинаем с цели, которую пытается (если это возможно) удовлетворить механизм правил. В этой книге обратная цепочка рассуждений не рассматривается; мы только скажем, что ее поддерживают такие языки программирования, как Пролог и ECLiPSe [см. Russell, S., and P. Norvig].

Процессор правил Drools

Есть две готовые к использованию реализации процессора правил на языке программирования Java. Первая называется Jess и была создана в лаборатории Sandia National Laboratories. Ко времени написания этой книги (весна 2008 г.) вышла версия 7.1 этого продукта, так что данная реализация довольно стабильна. Программное обеспечение Jess является бесплатным для академического использования, а для применения в коммерческих целях предлагается на платной основе [Friedman-Hill, E.]. Вторая реализация процессора правил называется Drools (*http://www.jboss.org/drools/*), вы также могли слышать название JBoss Rules. JBoss – это хорошо известный проект связующего программного обеспечения (middleware) с открытым исходным кодом, который в настоящее время находится под опекой компании Red Hat.

Система Drools представляет собой надежный процессор правил, снабженный обширной документацией и имеющий довольно либеральную лицензию открытого исходного кода (Apache 2.0), то есть вы можете бесплатно использовать это ПО в своем приложении. На протяжении последних четырех лет мы с большим успехом использовали процессор правил Drools. По нашему мнению, в мире Java предпочтение следует отдавать именно ему.

Процессор правил Drools состоит из двух главных модулей: модуля сопоставления с образцом и модуля плана решения. Модуль сопоставления с образцом определяет, какие из правил соответствуют фактам. После того как правила определены, они помещаются в модуль плана решения. На рис. 5.6 показаны базовые элементы процессора правил Drools. Чтобы выполнить сопоставление с образцом, процессор правил Drools реализует и дополняет алгоритм Rete; латинское слово «rete» означает «сеть» и произносится «ретэ» в Европе и «рити» в США. Алгоритм Rete был разработан Чарльзом Форджи (Charles Forgy) в 1974 году; он был и остается, во множестве своих воплощений, одним из самых эффективных и успешных алгоритмов сопоставления с образцом. Реализация этого алгоритма в процессоре правил Drools получила название *ReteOO*, отражая тот факт, что система Drools обладает усовершенствованной реализацией алгоритма Rete, наиболее подходящей для объектно-ориентированного программного обеспечения. Алгоритм Rete жертвует эффективностью потребления оперативной памяти ради быстрой обработки; теоретически, его производительность зависит от числа правил в системе. Но на деле по мере возрастания числа правил мы неизбежно сталкиваемся с хорошо известной проблемой систем ИИ – так называемой *проблемой эффективности* (*utility problem*). Об этом мы подробнее поговорим в разделе 5.6. Мы не будем вдаваться в детали самого алгоритма Rete; если вас интересует его реализация, то исходного

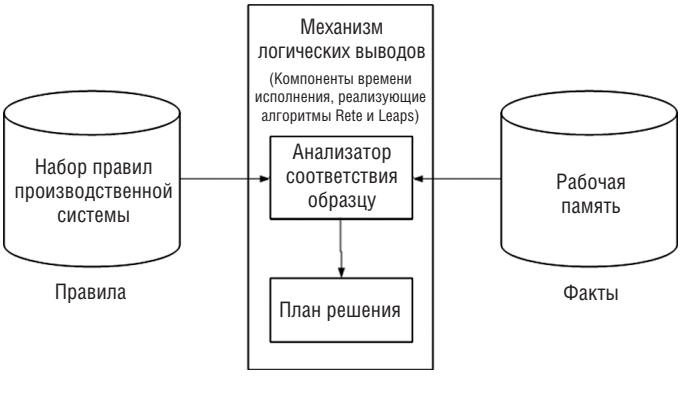

Рис. 5.6. *Базовые элементы системы процессора правил Drools (источник: Drools online)*

программного кода и документации процессора правил Drools, наряду со ссылками, приведенными в конце этой главы, должно хватить для того, чтобы полностью разобраться во внутренних механизмах его работы. Кроме того, работа алгоритма Rete подробнее объясняется в книге [Friedman-Hill, E.] (см., в частности, главу 8).

После этого краткого введения в процессоры правил мы готовы рассказать о том, как использовать библиотеку Drools в рассматриваемой нами задаче фильтрации сообщений электронной почты, содержащих спам. В частности, давайте посмотрим, как мы можем написать правила. Процессор правил Drools предоставляет язык сценариев (не XML), который легко освоить; его так просто выучить, что вы могли бы прямо предоставить его конечным пользователям вашего приложения! Давайте посмотрим.

Листинг 5.7. Простой набор правил для фильтрации сообщений электронной почты, содержащих спам

```
package demo;
import iweb2.ch5.classification.data.Email;
import iweb2.ch5.classification.rules.ClassificationResult;

global ClassificationResult classificationResult;

rule "Tests for viagra in subject"
when
    Email( $s : subject )
    eval( classificationResult.isSimilar($s, "viagra" ) )
then
    classificationResult.setSpamEmail(true);
end

rule "Tests for 'drugs' in subject"
when
subject
  Email( $s : subject )
    eval( classificationResult.isSimilar($s, "drugs" ) )
then
    classificationResult.setSpamEmail(true);
end
```

Правило для выявления слова "Viagra" в теме сообщения

Правило для выявления слова "drugs" в теме сообщения

В листинге 5.7 показано содержимое файла *spamRules.drl*, включающего два простых правила. Это содержимое почти не требует объяснений; указанный файл можно найти в каталоге *C:\iWeb2\data\ch05*. Как мы скоро увидим, правила передаются процессору Drools в виде пакетов, поэтому мы прежде всего определяем имя пакета, содержащего рассматриваемые правила, – `demo`. Предложения `import` предоставляют процессору выполнения правил информацию об определениях классов объектов, которые мы собираемся использовать в правилах, в данном случае это классы `Email` и `ClassificationResult`. Предложение `global` обеспечивает доступ к объекту с идентификатором `classificationResult`. Это пред-

ложение эквивалентно объявлению classificationResult как глобальной переменной в правилах. Итак, что означают заданные правила?

Первое правило называется "Tests for viagra in subject" (проверка наличия слова «Viagra» в теме сообщения). Как обещано в его названии, это правило проверяет, содержит ли переменная subject объекта Email слово *viagra*. Если указанное условие выполняется, переменной isSpamEmail объекта ClassificationResult присваивается значение true. Аналогично, второе правило, которое называется "Tests for 'drugs' in subject" (проверка наличия слова «drugs» в теме сообщения), проверяет, содержит ли переменная subject объекта Email слово *drugs*. Если это условие выполняется, переменной isSpamEmail объекта ClassificationResult присваивается значение true.

Мы не намекаем на то, что эти условия безоговорочно характеризуют электронную почту как спам; мы используем их только для того, чтобы проиллюстрировать структуру файла системы Drools. Как видите, общая структура определения правила в системе Drools имеет простую логику:

```
rule "Здесь указываем имя правила"
when
  <Здесь указываем условия>
then
  <Здесь указываем действия, которые должны быть выполнены,
➥ если указанные выше условия соблюдены>
end
```

В этом определении можно указать несколько условий и несколько действий. Вряд ли можно было сделать проще! Но при всей простоте и, как следствие, красоте языка правил системы Drools, думаем, истинная мощь этого процессора заключается в поддержке объектов. Обратите внимание: собственно оценка выполнения условий, указанных в обоих правилах, осуществляется в методе isSimilar класса ClassificationResult. Мы можем организовать довольно замысловатые оценки в объектно-ориентированном стиле.

Теперь посмотрим на эти правила в действии. Первая строка листинга 5.8 обеспечивает загрузку сообщений электронной почты из набора данных для тестирования; этот набор уже использовался в листинге 5.2. Поэтому разберем все остальные шаги, представленные в листинге 5.8.

Листинг 5.8. Применение правил фильтрации спама в электронной почте к набору данных

```
EmailDataset ds = EmailData.createTestDataset();

EmailRuleClassifier classifier =
  new EmailRuleClassifier("c:/iWeb2/data/ch05/spamRules.drl");   ❶

classifier.train();   ❷

classifier.run(ds,"Expecting one spam email.  :-(");   ❸
```

❶ Создаем классификатор электронной почты, который использует правила и надлежащим образом инкапсулирован в классе `EmailRuleClassifier`. Обратите внимание: единственным аргументом для конструктора этого класса является Drools-файл, который мы описали в листинге 5.7.

❷ Предлагаем классификатору «обучиться». В отличие от сценария из листинга 5.1, где мы обучали вероятностный классификатор, здесь мы не создавали набор данных для обучения. Конструктор классификатора не получает никаких ссылок на набор данных; ему передается только имя Drools-файла. Почему? Применяя вероятностный подход, мы пытаемся исходить из знаний, которые содержатся в наборе данных для обучения. В случае систем на основе правил знаниями являются правила. Для системы, действующей на основе правил, «обучающая» часть включает только загрузку правил из файла.

❸ «Применяем» правила к тестовому набору данных, передавая в качестве параметров информацию из набора данных для тестирования и описательное сообщение. Хотя бизнесмены и пользователи систем, основанных на правилах, предпочитают говорить о «применении правил», в действительности выполнение алгоритма Rete больше походит на фильтрацию данных (фактов) через «воронку». Эта «воронка» образована сетью (отсюда термин *rete*) узлов. По мере того как каждый факт «просачивается» сквозь такую «воронку», он проходит ряд проверок (условий правил), и когда достигает «дна», мы точно знаем, выполнение какого правила должно быть инициировано этим фактом. Подробное описание структуры алгоритма Rete вы найдете в [Doorenbos, R. B.].

Результаты выполнения кода из листинга 5.8 приведены на рис. 5.7. Сообщение электронной почты со спамом, соответствующее файлу *spam-biz-01.html*, привело к срабатыванию классификатора, как того требует правило спама из листинга 5.7, поскольку в теме этого сообщения содержится слово «drugs». Правило сработало, потому что были выполнены его условия.

Более пристальный взгляд на реализацию

Теперь, когда вы знаете, как использовать процессор правил Drools, рассмотрим подробнее классы-оболочки, которые мы использовали в листингах 5.7 (файл определения правил) и 5.8. Если вдуматься, нам удалось упаковать огромный объем функциональных возможностей всего лишь в несколько строк программного кода. Рассмотрим программный код, позволяющий применить процессор правил и получить результаты, ограничившись тремя простыми шагами. Начнем с базового элемента этой реализации – класса `RuleEngine`, который приведен в листинге 5.9.

```
bsh % classifier.run(ds,"Expecting one spam email.  :-(");

Expecting one spam email.  :-(
--------------------------------------------------

Classifying email: world-01.html ...
Rules classified email: world-01.html as: NOT-SPAM

Classifying email: spam-biz-01.html ...
Invoked ClassificationResult.setSpamEmail(true)
Rules classified email: spam-biz-01.html as: SPAM

Classifying email: sport-01.html ...
Rules classified email: sport-01.html as: NOT-SPAM

Classifying email: usa-01.html ...
Rules classified email: usa-01.html as: NOT-SPAM

Classifying email: biz-01.html ...
Rules classified email: biz-01.html as: NOT-SPAM
```

Рис. 5.7. *Выявление в электронной почте сообщения со спамом spam-biz-01 с помощью правил из листинга 5.7*

Листинг 5.9. Класс RuleEngine: создание процессора правил на основе библиотеки Drools

```
public class RuleEngine {

  private RuleBase rules;

  public RuleEngine(String rulesFile) throws RuleEngineException {
    try {
       Reader source = new InputStreamReader(new BufferedInputStream(
➥      new FileInputStream(rulesFile)));

       Properties properties = new Properties();
       properties.setProperty("drools.dialect.java.compiler", "JANINO" );
       PackageBuilderConfiguration cfg =
➥      new PackageBuilderConfiguration( properties );

       PackageBuilder builder = new PackageBuilder(cfg);

       builder.addPackageFromDrl(source);

       Package pkg = builder.getPackage();

       rules = RuleBaseFactory.newRuleBase();

       rules.addPackage(pkg);
```

❶ Определение компилятора периода выполнения

❷ Содержит наши правила

❸ Контейнер времени исполнения для правил

```
    } catch (Exception e) {
       throw new RuleEngineException(e);
    }
  }

  public void executeRules(ClassificationResult classificationResult,
➥ Email email ) {

    WorkingMemory workingMemory =
➥ rules.newStatefulSession();      ←❹ Рабочая память с сохранением состояния

    workingMemory.setGlobal("classificationResult",
➥ classificationResult);

    workingMemory.insert(email);    ←❺ Добавление факта в рабочую память

    workingMemory.fireAllRules();   ←❻ Выполнение всех правил
  }
}
```

Создание процессора правил Drools складывается из двух этапов: *подготовки* (authoring) и *исполнения* (runtime). Подготовительный этап начинается с разбора файла системы Drools – файла с расширением *.drl*. Парсер проверяет грамматическую непротиворечивость файла системы Drools и создает промежуточное *абстрактное синтаксическое дерево* (abstract syntax tree, AST). Для этого процессор правил Drools использует лексический парсер, предоставляемый проектом с открытым исходным кодом ANTLR (Another Tool for Language Recognition[1], *http://www.antlr.org/*). Проверенные правила загружаются в сериализованные объекты класса `Package`; экземпляр класса `Package` – это отдельный программный модуль, который содержит одно или несколько правил. Для той части процессора правил Drools, которая относится к этапу исполнения, базовым является класс `RuleBase`. Экземпляры класса `Package` можно в любое время добавить или удалить из экземпляра класса `RuleBase`.

Проанализируем все шаги, необходимые для того, чтобы создать и использовать процессор правил Drools, как показано в листинге 5.9:

❶ Создав ссылку на файл с определениями правил, мы создаем экземпляр класса `Properties` и присваиваем значение свойству `drools.dialect.java.compiler`. Что это за свойство? И что означает значение `JANINO`? Вы можете встроить Java-код прямо в файлы правил системы Drools. Это свойство позволяет указать компилятор времени исполнения, которым, согласно нашему желанию, должен пользоваться процессор правил Drools, чтобы осуществить компиляцию Java-кода. Janino – это имя встроенного Java-компилятора, включенного в дистрибутив Drools с лицензией BSD (*http://www.janino.net/*).

[1] Еще один инструмент для распознавания языков. – *Прим. перев.*

Чтобы завершить подготовительный этап, необходимо создать экземпляр класса `PackageBuilder`, который в свою очередь создаст экземпляры класса `Package`. Для конфигурации построителя пакетов мы используем вспомогательный класс `PackageBuilderConfiguration`. В этом классе определены значения по умолчанию, которые вы можете изменить с помощью соответствующих методов `set` или, как сделано здесь, путем настройки свойств при первом использовании. В данном случае мы задаем значение только для одного свойства, но можно было бы предоставить намного больше информации. Центром настройки является класс `ChainedProperties`, который выполняет поиск файлов `drools.packagebuilder.conf` в нескольких местах. Такими местами, в порядке очередности поиска, являются: системные свойства, файл, определенный пользователем в свойствах системы, «домашний» каталог пользователя, рабочий каталог и различные места хранения данных META-INF. Класс `PackageBuilderConfiguration` обеспечивает регистрацию в реестре классов `AccumulateFunctions`, `Dialects` и главного класса `ClassLoader`. Дополнительную информацию вы найдете в онлайн-документации для процессора Drools по адресу *http://downloads.jboss.com/drools/docs/4.0.7.19894.GA/html_single/index.html-d0e766*.

❷ Теперь, когда у нас есть класс `PackageBuilder`, мы можем создать пакеты, которые содержат правила. В метод `addPackageFromDrl` передается ссылка на файл, и сразу же вызывается метод `getPackage` построителя. Теперь правила готовы к применению!

❸ Наш первый шаг на пути построения той части процессора, которая относится к исполнению. Экземпляр класса `RuleBase` может иметь один или несколько пакетов `Package`. В любое время он может создать один или несколько экземпляров класса `WorkingMemory`; если в конфигурации не определено иное, поддерживается слабая ссылка. Класс `WorkingMemory` состоит из нескольких субкомпонентов; подробности – в онлайн-документации для процессора Drools.

❹ Класс `StatefulSession` является расширением класса `WorkingMemory`. Он обеспечивает дополнительные асинхронные методы для вставки, обновления и запуска правил, а также метод `dispose()`. Экземпляр класса `RuleBase` сохраняет ссылку на каждый созданный им экземпляр класса `StatefulSession`, чтобы обновлять их при добавлении новых правил. Метод `dispose()` необходим для удаления ссылки на объект `StatefulSession` из экземпляра класса `RuleBase`, чтобы не допускать утечек памяти.

В файле системы Drools, он показан в листинге 5.7, мы использовали предложение `global` для получения доступа к объекту с идентификатором `classificationResult`. Это эквивалентно объявлению `classificationResult` как глобальной переменной в наших правилах. Но это предложение не будет работать, если только мы не обратимся также

к методу `setGlobal` экземпляра класса `WorkingMemory`. Аргумент этого метода должен в точности совпадать с записью в файле правил системы Drools.

❺ Используем метод `insert`, чтобы добавить факты в экземпляр класса `WorkingMemory`. Если добавляется новый факт, процессор правил Drools будет сопоставлять его со всеми правилами. Это значит, что вся работа выполняется в момент добавления, но ни одно правило не будет выполнено до тех пор, пока не будет вызван метод `fireAllRules()`, что мы и делаем на следующем шаге.

❻ Инициируем выполнение правил. Вы не должны обращаться к методу `fireAllRules()`, пока не завершите добавление всех фактов. Ключевой этап сопоставления фактов и правил совпадает по времени с добавлением новых фактов, как уже говорилось. Таким образом, не стоит выполнять правила, прежде не сопоставив с ними все факты.

Практически, это все, что надо сделать, чтобы создать процессор правил с помощью библиотеки Drools. Теперь посмотрим, как класс `EmailRuleClassifier` делегирует свои действия классу `RuleEngine`, чтобы классифицировать сообщения электронной почты. Наша реализация `RuleEngine` ориентирована на обработку электронной почты; в одном из пунктов раздела «Сделать» этой главы мы приглашаем вас создать обобщение, которое использует интерфейс `Instance`. В листинге 5.10 приведен весь программный код класса `EmailRuleClassifier`, кроме метода `main`, в основном совпадающего с представленным в листинге 5.8.

Листинг 5.10. Процессор правил на базе библиотеки Drools, обнаруживающий спам в электронной почте

```
public class EmailRuleClassifier {

    private String ruleFilename;
    private RuleEngine re;
    private Concept spam;
    private Concept notSpam;

    public EmailRuleClassifier(String ruleFilename) {
        this.ruleFilename = ruleFilename;
    }

    public void train() {
        re = new RuleEngine(ruleFilename);   ❶

        spam = new BaseConcept("SPAM");      ❷
        notSpam = new BaseConcept("NOT-SPAM");
    }

    public Concept classify(Email email) {
        ClassificationResult result = new ClassificationResult();   ❸

        re.executeRules(result, email);   ❹
```

```
        if( result.isSpamEmail() ) {   ❺
            return spam;
        } else {
            return notSpam;
        }
    }

    public void run(EmailDataset ds, String msg) {
        System.out.println("\n");
        System.out.println(msg);
      System.out.
println("_____________________________________________________");
        for(Email email : ds.getEmails() ) {   ❻

            Concept c = classify(email);

            System.out.println("Email: "+
➥           email.getId()+" classified as: "+c.getName());
        }

      System.out.
println("_____________________________________________________");
    }
}
```

❶ Прежде всего, необходимо создать экземпляр класса `RuleEngine`, чтобы можно было делегировать ему применение правил. Мы передаем в качестве параметра имя файла, содержащего правила, и позволяем объекту `RuleEngine` выполнить всю тяжелую работу.

❷ Эти две вспомогательные переменные используются в методе `classify`. Поскольку в нашем случае они являются константами, независимо от того, что представляют собой правила или сообщения электронной почты, мы обращаемся с ними как с объектными переменными. Возможно, ваша реализация класса `ClassificationResult` отвечает за определение правильного концепта, принадлежащего более сложной структуре данных концептов (например, онтологии).

❸ Этот класс инкапсулирует две важные вещи. Он включает проверку условий правил (посредством метода `isSimilar`), а также действия правил (метод `setSpamEmail`). Можно создать другие объекты для инкапсуляции условий и действий. Если ваши условия или действия включают алгоритмически трудные реализации, лучше релизовать их по отдельности, обеспечив явное разделение этих двух составляющих.

❹ Здесь применение правил делегируется объекту `RuleEngine`. Этот метод рассматривался в листинге 5.9.

❺ Мы снова используем экземпляр класса `ClassificationResult`, чтобы получить информацию, которая была создана в результате действий правил (сработавших). Эту информацию можно было бы записать

в постоянную среду хранения (например, включить в запись базы данных или сохранить в файле); в нашем простом случае все необходимые действия обеспечивает класс `ClassificationResult`.

❻ Этот метод помогает классифицировать сразу все сообщения электронной почты из набора данных. Заметим, что мы могли бы передать в метод `classify` набор данных и переопределить метод `executeRule` в классе `RuleEngine`, так чтобы в рабочую память загружались сразу все сообщения электронной почты. Но обратите внимание: в контексте системы на основе правил классификация сообщения электронной почты как спама не зависит от того, являются ли спамом другие сообщения.

Разрешение конфликтов

Последнее замечание связано (но по-особому) с еще одной интересной темой. Что происходит, если действие правила *A* модифицирует факт *X*, который активизирует правило *B*, которое затем модифицирует факт *Y* и снова запускает правило *A*? Здесь возможны бесконечные циклы, если не найти способ, позволяющий прервать рекурсию. Что если по одному правилу сообщение электронной почты будет классифицировано как спам, а по другому – как «не спам»? Другими словами, что происходит, когда возникает конфликт между двумя и более правилами? По завершении работы метода `executeRule` мы должны получить ответ – так каким он будет?

К счастью для нас, процессор Drools обеспечивает решение этой, а также многих других проблем с помощью атрибутов правил. Решить первую проблему позволяет атрибут правила `no-loop` (без цикла); вторая проблема решается с помощью атрибута правила `salience` (значение). Атрибуты правил позволяют повлиять на поведение правил декларативно – в файле правил системы Drools. Некоторые атрибуты довольно просты, например `salience`, тогда как другие весьма замысловаты, например `ruleflow-group` (группа связанных последовательно выполняемых правил). Чтобы получить полное представление о каждом атрибуте, обратитесь к официальной документации процессора Drools, а также к исходному программному коду.

Теперь посмотрим, как можно задействовать атрибут `salience`, чтобы обеспечить *разрешение конфликтов* (conflict resolution). В листинге 5.11 показан файл правил системы Drools, содержащий три правила. На этот раз при обработке определенных сообщений электронной почты должен возникнуть конфликт правил, потому что одновременно будут удовлетворяться условия нескольких правил. Внешне этот файл почти не отличается от первого файла правил системы Drools, приведенного в листинге 5.7, но теперь для каждого правила мы ввели атрибут `salience` с целым значением.

Листинг 5.11. Простой набор правил для фильтрации сообщений электронной почты, содержащих спам (с конфликтами)

```
package demo;

import iweb2.ch5.classification.data.Email;
import iweb2.ch5.classification.rules.ClassificationResult;

global ClassificationResult classificationResult;

rule "Rule 1: Tests for viagra in subject"
salience 100
when
    email: Email( $s : subject )
    eval( classificationResult.isSimilar($s, "viagra" ) )
then
    email.setRuleFired(1);
    classificationResult.setSpamEmail(true);
end

rule "Rule 2: Tests for 'drugs' in subject"
salience 100
when
    email: Email( $s : subject )
    eval( classificationResult.isSimilar($s, "drugs" ) )
then
     email.setRuleFired(2);
     classificationResult.setSpamEmail(true);
end

rule "Rule 3: Tests for known sender address"
salience 10
when
    email: Email( $sender : from )
    eval( classificationResult.isSimilar($sender, "friend@senderhost" ) )
then
    email.setRuleFired(3);
    classificationResult.setSpamEmail(false);
end
```

Правило для выявления слова «Viagra» в теме сообщения электронной почты

Правило для выявления слова «drugs» в теме сообщения электронной почты

Это правило могло бы конфликтовать с первым или вторым

Тема сообщения электронной почты, которое мы собираемся создать из документа *spam-biz-01.html*, содержит два слова «drugs», следовательно, должно сработать второе правило. В то же время это сообщение отправлено пользователем friend@senderhost, поэтому должно выполняться третье правило. Согласно второму правилу, это сообщение является спамом; по третьему правилу это допустимое сообщение. Иными словами, для этого конкретного сообщения (факта) второе и третье правила вступают в конфликт, который нам необходимо разрешить. На выручку приходит атрибут `salience`!

Понятие *значение* (salience) в контексте систем на основе правил, вероятно, взято из *семиотики*, науки о знаках: греческое слово σημειωση

означает знак, символ или характерный признак, в зависимости от контекста. В семиотике значением называют относительную важность знака среди множества знаков, которые субъект получает в каждый момент. Аналогично, в контексте наших правил атрибут `salience` означает приоритет правила в сравнении с другими правилами, когда к конкретному факту или набору фактов применимы все правила. Чем меньше значение (`salience`) правила, тем сильнее это правило доминирует над другими правилами. В сущности, доминирование правила находит свое отражение в порядке выполнения правил. Если значение правила *X* меньше значения правила *Y*, правило *Y* будет выполнено первым, а правило *X* – последним. Где бы и когда бы ни возник конфликт между действиями правил *Y* и *X*, действия правила *X* будут доминировать над действиями правила *Y*.

Рассмотрим все это в действии. В листинге 5.12 показан сценарий, почти идентичный сценарию из листинга 5.8, кроме того, что теперь мы используем правила с конфликтами.

Листинг 5.12. Разрешение конфликта правил обработки электронной почты с помощью атрибута salience

```
EmailDataset ds = EmailData.createTestDataset();

EmailRuleClassifier classifier = new EmailRuleClassifier(
➥  "c:/iWeb2/data/ch05/spamRulesWithConflict.drl");

classifier.train();

classifier.run(ds," Hurray! No spam emails here.");
```

Результаты выполнения этого кода представлены на рис. 5.8. Как видите, для сообщения электронной почты на основе файла *spam-biz-01.html* отработало как второе, так и третье правило. Но второе правило (`salience` = 100) было выполнено первым, а третье правило (`salience` = 10) – вторым, и третье правило сбросило флажок сообщения электронной почты в состояние `NOT-SPAM`.

Это простой пример, который позволяет проследить каждый шаг и понять, как именно влияет на результаты введение атрибута правил `salience`. Реальное значение систем на основе правил заключается в том, что они могут эффективно функционировать в условиях, когда имеются тысячи сложных правил и миллионы фактов, и при этом позволяют экспериментировать с различными условиями, задавая их декларативно, вместо того чтобы вносить изменения в программный код. Одна только мысль о необходимости пробираться сквозь тысячи возможно вложенных предложений `if-then` приводит меня в дрожь!

Итак, мы рассмотрели классификацию сообщений электронной почты. Теперь вы узнали, как можно классифицировать обычный текстовый документ с помощью вероятностного наивного байесовского алгоритма, а также процессора правил Drools, предпочтительного при разработке

```
bsh % EmailDataset ds = EmailData.createTestDataset();
bsh % EmailRuleClassifier classifier = new EmailRuleClassifier(
å    "c:/iWeb2/data/ch05/spamRulesWithConflict.drl");

bsh % classifier.train();
bsh % classifier.run(ds," Hurray! No spam emails here.");

Hurray! No spam emails here.
--------------------------------------------------

Classifying email: world-01.html ...
Rules classified email: world-01.html as: NOT-SPAM

Classifying email: spam-biz-01.html ...
Invoked Email.setRuleFired(2), current value ruleFired=0,
emailId: spam-biz-01.html
Invoked ClassificationResult.setSpamEmail(true)
Invoked Email.setRuleFired(3), current value ruleFired=2,
emailId: spam-biz-01.html
Invoked ClassificationResult.setSpamEmail(false)
Rules classified email: spam-biz-01.html as: NOT-SPAM

Classifying email: sport-01.html ...
Rules classified email: sport-01.html as: NOT-SPAM

Classifying email: usa-01.html ...
Rules classified email: usa-01.html as: NOT-SPAM

Classifying email: biz-01.html ...
Rules classified email: biz-01.html as: NOT-SPAM
--------------------------------------------------
```

Рис. 5.8. *Разрешение конфликтов правил с помощью атрибута правил salience*

систем на основе правил на языке Java. В следующем разделе мы представим дополнительные алгоритмы классификации, а чтобы не заскучать, будем работать в контексте нового примера – рассмотрим случай *обнаружения мошенничества* (fraud detection).

5.4. Обнаружение мошенничества с помощью нейронных сетей

В нашем электронном мире мошенничество не редкость. Мошенничество многолико: махинации со страховками и интернет-аукционами, поддельные регистрационные формы и анкеты, заполнение которых сулит выгоду, и аферы в сфере телекоммуникаций. Если вы занимае-

тесь какой-то деятельностью в Интернете, задействуя несколько человек или юридических лиц, ценной оказывается способность выявлять случаи, когда кто-то из них играет не по правилам. В этом разделе мы рассмотрим сценарий мошеннических транзакций для случая покупки товаров. Мы покажем, как с помощью алгоритмов классификации отличать мошеннические транзакции от допустимых.

5.4.1. Сценарий выявления мошенничества в транзакционных данных

В примере мы используем модель данных – ведь никому не хочется увидеть сведения о своих транзакциях опубликованными в технической книге. Тем не менее мы постарались сделать эти данные по возможности реалистичными, чтобы получить результаты, которые вы наблюдали бы и для реальных данных. Этот сценарий подойдет всем. Предположим, вы работаете на крупный банк, выпускающий кредитные карты, и хотите гарантировать способность вашей системы быстро, если не в реальном времени, обнаруживать мошенническое поведение, с тем чтобы активизировать надлежащие механизмы защиты вашего клиента. Рассмотрим следующие типичные атрибуты, которые можно ассоциировать с транзакцией:

- Описание транзакции
- Сумма транзакции
- Место совершения транзакции

Мы создали набор описаний допустимых транзакций, который поместили в файл *descriptions.txt*, а также набор данных, которые будут рассматриваться как описания мошеннических транзакций, – файл *fraud-descriptions.txt*. Оба эти файла можно найти в каталоге *data\ch05\fraud*. У нас есть пять разных профилей пользователя, потому что привычки, связанные с расходованием денег, варьируются в зависимости от многих факторов: транзакцию на сумму 3000 долл. по одному счету можно было бы заподозрить как мошенническую, но она могла бы быть вполне допустимой для другого счета. Пяти профилей достаточно для демонстрации, разумеется, на практике «профилей», расходующих деньги, гораздо больше. Сумма транзакции берется из гауссова распределения и определяется исходя из средней суммы транзакции по данному профилю и его стандартного отклонения. Если вы не знаете, что такое гауссово распределение или стандартное отклонение, прочтите *приложение А*.

Теперь самое время сообщить вам об интригующем свойстве больших совокупностей (aggregates) транзакционных данных. Если вы занимаетесь сбором транзакционных данных из различных источников и поинтересуетесь, как часто первой значащей цифрой этих чисел будет 1, то обнаружите, что это случается намного чаще, чем можно было ожидать. Любой обычный человек (то есть не математик) скажет: по-

скольку цифр девять, вероятность увидеть цифру 1 равна 11,1%, цифру 2 – 11,1% и так далее. Так? А вот и нет! *Закон Бенфорда* (Benford's law) гласит, что вероятность должна быть логарифмической, а не однородной. Оказывается, вероятность того, что первая значащая цифра будет равна 1, составляет примерно 30%. С этим мощным статистическим фактом связана интересная история, когда в 1995 году он с успехом был использован службой окружного прокурора Бруклина для обнаружения мошенничества в семи компаниях Нью-Йорка [Hill, T.].

Возвращаясь к описанию наших транзакционных данных, мы упрощаем указание места проведения транзакции, предоставляя евклидовы координаты (x, y). В реальной системе, наверное, следовало бы получить точное описание места проведения транзакции с помощью данных глобальной системы навигации (Global Positioning System, GPS). В нашем случае вполне сгодятся обычные координаты (x, y), не усложняющие без необходимости рассматриваемый сценарий. Координаты (x, y) места проведения транзакции берутся из равномерного распределения в интервале от минимального до максимального значения. Другими словами, для каждого профиля задаются минимальное и максимальное значения как X, так и Y, и данной транзакции назначается случайное место ее проведения, которое с равной вероятностью попадает в какую-то точку из интервала, определенного для этих координат (x, y).

Вы можете поэкспериментировать с программным кодом и сгенерировать собственные данные – добавить больше профилей или новых пользователей и большее число транзакций для каждого пользователя. Удобную отправную точку для того, чтобы заняться этим, обеспечивает класс `TenUsersSample` который можно найти в пакете `iweb2.ch5.usecase.fraud.util` вместе с другими вспомогательными классами. В результате выполнения метода `main` этого класса будут сгенерированы два файла: *generated-training-txns.txt* и *generated-test-txns.txt*. Эти файлы содержат данные для обучения и тестирования соответственно, как вы, возможно, догадались. В папке *data\ch05\fraud* вы найдете данные, использованные нами для написания этого раздела; файлы с нашими данными получили имена *training-txns.txt* и *test-txns.txt*. Для обучения доступны примерно 10 000 транзакций, для тестирования – примерно 1000. Каждая транзакция определяется значениями следующих атрибутов (в порядке их перечисления):

- Идентификатор пользователя
- Идентификатор транзакции
- Описание транзакции
- Сумма транзакции
- Координата x транзакции
- Координата y транзакции

- Булевская переменная, которая определяет, является транзакция мошеннической (true) или нет (false)

Наша цель довольно проста. Мы хотим создать классификатор, способный научиться идентифицировать мошенническую транзакцию по данным транзакций из набора для обучения. После того как классификатор будет создан (обучен), потребуется протестировать его на тестовых данных, полученных из тех же статистических распределений. В следующих разделах мы намерены достичь поставленной цели посредством двух разных систем классификации. Первая основана на алгоритме нейронных сетей, вторая – на дереве решений. Оба подхода к решению задач классификации мы вкратце рассмотрели в разделе 5.2, теперь пора познакомится с ними поближе.

5.4.2. Обзор нейронных сетей

В этом разделе мы в двух словах представим важнейшие идеи, лежащие в основе нейронных сетей. Тема нейронных сетей обширна. Мы представим то, что называют *вычислительными нейронными сетями* (computational neural networks): мы стараемся не использовать термин «искусственные» (artificial), поскольку существуют реализации нейронных сетей на аппаратной основе [см., например, Maier, K. D., C. Beckstein, R. Blickhan, W. Erhard, and D. Fey]. Мы сосредоточимся на программных реализациях нейронных сетей.

Вообще говоря, нейронная сеть состоит из узлов нейронов, или просто *нейронов,* и связей между нейронами, который называются *синапсами*, или *связями* (links). Одни узлы отвечают за простую передачу данных в сеть и из нее, другие – за обработку данных. Первые узлы обеспечивают сеть возможностью ввода/вывода. Они носят подходящее название уровней *ввода* (input) и *вывода* (output) в зависимости от того, вводят они данные в сеть или экспортируют обработанные данные из сети, соответственно. Все другие узлы называются *скрытыми* (hidden) и не взаимодействуют с «внешним» миром.

Типичная нейронная сеть показана на рис. 5.9. Для данного входа, обозначенного здесь с помощью вектора $\{x1, x2, x3\}$, нейронная сеть производит выходные данные, которые являются функцией входных данных и параметров сети. На рисунке выходные данные сети обозначены как y; в общем случае выходные данные могли бы быть вектором, а не просто значением. Будем считать, что входные значения $\{x1, x2, x3\}$ распространяются слева направо. Каждый узел собирает свои входные значения и вычисляет свои выходные значения. Окончательное значение (y) зависит от входных значений $\{x1, x2, x3\}$ и от способа распространения этих значений по сети.

Синапсы соединяют узлы, в том смысле что любые два связанные синапсом узла могут обменяться информацией. Обмен информацией ре-

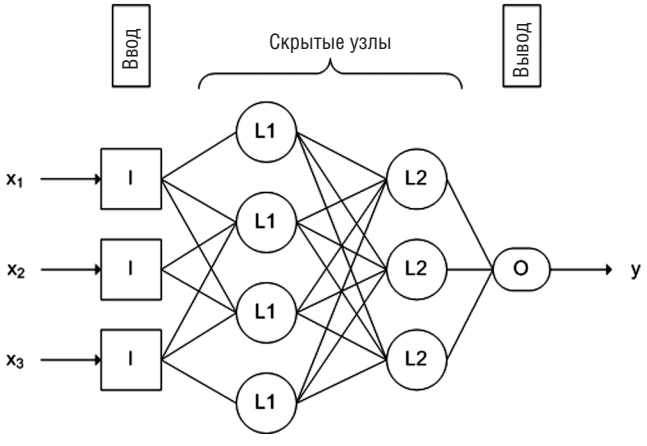

Рис. 5.9. *Типичная нейронная сеть с тремя узлами ввода, двумя скрытыми уровнями узлов и одним узлом вывода*

гулируется параметром, называемым *весом* (weight) синапса, который служит чем-то вроде индикатора важности соединения между этими двумя узлами. На этапе обучения нейронной сети эти веса синапсов постоянно оцениваются и изменяются в соответствии со значениями из набора данных для обучения.

Графическое представление нейронной сети является стандартным. Глядя на такое графическое представление, о нейронной сети можно сказать очень многое.

Во-первых, обратите внимание на то, что мы указали стрелку только в тех связях, которые помещают переменные в узлы ввода, и в той, которая обеспечивает «ответ» – значение y. Если узлы и синапсы образуют *ациклический направленный граф* (directed acyclic graph, DAG), практическим правилом для этого условия была бы проверка, все ли стрелки направлены слева направо, – в таком случае мы говорим, что у нас есть нейронная сеть *прямого распространения* (feedforward). В противном случае говорим, что есть *нейронная сеть с обратным распространением* (feedback).

Во-вторых, заметим, что мы выстроили узлы как вертикальные стеки, идущие слева направо. Это необязательно, но так принято. Мы говорим, что узлы из данного вертикального стека принадлежат данному *уровню* (layer). Следуя этому принятому соглашению, мы обозначили узлы первого скрытого уровня $L1$, а узлы второго скрытого уровня – $L2$. Узлы уровня ввода обозначены буквой I, а единственный узел уровня вывода – буквой O.

В-третьих, обратите внимание на то, что узлы ввода соединены не со всеми узлами первого скрытого уровня. Но каждый узел первого скры-

того уровня соединен с каждым узлом второго скрытого уровня. Если все узлы одного уровня соединены с каждым узлом следующего уровня, мы говорим, что эти уровни являются *полностью связанными* (fully connected).

Эти наблюдения являются психологической подготовкой к следующей мантре: мы можем полностью определить нейронную сеть, определив три важнейших элемента [MacKay, D. J. C.]:

- *Архитектуру нейронной сети* (neural network architecture), задающую число узлов в сети, число входных и выходных узлов, а также синапсы и их направленность
- *Правило активизации* (activation rule), задающее законы прямого взаимодействия между узлами
- *Правило обучения* (learning rule), задающее законы косвенного взаимодействия между узлами и информацию, которая распространяется по сети

Подобная универсальность определения нейронной сети обеспечивает огромные возможности, но в то же время затрудняет определение идеальной нейронной сети. Мы не собираемся предоставлять исчерпывающее введение в нейронные сети на нескольких страницах, с нашей стороны это было бы слишком самонадеянно. Ссылки на дополнительную литературу, посвященную нейронным сетям, можно найти в *приложении E*.

5.4.3. Детектор мошенничества на основе нейронной сети в действии

Пора сделать первые шаги в направлении использования нейронной сети, которая помогла бы выявить мошеннические транзакции. В листинге 5.13 показано, как:

- Загрузить набор данных о транзакциях и вычислить по ним статистику пользователей;
- Создать классификатор `nnfraudclassifier`, обучить его и сохранить на диске;
- Загрузить экземпляр классификатора `nnfraudclassifier` с диска и использовать его для классификации транзакций;
- Загрузить набор новых транзакций для тестирования классификатора с помощью экземпляра класса `frauderrorestimator`

Листинг 5.13. Класс NNFraudClassifier: классификатор на основе нейронной сети для обнаружения случаев мошенничества

```
TransactionDataset ds = TransactionLoader.loadTrainingDataset();   ❶

ds.calculateUserStats();   ❷
```

```
NNFraudClassifier nnFraudClassifier = new NNFraudClassifier(ds);  ❸

nnFraudClassifier.setName("MyNeuralClassifier");

nnFraudClassifier.useDefaultAttributes();  ❹

nnFraudClassifier.setNTrainingIterations(10);  ❺

nnFraudClassifier.train();  ❻

nnFraudClassifier.save();  ❼

NNFraudClassifier nnClone = NNFraudClassifier
➥  .load(nnFraudClassifier.getName());  ❽

nnClone.classify("1");  ❾

nnClone.classify("305");

TransactionDataset testDS = TransactionLoader.loadTestDataset();  ❿

FraudErrorEstimator auditor = new FraudErrorEstimator(testDS, nnClone);  ⓫

auditor.run();
```

Как видите, с нашим программным кодом создание и применение классификатора на основе нейронной сети не вызывает сложностей. Все можно запрограммировать в несколько шагов. Проанализируем эти шаги по порядку.

❶ Набор данных с транзакциями из файла *training-txns.txt* инкапсулирован в классе `TransactionDataset`. Программный код из пакетов `iweb2.ch5.usecase.fraud`.* позволяет вам сформировать свой набор данных. Мы могли бы просто предоставить сам файл с данными транзакций, оставив в стороне остальные детали. Но эта книга – практическое руководство, и проведя вас через процесс создания собственного набора данных (и возможно дополнить то, что даем мы), поможет вам смоделировать данные вашего приложения. Для интеллектуальных приложений крайне важно правильно использовать правильные данные.

❷ Получив необработанные транзакции, мы накапливаем статистическую информацию о привычках каждого пользователя, связанных с расходованием денег. Не забудьте, что на практике вы будете собирать данные с некоторого сервера или получать их из системы хранилища данных. Нам необходимо добыть данные для информации, которая поможет определить базовый уровень расходов каждого пользователя. Если пользователь *A* ограничивает свои расходы диапазоном 20–200 долл., а для пользователя *B* это диапазон 100–5000 долл., то для этих двух пользователей транзакция на сумму 2000 долл. означает нечто совершенно противоположное. Этот процесс принадлежит к общей категории предварительной обработки данных, называемой *нормализацией данных* (data normalization).

Взгляните на класс `UserStatistics`, который инкапсулирует данные о базовом уровне расходов каждого пользователя. Внимания заслуживают три вещи. Первое, о чем мы уже говорили, – это «вилка» расходов. Мы определяем минимальную и максимальную суммы *допустимой* (legitimate) транзакции, взяв их из набора данных для обучения. Во-вторых, обратите внимание на коллекцию термов, обнаруженных в описаниях допустимых транзакций. В-третьих, обратите внимание на то, что помимо минимальной и максимальной координат, ограничивающих область проведения транзакций, мы также вычисляем центроид этой области. Здесь использование понятия «центроид» оправдано тем, что большая часть транзакций совершаются в границах района проживания. Следовательно, если поступает новая транзакция и место ее проведения удалено от базы – центроида области, определенной для этого пользователя, – то этот фактор следует учитывать, хотя его вклад в получение результата не должен доминировать, поскольку люди время от времени совершают какие-то поездки.

❸ `NNFraudClassifier` – главный класс для классификации транзакций. Сам по себе этот класс не является нейронной сетью: он делегирует вопросы своего функционирования, связанные с обучением, классу `TransactionNN`. Позже в этом разделе мы изучим оба эти класса. Наш классификатор получает имя, таким образом, на него можно будет ссылаться позже. Это имя будет использовано для формирования имени файла классификатора (во время сериализации), поэтому вы должны назвать свой классификатор так, чтобы его имя описывало ваши действия.

❹ Если вы не забыли наши предыдущие примеры (особенно пример с переходами пользователя по ссылкам из главы 2), нам надо определить атрибуты транзакций, которые следует использовать для классификации. В реальном мире это важно, потому что транзакции насчитывают, как правило, десятки, если не сотни атрибутов. Чтобы выбрать атрибуты, которые будут использоваться для обучения, необходимо провести некоторый тщательный анализ. Нерелевантные атрибуты могут сбить классификатор с толку, существенно снизив его способность выявлять мошеннические транзакций. Данный метод автоматически выбирает три атрибута транзакций, о которых мы говорили: сумму, место проведения и описание.

❺ На этом шаге определяется, сколько раз данные будут пропущены через сеть. Насколько велико должно быть это значение? Зависит от ваших данных и вашей сети. Как вы увидите, для рассматриваемого примера хорошие результаты можно получить, если пропустить данные через сеть 10 раз.

❻ Настроив все параметры нашей нейронной сети, мы готовы приступить к процессу обучения. Когда этот метод завершит работу, клас-

сификатор будет готов к применению и обучен всему, чему он смог научиться, анализируя данные в предоставленном наборе для обучения.

❼ Сохраняем экземпляр обученного классификатора на диске. Это важно, потому что вам, возможно, потребуется внедрить этот классификатор в несколько систем, или потому, что классификатор создан не в той системе, в которой он будет использоваться. Кроме того, сохранение классификатора – мера защиты от сбоев системы. Представьте, что вы потратили два часа на обучение классификатора, использующего 10 атрибутов, на наборе, содержащем несколько миллионов транзакций, как вдруг по какой-то там причине система «падает»! Как быть? С обученными классификаторами следует обходиться так же, как с любым другим электронным документом, чье содержимое может измениться: если за время применения классификатора было проведено его дополнительное обучение, следует сохранить копию этого классификатора, например, на диске.

❽ А так можно загрузить обученный классификатор. Достаточно знать имя файла классификатора. В нашей реализации все сериализованные классификаторы хранятся в одном месте на диске, которое определено в константной переменной `NNFraudClassifier.SERIALIZATION_PATH`. Если это неудобно, можно изменить значение указанной переменной или соответствующий программный код, обеспечив более универсальную возможность хранения классификаторов.

❾ Приступим! Мы готовы классифицировать пару образцов. Первый идентификатор транзакции (1) соответствует допустимой транзакции, второй идентификатор транзакции (305) – мошеннической транзакции. Это «санитарная» проверка, а не скрупулезная оценка классификатора, поскольку мы отобрали транзакции, которые уже встречались этому классификатору во время обучения.

❿ Давайте создадим набор данных, содержащий транзакции, которые классификатор никогда ранее не видел. Мы называем такие данные *тестовым набором данных* (testing dataset) и обозначаем `testDS`.

⓫ `FraudErrorEstimator` – это вспомогательный класс, помогающий оценить правильность классификатора. Процедура оценки начинается с инициации метода `run()`. К концу работы этот метод подсчитывает число транзакций, классифицированных верно, и число транзакций, классифицированных неверно. То есть он отдельно сообщает число допустимых транзакций, которые классификатор посчитал мошенническими, и число мошеннических транзакций, которые классификатор счел допустимыми. Между этими двумя типами ошибочной классификации есть разница, которую мы обсудим в следующем разделе.

Прежде чем углубляться в программный код реализации нейронной сети, взглянем на результаты. На рис. 5.10 показаны выходные данные, полученные в результате выполнения кода из листинга 5.13. Ого! Что

скажете? Классификатор, по-видимому, безупречен! Возможно ли это? Скажем, весьма маловероятно. Это классическая ловушка для тех, кто использует библиотеки по принципу «черного ящика» и не понимает внутренних механизмов работы алгоритмов классификации. Чтобы разобраться в этом, откройте в своем любимом текстовом редакторе файл *test-txns.txt* и замените каждое вхождение записи `BLACK DIAMOND COFFEE` на `SOME DUDE`.

```
bsh % nnClone.classify("1");
Transaction:
  >> 1:1:EXPEDIA TRAVEL:63.29:856.0:717.0:false

Assessment:
  >> This is a VALID_TXN
bsh % nnClone.classify("305");
Transaction:
  >> 1:305:CANADIAN PHARMACY:3978.57:52.0:70.0:true

Assessment:
  >> This is a FRAUD_TXN
bsh % TransactionDataset testDS =
å  TransactionLoader.loadTestDataset();

bsh % FraudErrorEstimator auditor =
å  new FraudErrorEstimator(testDS, nnClone);

bsh % auditor.run();
Total test dataset txns: 1100, Number of fraud txns:100
Classified correctly: 1100,
Misclassified valid txns: 0,
Misclassified fraud txns: 0
```

Рис. 5.10. *Результаты классификации, полученные от классификатора на основе нейронной сети NNFraudClassifier (листинг 5.12)*

Выполнив повторно последние три шага из листинга 5.13, вы увидите результаты, показанные на рис. 5.11; ваши результаты будут также включать нормализованные значения транзакций, которые мы здесь опустили для удобства чтения. Наиболее подходящие транзакции, ассоциированные с замененным описанием, были допустимыми: их последний атрибут имел значение `false`. Согласно выходным данным, четыре допустимые транзакции (`VALID_TXN`) были ошибочно классифицированы как мошеннические. Безупречная репутация нашей оценки подорвана, поскольку замена внесла в данные *шум* (noise). Другими словами, теперь мы имеем дело с данными, которые никогда ранее не встречались.

```
bsh % TransactionDataset testDS =
å  TransactionLoader.loadTestDataset();

bsh % FraudErrorEstimator auditor =
å  new FraudErrorEstimator(testDS, nnClone);

bsh % auditor.run();

userid = 25.0 - txnid = 500523 - txnamt = 63.79 -
å  location_x = 533.0 - location_y = 503.0 -
å  description = SOME DUDE -->  VALID_TXN

userid = 26.0 - txnid = 500574 - txnamt = 127.97 -
å  location_x = 734.0 - location_y = 507.0 -
å  description = SOME DUDE -->  VALID_TXN

userid = 23.0 -  txnid = 500273 -  txnamt = 47.76 -
location_x = 966.0 -  location_y = 991.0 -
description = SOME DUDE -->  VALID_TXN

userid = 21.0 -  txnid = 500025 -  txnamt = 50.47 -
location_x = 980.0 -  location_y = 996.0 -
description = SOME DUDE -->  VALID_TXN

Total test dataset txns: 1100, Number of fraud txns:100
Classified correctly: 1096,
Misclassified valid txns: 4,
Misclassified fraud txns: 0
```

Рис. 5.11. *Внесение шума в данные путем замены описания допустимых транзакций*

Первый набор тестовых транзакций (на котором получены результаты, показанные на рис. 5.10) не включал ни единой транзакции из набора для обучения. Однако в рассматриваемом случае все тестовые транзакции были созданы из одних и тех же статистических распределений. В частности, описания транзакций были взяты из фиксированного набора описаний без каких-либо вариаций. Даже несмотря на то, что ни одна тестовая транзакция никогда прежде не встречалась[1], все они принадлежат одному и тому же конкретному пространству данных. В этом случае почти все алгоритмы классификации могут успешно отработать, но все они выдают ошибки на данных, которые существенно

[1] В том смысле, что, сравнив поочередно все значения атрибутов тестовой транзакции с значениями атрибутов всех обучающих транзакций, мы не встретили бы ни одной обучающей транзакции, идентичной тестовой.

отличаются от набора данных для обучения. Способность классификатора изящно обработать данные, которые не встречались ранее, является мерой его способности к обобщению.

Мораль истории такова: исключительно важно хорошо понимать данные. В частности, важно знать степень репрезентативности выбранных данных с точки зрения представления всех возможных данных вашего приложения. Как правило, сбор очень большого количества данных помогает получить очень много релевантных данных.

На практике сделать это не так легко, как кажется, потому что одни и те же данные могут означать разные вещи в разных контекстах. К тому же при уменьшении числа атрибутов возрастает неопределенность истолкования данных, а при его увеличении важнейшие классификационные признаки могут теряться за малозначительной информацией. Приходится тщательно балансировать между точностью классификатора в отношении того, что нам уже известно, и способностью этого классификатора к обобщениям.

Важно знать *чувствительность* (sensitivity), которую демонстрирует ваш классификатор, когда вы вводите шум. В предыдущем примере мы изменили описание для 39 транзакций из набора, где всего их было 1100, и классификатор на основе нейронной сети оказался неточным в 4 из 39 случаев «загрязненных» транзакций. Что бы произошло при выборе другой замещающей строки? Сколько «загрязненных» транзакций классифицируются неверно по мере возрастания их числа? Возьмите в качестве замещающей строки собственное имя и проанализируйте результаты.

5.4.4. Анатомия нейронной сети детектора мошенничества

Итак, пора пристальнее взглянуть на класс `NNFraudClassifier`, приведенный в листинге 5.14. Его ядро – это класс `TransactionNN`, который является нейронной сетью, специально созданной для нужд нашего случая применения – обнаружения мошенничества. В свою очередь, класс `TransactionNN` расширяет возможности универсального класса нейронной сети `BaseNN`, который можно использовать как базовый для написания собственной нейронной сети; мы будем анализировать класс `BaseNN` из листинга 5.16.

Листинг 5.14. Классификатор для обнаружения мошенничества на основе специальной нейронной сети

```
public class NNFraudClassifier
   implements Classifier, java.io.Serializable {

private String name;
private TransactionNN nn;
private TransactionDataset ds;
```

```
private transient TrainingSet ts;
private TransactionInstanceBuilder instanceBuilder;
private List<String> availableAttributeNames;

public NNFraudClassifier(TransactionDataset ds) {   ❶
  this.ds = ds;
  this.ts = ds.createTrainingDataset();
  this.instanceBuilder = ds.getInstanceBuilder();
  this.availableAttributeNames = new ArrayList<String>();

  nn = createNeuralNetwork();
}

public Concept classify(String transactionId) {   ❷
  setVerbose(true);
  Transaction t = ds.findTransactionById(transactionId);
  return classify(t);
}

public Concept classify(Transaction t) {
  return classify(instanceBuilder.createInstance(t));
}

public Concept classify(Instance instance) {   ❸

  double[] x = createNNInputs(instance);

  double[] y = nn.classify(x);

  Concept c = createConceptFromNNOutput(y);

  return c;
}

public boolean train() {   ❹

  if( ts == null ) {
     throw new RuntimeException("Can't train classifier -
➥      training dataset is NULL.");
  }
  if( nn == null ) {
     throw new RuntimeException("No Neural Network found.");
  }
  if( nn.getInputNodeCount() != availableAttributeNames.size()) {
     throw new RuntimeException("Number of attributes doesn't match");
  }
  if( nn.getOutputNodeCount() != 1) {
     throw new RuntimeException("Classifier expects network
➥      with only one output node.");
  }

  trainNeuralNetwork(nTrainingIterations);

  return true;
}
```

```
private void trainNeuralNetwork(int nIterations) {  ❺

  for(int i = 1; i <= nIterations; i++) {

      for(Instance instance : ts.getInstances().values()) {

           double[] nnInput = createNNInputs(instance);

           double[] nnExpectedOutput = createNNOutputs(instance);

           nn.train(nnInput, nnExpectedOutput);
      }
  }
}

public double[] createNNInputs(Instance instance) {  ❻

  int nInputNodes = nn.getInputNodeCount();

  double[] x = new double[nInputNodes];

  for(int i = 0; i < nInputNodes; i++) {

    String attrName = this.availableAttributeNames.get(i);
    Attribute a = instance.getAttributeByName(attrName);

    if( a instanceof DoubleAttribute ) {

      x[i] = (Double)a.getValue();

    } else {

      if( a == null ) {
         throw new RuntimeException("Failed to find attribute with name:
➥      '"+attrName);
      } else {
        throw new RuntimeException("Invalid attribute type.");
      }
    }
  }
  return x;
}

public double[] createNNOutputs(Instance i) {  ❼

  int nOutputNodes = nn.getOutputNodeCount();

  double[] y = new double[nOutputNodes];

  if( TransactionConcept.CONCEPT_LABEL_FRAUD.equals(i.getConcept().
getName()) ) {
      y[0] = 1;

  } else {
     y[0] = 0;
  }
  return y;
```

```
}
private Concept createConceptFromNNOutput(double[] y) {   ❽
  double threshold = 0.5;
  Concept c = null;
  if( y[0] >= threshold ) {
     c = new TransactionConcept(TransactionConcept.CONCEPT_LABEL_FRAUD);
  } else {
     c = new TransactionConcept(TransactionConcept.CONCEPT_LABEL_VALID);
  }
  return c;
}
public void useDefaultAttributes() {   ❾
     trainOnAttribute(TransactionInstance.ATTR_NAME_N_TXN_AMT);
     trainOnAttribute(TransactionInstance.ATTR_NAME_N_LOCATION);
     trainOnAttribute(TransactionInstance.ATTR_NAME_N_DESCRIPTION);
    }
}
```

Листинг 5.14 содержит важнейшие методы класса `NNFraudClassifier`; для краткости из него удалены Javadoc, методы `get` и `set` и так далее. Как видете, классификатор является оболочкой для более простых классов, позволяющих представить обнаружение мошеннических транзакций в виде стандартной схемы «соответствия образца концепту». Прокомментируем эти методы по порядку.

❶ Конструктор получает ссылку на набор данных с транзакциями и создает объекты, которые потребуются для классификации. Не забудьте, что нашими данными являются транзакции, поэтому необходимо создать из них образцы, чтобы можно было применять алгоритмы классификации. Эта роль отводится классу `TransactionInstanceBuilder`. Обращение к методу `createNeuralNetwork()` обеспечивает создание экземпляра класса `TransactionNN`, который описывается в листинге 5.14.

❷ Метод `classify` переопределяется для конкретного использования этого классификатора. Согласно интерфейсу `iweb2.ch5.classification.core.intf.Classifier`, классификатор обязан предоставить реализацию, которая в качестве единственного аргумента получает образец и возвращает концепт. Мы облегчаем использование нашего классификатора, предоставляя дополнительные методы `classify`, которые, в конечном счете, делегируют выполнение своих функций главному методу `classify`.

❸ Это самый важный метод классификатора, и его реализация включает три шага. Наши нейронные сети получают на входе массив значений типа `double` и обеспечивают на выходе массив значений типа

double. Первый шаг – трансляция данных образца транзакции в массив значений типа double. Второй шаг – загрузка входных значений в сеть и получение результатов классификации, выполненной нейронной сетью (единственное значение типа double). Поскольку мы не заинтересованы в точности значения типа double, возвращаемого нейронной сетью, но хотим, чтобы классификатор ответил, является ли этот образец мошенничеством, необходимо транслировать результирующее значение типа double в один из двух концептов: CONCEPT_LABEL_FRAUD или CONCEPT_LABEL_VALID. Именно этим и занимается метод createConceptFromNNOutput.

❹ Это обучающий метод, который необходимо вызвать для классификатора NNFraudClassifier и результатом работы которого является обучение нейронной сети. Во-первых, этот метод выполняет ряд проверок, прежде чем делегировать главному обучающему методу. В частности, он проверяет соблюдение следующих условий:

- Существование набора для обучения
- Существование экземпляра класса transactionnn
- Соответствие входных данных спецификациям экземпляра класса transactionnn
- Соответствие выходных данных спецификациям экземпляра класса transactionnn

❺ Это главный обучающий метод. Он получает единственный аргумент – число, которое определяет, сколько раз образцы обучающего набора должны быть пропущены через нейронную сеть. Каждый образец изменяет, в итоге, веса синапсов нейронной сети, чтобы оптимизировать результаты классификации, выполненной нейронной сетью, для всех образцов, которые уже прошли через сеть. Другими словами, вы все время говорите нейронной сети, каким должен быть вывод для данного ввода, и сеть пытается настроиться так, чтобы суметь «запомнить» ответ, «не забыв» все остальные ответы, уже виденные ею ранее.

❻ Вспомогательный метод, который получает в качестве аргумента образец и создает входные значения для нейронной сети.

❼ Вспомогательный метод, который получает в качестве аргумента образец и создает выходные значения нейронной сети. Этот метод используется только на стадии обучения.

❽ Вспомогательный метод, который получает в качестве аргумента выходное значение, предоставляемое нейронной сетью, и транслирует его в один из двух концептов: CONCEPT_LABEL_FRAUD или CONCEPT_LABEL_VALID.

❾ Вспомогательный метод, определяющий атрибуты образца транзакции, которые мы хотим использовать для классификации. В данном

случае у нас мало атрибутов, но такая оболочка упрощает сценарии. В общем случае удобно и целесообразно определить список атрибутов для обучения в одном месте программного кода. Можно было бы добавить также метод для получения переменной `availableAttributeNames`.

К этому моменту вы, скорее всего, усвоили высокоуровневое определение нашего классификатора мошенничества на основе нейронной сети. Но как нам определить нейронную сеть? Какие шаги следует предпринять, чтобы написать собственный классификатор для выявления мошенничества с помощью другой нейронной сети? В листинге 5.15 показан программный код из класса `TransactionNN`. Это та нейронная сеть, которую мы используем в классификаторе мошенничества, но, как видите, в определении этого класса нет ничего специального, относящегося к мошенничеству или транзакциям. Он содержит только сигнатуру для нашего решения о представлении задачи обнаружения мошенничества в виде нейронной сети.

Листинг 5.15. Специальная нейронная сеть для обнаружения мошенничества

```
public class TransactionNN extends BaseNN {  ❶

    public TransactionNN(String name) {  ❷
        super(name);

        createNN351();
    }

    private void createNN351() {

        Layer inputLayer = createInputLayer(  ❸
                0, // Идентификатор уровня
                3  // Число узлов
                );

        Layer hiddenLayer = createHiddenLayer(  ❹
                1, // Идентификатор уровня
                5, // Число узлов
                new double[] {1, 1.5, 1, 0.5, 1} // Погрешности узлов
                );

        Layer outputLayer = createOutputLayer(  ❺
                2, // Идентификатор уровня
                1, // Число узлов
                new double[] {1.5} // Погрешности узлов
                );

        setInputLayer(inputLayer);  ❻
        setOutputLayer(outputLayer);
        addHiddenLayer(hiddenLayer);
```

```
        setLink("0:0", "1:0", 0.25);   ❼
        setLink("0:0", "1:1", -0.5);
        setLink("0:0", "1:2", 0.25);
        setLink("0:0", "1:3", 0.25);
        setLink("0:0", "1:4", -0.5);

        setLink("0:1", "1:0", 0.25);
        setLink("0:1", "1:1", -0.5);
        setLink("0:1", "1:2", 0.25);
        setLink("0:1", "1:3", 0.25);
        setLink("0:1", "1:4", -0.5);

        setLink("0:2", "1:0", 0.25);
        setLink("0:2", "1:1", -0.5);
        setLink("0:2", "1:2", 0.25);
        setLink("0:2", "1:3", 0.25);
        setLink("0:2", "1:4", -0.5);

        setLink("1:0", "2:0", -0.5);
        setLink("1:1", "2:0", 0.5);
        setLink("1:2", "2:0", -0.5);
        setLink("1:3", "2:0", -0.5);
        setLink("1:4", "2:0", 0.5);

        System.out.println("NN created");
    }
}
```

В этом листинге определение нашей нейронной сети для обнаружения мошенничества включает следующие шаги.

❶ Класс `TransactionNN` — расширение класса `BaseNN`, которое мы подробнее опишем позже. Вы можете создавать универсальные нейронные сети, дополняя класс `BaseNN`. В сопровождающем эту книгу программном коде можно также найти нейронную сеть, которая моделирует логический элемент XOR, то есть получает два значения типа `double` на входе и создает одно значение типа `double` на выходе. Если оба значения приблизительно равны единице или нулю, на выходе будет ноль. Если одно значение приблизительно равно единице, а другое приблизительно равно нулю, на выходе будет единица. Этот класс называется `XORNetwork`, и он еще проще, чем класс `TransactionNN`. Прочтите код этого класса и выполните его, чтобы упрочить свое понимание того, как создается нейронная сеть.

❷ Этот конструктор делегирует конструктору `BaseNN` все шаги базовой инициализации и создает конкретную топологию сети с тремя узлами ввода, пятью скрытыми уровнями и одним узлом уровня вывода. Стоп. Три входных узла? У наших транзакционных данных, которые мы описали ранее, было гораздо больше значений атрибутов. А именно, мы включили в них идентификатор пользователя, сумму

транзакции, место проведения транзакции, представленное двумя координатами, а также строку с описанием транзакции. Строка описания не является числом, но ее можно каким-либо способом преобразовать в число, и разумно предположить, что это добавило бы во входные данные по крайней мере один узел. В сумме это дает пять входных значений (как минимум), так почему мы используем только три?

Все значения данных, которые мы передаем в качестве входных в нейронную сеть, являются нормализованными значениями; чтобы убедиться в этом, взгляните на метод `createInstance(Transaction t)` класса `TransactionInstanceBuilder`. Сумма транзакции нормализована по минимальному и максимальному значениям допустимых транзакций, так что ее значение всегда находится в интервале от 0 до 1. Такой же результат для описания транзакции мы получаем с помощью `JaccardCoefficient`. С местом проведения транзакции чуть сложнее. Мы нормализуем как местоположение центроида пользователя, так и место проведения транзакции (на основании минимального и максимального значений координат x и y), а затем вычисляем расстояние между этими двумя нормализованными точками. Это расстояние и является одним из наших трех входных значений нейронной сети `TransactionNN`. Вот почему у нас только три входных узла. Понятно, что это выбор проекта, и достаточно произвольный, что характерно для проектирования нейронных сетей. Но это неплохой вариант, и его можно уточнить; в сущности, этим мы и предлагаем вам заняться в одном из пунктов раздела «Сделать».

Общая топология сети (3/5/1 узлов, только один скрытый уровень, полная связанность) также является проектным решением, которое не является догмой и может быть оптимизировано экспериментальным путем. Вы можете попытаться выделить топологии, которые дали бы в результате разные классификаторы. В том же пункте раздела «Сделать», где речь идет о нормализации данных, мы советуем вам реализовать свою сетевую топологию и сравнить результаты полученных классификаторов. Какая топология предпочтительнее – зависит от природы ваших входных данных и самой задачи. Случай обнаружения мошенничества и базовая реализация класса `TransactionNN` обеспечивают фундамент, помогающий исследовать эту зависимость.

❸ Создаем уровень ввода, предоставляя идентификатор и указывая, что этот уровень входа должен иметь три узла.

❹ Создаем скрытый уровень сети. Обратите внимание: здесь мы ввели массив новых параметров (каждое значение в этом массиве соответствует одному узлу уровня), которые называются *погрешностями* (biases). Ранее мы говорили о весах синапсов. Пока что считайте, что

это дополнительный постоянный вес, означающий погрешность, которую следует прибавить к выходному значению узла.

Заметим также, что этот метод использует префикс *add* вместо *set*. Это сделано намеренно, поскольку мы хотим подчеркнуть, что у вас может быть больше одного скрытого уровня. Рекомендуем вам изучить влияние числа скрытых уровней как один из вопросов проектирования структуры нейронных сетей.

❺ Последний из наших трех уровней. Мы также определяем значение погрешности для уровня вывода, но вы, возможно, предпочтете не иметь погрешности в выходном узле. В последнем случае просто задайте для погрешности значение, равное нулю.

❻ Назначаем связи этих трех уровней сети. К этому моменту все узлы готовы. Единственное, что осталось сделать, это создать связи (ребра) нашей сети.

❼ Поочередно создаем все связи (синапсы) между узлами. Первый аргумент определяет начало связи в виде `Идентификатор уровня:Идентификатор узла`. Второй аргумент определяет конец связи в виде `Идентификатор уровня:Идентификатор узла`. Третий аргумент определяет вес связи на стадии инициализации. Как мы показали, во время обучения значения весов постоянно меняются.

5.4.5. Базовый класс для создания универсальных нейронных сетей

В предыдущих разделах мы рассматривали конкретный случай обнаружения мошенничества. Создавая нейронную сеть, а также каждый раз, когда требуется доступ к внутренним механизмам нейронной сети, мы делегируем обращения к предоставленной нами общей реализации – классу `BaseNN`. В данном разделе мы рассмотрим этот класс, поскольку он важен и универсален в применении.

Для наглядности мы представим этот класс в виде двух листингов. Листинг 5.16 отображает этот класс с точки зрения создания структуры сети (настройка нейронной сети), а листинг 5.17 – с точки зрения ее функционирования (программный код, связанный с обучением и классификацией).

Листинг 5.16. Класс BaseNN (структура сети): фрагмент базового класса универсальной нейронной сети

```
public Layer createInputLayer(int layerId, int nNodes) {   ❶
    BaseLayer baseLayer = new BaseLayer(layerId);

    for(int i = 0; i < nNodes; i++) {
        Node node = createInputNode(layerId + ":" + i);
        Link inlink = new BaseLink();
```

```
        inlink.setFromNode(node);
        inlink.setWeight(1.0);
        node.addInlink(inlink);
        baseLayer.addNode(node);
    }

    return baseLayer;
}

public Layer createHiddenLayer(int layerId, int nNodes, double[] bias) {   ❷

    if( bias.length != nNodes ) {
      throw new RuntimeException("Each node should have bias.");
    }
    BaseLayer baseLayer = new BaseLayer(layerId);
    for(int i = 0; i < nNodes; i++) {
        Node node = createHiddenNode(layerId + ":" + i);
        node.setBias(bias[i]);
        baseLayer.addNode(node);
    }
    return baseLayer;
}

public Layer createOutputLayer(int layerId, int nNodes, double[] bias) {   ❸

    if( bias.length != nNodes ) {
      throw new RuntimeException("Each node should have bias.");
    }

    BaseLayer baseLayer = new BaseLayer(layerId);
    for(int i = 0; i < nNodes; i++) {
        Node node = createOutputNode(layerId + ":" + i);
        node.setBias(bias[i]);
        baseLayer.addNode(node);
    }
    return baseLayer;
}

public void setLink(String fromNodeId, String toNodeId, double w) {   ❹
    Link link = new BaseLink();
    Node fromNode = allNodes.get(fromNodeId);
    if( fromNode == null ) {
        throw new RuntimeException("Unknown node id: " + fromNodeId);
    }
    Node toNode = allNodes.get(toNodeId);
    if( toNode == null ) {
      throw new RuntimeException("Unknown node id: " + toNodeId);
    }
    link.setFromNode(fromNode);
    link.setToNode(toNode);
    link.setWeight(w);
```

```
        fromNode.addOutlink(link);
        toNode.addInlink(link);
    }

    protected Node createInputNode(String nodeId) {   ❺
      Node node = new LinearNode(nodeId);
      node.setLearningRate(learningRate);
      return node;
    }

    protected Node createHiddenNode(String nodeId) {
      Node node = new SigmoidNode(nodeId);
      node.setLearningRate(learningRate);
      return node;
    }

    protected Node createOutputNode(String nodeId) {
      Node node = new LinearNode(nodeId);
      node.setLearningRate(learningRate);
      return node;
    }

    public abstract double fireNeuron();

    public abstract double fireNeuronDerivative();
    }
```

Начнем с вопросов, относящихся к структуре, как показано в листинге 5.16. Здесь представлена не вся реализация. Мы оставили минимум методов, необходимых для описания структуры нейронной сети.

❶ Этот метод обеспечивает создание уровня ввода сети; в качестве аргументов он получает идентификатор уровня и число узлов, которые должен иметь этот уровень. В этом методе создается экземпляр класса `BaseLayer` – базовой реализации уровня нейронной сети в нашей системе. Этот класс состоит из идентификатора уровня и списка узлов. Для создания всех узлов уровня ввода цикл выполняется `nNodes` раз. Каждому узлу уровня ввода назначается связь (синапс), которую мы называем `inlink`, подчеркивая, что эта связь отвечает за передачу данных в сеть. Вес такой связи получает значение, равное единице, и не меняется во время обучения, потому что мы не хотим искажать исходные значения данных. По этой причине многие авторы не считают сам уровень ввода частью нейронной сети.

❷ Этот метод обеспечивает создание скрытого уровня сети; в качестве аргументов он получает идентификатор уровня, число узлов, которые должен иметь этот уровень, и погрешность каждого из этих узлов. Проверив, совпадает ли количество значений погрешности с количеством узлов, этот метод создает экземпляр класса `BaseLayer`. Цикл выполняется `nNodes` раз, чтобы создать все узлы скрытого уров-

ня. Каждому узлу уровня ввода назначается погрешность, соответствующая номеру цикла; поскольку это этап создания, видимо, такой порядок был предусмотрен проектом. В отличие от случая узлов уровня ввода, связь (синапс) на этом этапе не создается, следовательно, и вес ей не назначается. Как мы скоро увидим, это выполняется отдельно, с помощью метода `setLink`.

❸ Создание уровней нейронной сети завершается созданием уровня вывода. Создание уровня вывода похоже на создание скрытого уровня. Но есть неявное отличие, связанное с тем, что узлы уровня вывода являются экземплярами класса `LinearNode`, тогда как узлы скрытого уровня – это экземпляры класса `SigmoidNode`.

❹ Предыдущие методы отвечали за создание узлов нейронной сети. Этот метод отвечает за создание связей нейронной сети (синапсов). Пока мы создали связи только для узлов уровня ввода. Остальные узлы связываются с помощью этого метода. Его аргументы: идентификаторы двух узлов, которые должна соединить создаваемая связь, и вес, который назначается создаваемой связи. Можно также определить другие методы, например метод `connectFully(Layer x, Layer y)`, который создавал бы связи для всех возможных комбинаций узлов этих двух уровней. В качестве эксперимента вы можете исследовать имеющиеся возможности в соответствии со своими потребностями.

❺ Остальные методы, представленные в листинге 5.16, отвечают за создание экземпляров классов для конкретных реализаций узлов нейронной сети. Мы написали две конкретные реализации абстрактного класса `BaseNode`. Первую реализацию обеспечивает класс `LinearNode`; ее используют для узлов уровней ввода и вывода. Вторую реализацию обеспечивает класс `SigmoidNode`; ее используют для узлов скрытого уровня. После того как узлы созданы, мы задаем *коэффициент обучения* (learning rate).

Базовый класс обеспечивает почти всю функциональность узла. Классы `LinearNode` и `SigmoidNode` предлагают реализации только двух методов: `fireNeuron()` и `fireNeuronDerivative()`. Если вы не забыли нашу мантру проектирования из раздела 5.4.2, мы можем полностью определить нейронную сеть, определив архитектуру сети, правило активизации и правило обучения. Создание уровней сети, их узлов и связей обеспечивает создание сетевой архитектуры, но ничего не говорит о том, как будут реагировать узлы на данный вход (правило активизации), или о том, как сеть будет обучаться. Метод `fireNeuron()` определяет отклик узла-нейрона на данный вход, то есть решает главную задачу правила активизации, а метод `fireNeuronDerivative()` (обеспечивающий числовой результат работы метода `fireNeuron()`) напрямую связан с правилом обучения. Параметр `learningRate` не зависит от конкретной реализации узла и, как правило, имеет значение из интервала от 0 до 1.

Предыдущие методы в достаточной мере определяют структуру представления нейронной сети. Перейдем к листингу 5.17, иллюстрирующему функционирование нашей сети.

Листинг 5.17. Класс BaseNN (функционирование): фрагмент базового класса универсальной нейронной сети

```
public void train(double[] tX, double[] tY) {
  double lastError = 0.0;
  int i = 0;

  while( true ) {  ❶
    i++;
    double[] y = classify(tX);

    double err = error(tY, y);

    if( Double.isInfinite(err) || Double.isNaN(err) ) {  ❷
      throw new RuntimeException("Training failed.");
    }

    double convergence = Math.abs(err - lastError);

    if(err <= ERROR_THRESHOLD ) {  ❸
      lastError = err;
      break;
    }

    if( convergence <= CONVERGENCE_THRESHOLD ) {  ❹
      break;
    }
    lastError = err;

    outputLayer.setExpectedOutputValues(tY);  ❺

    outputLayer.calculateWeightAdjustments();

    for(Layer hLayer : hiddenLayers) {
       hLayer.calculateWeightAdjustments();
    }

    outputLayer.updateWeights();  ❻

    for(Layer hLayer : hiddenLayers) {
       hLayer.updateWeights();
    }
  }
}

public double[] classify(double[] x) {  ❼

  inputLayer.setInputValues(x);

  inputLayer.calculate();
  inputLayer.propagate();
```

```
    for(Layer hLayer : hiddenLayers) {
       hLayer.calculate();
       hLayer.propagate();
    }

    outputLayer.calculate();
    double[] y = outputLayer.getValues();
    return y;
  }
```

Каждая нейронная сеть имеет две основные функциональные характеристики. Сеть должна быть в состоянии обучить себя и суметь классифицировать свой вход – создать ожидаемые выходные значения. Алгоритм, который мы адаптировали в своей реализации, называется алгоритмом *обратного распространения* (back propagation); это алгоритм обучения по методу градиентного спуска в режиме онлайн. На практике этот алгоритм анализирует каждый обучающий образец и настраивает веса связей (синапсов), так чтобы полученное выходное значение минимально отличалось от ожидаемого. Минимизация полагается на анализ перепада ошибки. Для каждого объекта инициируется бесконечный цикл, который прерывается по трем условиям.

❶ Инициируем открытый цикл, в котором пытаемся получить более высокую степень правильности результатов классификатора. Условия прекращения цикла описываются в пп. 2–4.

❷ Первое условие прекращения цикла: нам удалось вычислить ошибку. Если мы не можем вычислить ошибку, нет смысла продолжать итерации. Это всего лишь «санитарная» проверка. Соблюдение указанного условия, как правило, свидетельствует о неудачной структуре нейронной сети или о какой-то другой ошибке в том подходе, с помощью которого вы пытаетесь решить свою задачу посредством классификации на основе нейронных сетей.

❸ Второе условие завершения цикла связано с величиной погрешности. Если ошибка классификации данного образца не превышает заданный порог, можно остановиться.

❹ Третье условие завершения цикла – это проверка, которая позволяет установить, наблюдается ли со временем существенное сокращение расхождения в получаемых ошибках. Мы пытаемся уменьшить ошибку, следовательно, продолжаем варьировать веса и стараемся получить лучший вариант выходного значения. Возможно, мы не сможем достичь заданного нами порогового значения ошибки. Другими словами, мы можем установить слишком высокую планку для классификатора. Рассмотрите наш способ реализации этого условия. Может быть, вы предложите более удачный критерий сходимости?

❺ На этом этапе еще не встретился ни один из наших критериев прекращения цикла. Следовательно, присваиваем в качестве значения

узла вывода ожидаемое значение и приступаем к переоценке весов сети. Это делается путем обращения к методу `calculateWeightAdjustments()` для всех узлов, начиная с узлов уровня вывода.

❻ На предыдущем шаге мы вычислили поправки весов, но не предприняли никаких действий. На этом шаге, завершив вычисление поправок весов для всех узлов, мы обновляем значения весов путем обращения к методу `updateWeights()`. Вот! Цикл завершен, и мы готовы повторять его, пока не будет выполнено одно из трех определенных нами условий его завершения.

❼ Этот метод является оболочкой верхнего уровня для процесса классификации. Как уже говорилось, если нейронная сеть функционирует, считайте, что информация распространяется по узлам от входных к выходным. Это распространение сконцентрировано в данном методе. Мы начинаем с узлов уровня ввода, переходим к скрытым уровням и в конце концов вычисляем выходное значение сети. Каждый узел выполняет собственные вычисления на основании весов и погрешности данного узла; об этом заботится метод `calculate()`. С помощью метода `propagate()` узел передает свое выходное значение связанным с ним узлам. После обучения нейронной сети логика ее функционирования довольно проста.

Большая часть этого материала опирается на познания в математике, которых мы не предполагаем у основной аудитории этой книги. Мы сконцентрировались не на научных основах классификаторов нейронных сетей, но на механике их работы. Если вас интересует более глубокое изучение внутренних механизмов нейронных сетей, есть масса литературы, к которой вы можете обратиться. В *приложении E* мы указали много хороших книг, посвященных нейронным сетям, – они помогут вам пополнить знания в этой области.

В предыдущем разделе мы сосредоточились на определении и описании алгоритмов классификации, необходимых для создания классификатора. С точки зрения готового продукта, есть еще несколько важных вопросов, таких как достоверность классификации, непротиворечивость результатов, полученных на больших наборах данных, а также вычислительные требования классификации, которые необходимо учитывать. Некоторые из этих вопросов мы рассмотрим в следующих разделах.

5.5. Можно ли доверять полученным результатам?

Допустим, вы создали свой классификатор на основе теоремы Байеса или нейронных сетей, или чего-то еще. Как понять, хорошо ли вы справились с делом? Как узнать, что пора сдать свой интеллектуальный мо-

дуль в эксплуатацию, пожиная благоговение коллег и похвалы начальства? Оценить классификатор не менее важно, чем создать его. В «деловых кругах» (именуемых также «совещаниями по организации сбыта») вам доведется услышать самые разные мнения, от преувеличений до полной чепухи. Цель этого раздела – помочь вам оценить созданный классификатор, если вы разработчик, и понять, соответствуют (или нет) необходимым законам, правилам и принципам продукты сторонних производителей, если вы разработчик или менеджер продукта.

Для начала заявляем, что не существует классификатора, который бы обеспечил успешную классификацию любой задаче и любому набору данных. Считайте, что это «оцифрованный» вариант утверждений «никто не знает всего» и «все совершают ошибки». Методики обучения, которые мы рассмотрели в контексте классификации, принадлежат категории обучения с учителем (пример алгоритма обучения без учителя ищите в соответствующем пункте раздела «Сделать»). Здесь происходит «обучение с учителем», потому что процедура обучения основана на известных классификациях, когда классификатор пытается с помощью «учителя» усвоить информацию, содержащуюся в наборе данных для обучения. Как можно представить, для того чтобы классификация оказалась успешной, крайне важна взаимосвязь данных для обучения с реальными данными при внедрении вашего классификатора.

Для ясности введем несколько понятий. Чтобы не усложнять, рассмотрим стандартную задачу двоичной классификации, такую как выявление спама в электронной почте или мошенничества. Например, сделаем вид, что пытаемся распознать, надо ли характеризовать конкретное сообщение электронной почты как спам. Главным инструментом оценки достоверности классификатора и, как правило, отправной точкой такого анализа является *матрица неточностей* (confusion matrix). Это простая матрица, где строки соответствуют категории, которую классификатор назначает конкретному образцу, а столбцы – той категории, к которой указанный образец принадлежит. В случае двоичной классификации в этой матрице только четыре элемента. Общий случай (многоклассовая классификация) принципиально не отличается от случая двоичной классификации, но требует более сложного анализа.

В табл. 5.1 представлена матрица неточностей для двоичной классификации вроде фильтрации спама в электронной почте или обнаружения мошенничества. В этой таблице зафиксированы возможные результаты двоичной классификации. Если в результате классификации конкретное сообщение электронной почты было отнесено к категории спама, мы говорим, что классификация является *положительной* (positive). В противном случае мы говорим, что классификация является *отрицательной* (negative). При этом сама классификация может оказаться правильной (истинной, true) или неправильной (ложной, false). Таким образом, матрица неточностей содержит четыре возможных выходных

значения – все возможные комбинации вариантов положительная/отрицательная и истинная/ложная. Отсюда вывод, что есть два вида ошибок. Первый – ложные положительные классификации; такие ошибки называют *ошибкой I типа* (type I error). Второй – ложные отрицательные классификации; такие ошибки типа называют *ошибкой II типа* (type II error). Проще говоря, совершая ошибку I типа, вы приговариваете невиновного, а совершая ошибку II типа, – освобождаете виновного! Эта аналогия особенно хорошо высвечивает важность *стоимости классификации* (classification cost). Вольтер предпочел бы отпустить 100 виновных, чем осудить одного невиновного; подобная деликатность еще сохраняется в европейских судах. Мораль этой истории в том, что у решений есть последствия, и эти последствия несоизмеримы. Особенно это верно в случае многоклассовой классификации. Мы еще вернемся к этому.

Таблица 5.1. Типичная матрица неточностей для простой задачи двоичной классификации

	Положительная	**Отрицательная**
Истинная	Истинная положительная (True Positive, TP)	Истинная отрицательная (True Negative, TN)
Ложная	Ложная положительная (False Positive, FP)	Ложная отрицательная (False Negative, FN)

На основе значений из табл. 5.1 введем следующие определения:

- *Коэффициент FP (FP rate, FPR)* $= FP / N$, где $N = TN + FP$
- *Специфичность (Specificity)* $= 1 -$ *Коэффициент FP* $= TN / N$
- *Полнота (Recall)* $= TP / P$, где $P = TP + FN$
- *Точность (Precision)* $= TP / (TP + FP)$
- *Правильность (Accuracy)* $= (TP + TN) / (P + N)$
- *F-балл (F-score)* = *Точность* × *Полнота*

Предположим, мы нашли классификатор, о котором заранее известно, что его правильность 75%. Насколько близка к истинной эта оценка правильности классификатора? Другими словами, если повторить классификацию применительно к другим данным, какова вероятность того, что правильность классификатора оказалась бы равной 75%? Чтобы ответить на этот вопрос, обратимся к тому, что в статистике известно как *процесс Бернулли*. Этот процесс описывается как последовательность независимых событий, для которых результат рассматривается либо как успешный, либо как неуспешный. Это превосходный пример для рассмотренных нами случаев фильтрации спама в электронной почте

или обнаружения мошенничества, и для любой двоичной классификации вообще. Если мы обозначаем истинную правильность A^*, а измеренную правильность A, то нам надо знать, является ли A хорошей оценкой для A^*.

Может быть, вам знакомо, например из прослушанного вами курса статистики, понятие *доверительного интервала* (confidence interval). Это мера доверия, которую мы назначаем конкретному утверждению. Если правильность классификатора, полученная для набора из 100 сообщений электронной почты, равна 75%, степень доверия может оказаться не очень высокой. Но если правильность равна 75% для набора из 100 000 сообщений электронной почты, степень доверия, скорее всего, будет намного выше. Интуитивно мы понимаем, что по мере возрастания количества данных в наборе доверительный интервал должен сужаться, и мы больше доверяем полученным результатам. В частности, можно показать, что для процесса Бернулли со 100 образцами истинная правильность попадает в интервал от 69,1 до 80,1%, со степенью доверия 80% [Witten, I. and Frank, E.]. Если размер набора данных, используемого для измерения правильности классификатора, увеличить в 10 раз, новый доверительный интервал имеет границы от 73,2 до 76,7% с той же степенью доверия (80%). В любом хорошем учебнике статистики приведены формулы для вычисления этих интервалов. Теоретически результаты являются допустимыми, если размер выборки превышает 30 образцов. На практике следует использовать столько образцов, сколько возможно.

К сожалению, на практике у вас может не оказаться столько образцов, сколько хотелось бы. Чтобы решить эту проблему, коллеги, занимающиеся машинным обучением, разработали ряд методик, помогающих оценить степень доверия (credibility) для результатов классификации в том случае, когда данных недостаточно. Стандартный метод оценки называется *десятикратной перекрестной проверкой* (10-fold cross-validation). Это простая процедура, которую лучше проиллюстрирует пример. Допустим, у нас есть 1000 сообщений электронной почты, которые мы уже классифицировали вручную. Для оценки классификатора необходимо часть этих сообщений использовать как набор для обучения, а часть – как набор для тестирования. Десятикратная перекрестная проверка говорит о том, что 1000 сообщений электронной почты надо разбить на 10 групп по 100 сообщений; каждый пакет из 100 сообщений должен включать допустимые сообщения и сообщения со спамом примерно в таком же соотношении, в каком они содержатся в наборе из 1000 сообщений. Затем мы берем 9 из этих групп сообщений электронной почты и обучаем классификатор. Завершив обучение, тестируем классификатор с помощью группы из 100 сообщений, которая не была включена в набор для обучения. Можно вычислить метрики (некоторые из них мы упоминали ранее) – как правило, люди измеряют правильность классифи-

катора. Этот опыт повторяется 10 раз, и каждый раз мы исключаем из набора для обучения другую группу из 100 сообщений электронной почты. Завершив эти испытания, мы имеем 10 значений правильности, из которых теперь можем вывести среднее значение правильности.

Вы спросите, изменится ли значение правильности, если разделить исходный набор на 8 или 12 частей. Разумеется, вы вряд ли получите идентичный ответ. Тем не менее новое среднее значение правильности должно быть достаточно близким полученному ранее значению. Результаты, полученные на большом количестве тестов, для разных наборов данных и с применением нескольких различных классификаторов, дают понять, что десятикратная перекрестная проверка дает достаточно представительную оценку вашего классификатора.

Доводя процесс десятикратной перекрестной проверки до крайности, вы всегда можете использовать в качестве набора для обучения все образцовые сообщения электронной почты, кроме одного, и использовать это одно, исключенное из набора для обучения сообщение для тестирования. Как и следовало ожидать, такая методика называется *исключение-по-одному* (leave-one-out). У этой методики есть определенные теоретические преимущества, но для реальных наборов данных (с сотнями тысяч, если не миллионами образцов) высокая стоимость вычислений обычно считается недостатком. Возможно, вы бы предпочли исключить из набора для обучения один какой-то образец, а не все образцы поочередно. Подобное желание породило методику *самонастройки* (bootstrap). Основная идея самонастройки: можно создать набор для обучения путем выборки образцов из исходного набора данных с замещением. Другими словами, можно использовать образец из исходного набора данных неоднократно и создать набор для обучения, состоящий из 1000 сообщений электронной почты, в котором конкретный образец может появляться несколько раз. Поступив так, в итоге вы получите тестовый набор, насчитывающий примерно 368 образцов сообщений электронной почты, которые не были использованы в наборе для обучения. Размер набора для обучения остается равным 1000 образцам сообщений электронной почты, потому что некоторые из оставшихся 632 образцов сообщений повторяются в наборе для обучения; более подробное математическое истолкование этих чисел ищите в [Witten, I. and Frank, E.].

Оказалось, что для анализа степени доверия к классификатору полезно строить график коэффициента TP (TP rate, TPR) в сравнении с коэффициентом FP (FPR). Эти графики называют *ROC-кривыми* (ROC curves), они появились в теории определения сигналов в 1970-е годы. В последнее время в очень многих работах из области машинного обучения для анализа производительности одного или нескольких классификаторов использовались ROC-кривые. Основная идея состоит в том, что ROC-

кривая должна находиться как можно дальше от диагонали графика TPR/FPR. Мы уступим анализ ROC-кривых превосходному техническому докладу [Fawcett, T.], включающему псевдоалгоритмы и множество советов по решению вопросов, возникающих на практике.

В реальном мире системы классификации часто используются для поддержки систем принятия решений; ошибки классификации могут привести к принятию неверных решений. Иногда принятие неверных решений, хотя и нежелательное, может оказаться относительно безвредным. Но порой речь идет о жизни и смерти; подумайте о враче, который не сможет диагностировать рак, или об опасности для астронавта в открытом космосе, если они полагаются на результат работы вашего классификатора. Оценка систем классификации должна была бы анализировать как степень доверия, так и связанную с ней стоимость классификации. Для двоичной классификации можно назначить *функцию стоимости* (cost function), которая является функцией коэффициентов FP и FN. О назначении стоимости в случае многоклассовой классификации говорится в одном из пунктов раздела «Сделать».

Таким образом, один из наиболее важных вопросов, связанных с классификаторами, – степень доверия к результатам их работы. В этом разделе мы рассмотрели ряд метрик, таких как точность, правильность, полнота и специфичность, помогающих оценить степень доверия к результатам работы классификаторов. Результатом объединения этих метрик могут быть новые метрики, такие как F-балл. Мы также рассмотрели идею о перекрестной проверке результатов путем разбиения разными способами набора для обучения на группы и наблюдения за тем, как меняются метрики классификатора с изменением наборов данных. Мы рассмотрели понятие ROC-кривой, которая является простым графиком, связывающим коэффициенты TPR и FPR. В следующем разделе мы рассмотрим ряд вопросов, связанных с большими наборами данных.

5.6. Классификация очень больших наборов данных

Многие наборы данных, используемые в учебных и исследовательских целях, довольно малы по сравнению с практическими реализациями. Набор данных о транзакциях крупной корпорации насчитывает от 10 до 100 миллионов записей, если не больше; требования по гарантии, файлы журналов в системах телекоммуникаций, записи биржевых курсов, журналы клавиатурных перехватчиков, аудиторские журналы и так далее (длинный список) имеют размеры того же порядка. Следовательно, обработка больших наборов данных – это правило, а не исключение в промышленных приложениях, будь то веб-приложения или нет. Классификация очень больших наборов данных заслуживает осо-

бого внимания по трем (как минимум) причинам: 1) надлежащее представление набора данных в наборе для обучения; 2) вычислительная сложность этапа обучения; 3) производительность времени исполнения классификатора применительно к очень большому набору данных.

Независимо от конкретной предметной области приложения и той функциональности, которую поддерживает ваш классификатор, вы должны гарантировать, что данные для обучения являются репрезентативной выборкой данных, которые встретятся на этапе эксплуатации. Вряд ли классификатор будет работать с тем же успехом, который можно предположить на основании измерений, выполненных на стадии тестирования, если данные для обучения не являются выборкой рабочих данных с очень высокой степенью репрезентативности. Мы повторяемся, чтобы подчеркнуть эту мысль! Часто преждевременный восторг вскоре оборачивается разочарованием просто потому, что данное условие не встречалось ранее. Значит, вас в таком случае должно интересовать, как можно гарантировать репрезентативность данных для обучения.

Легче разобраться со случаем двоичной классификации, поскольку тут есть только два класса: сообщение электронной почты либо является спамом, либо нет, транзакция либо мошенническая, либо нет, и так далее. В этом случае, при условии, что имеется приемлемое число образцов для обучения из обоих классов, следовало бы сосредоточиться на покрытии значений атрибутов образцов, отобранных для обучения. Оценка может быть чисто эмпирической («Ага, годится. У нас достаточно значений; давайте выполним прогон в производственных условиях!»); она может быть строго научной (отбор образцов данных осуществляется на протяжении некоторого времени и проводится тестирование, позволяющее выяснить, взяты ли эти образцы из того же статистического распределения, что и данные для обучения); или оценка может быть чем-то средним между двумя указанными крайностями. На практике последний сценарий наиболее вероятен; мы могли бы назвать его полуэмпирическим подходом к обучению с учителем. Эмпирической стороной этого подхода является то, что наряду со способом оценки полноты набора для обучения вы делаете ряд обоснованных предположений, отражающих ваше понимание используемых в приложении данных и ваш опыт. Научной стороной этого подхода является то, что вам следовало бы собрать некоторую элементарную статистическую информацию о данных, например, получить минимальное и максимальное значения, средние значения, значения медианы, допустимые выбросы, процент пропущенных данных в значениях атрибутов и так далее. На основе этой информации можно отобрать образцы ранее не встречавшихся в вашем приложений данных и включить их в набор для обучения.

Случай многоклассовой классификации, в принципе, похож на случай двоичной классификации. Но вдобавок к упомянутым ранее руководя-

щим принципам здесь мы встречаемся с дополнительной сложностью. Наша новая проблема заключается в том, что надо выбрать такие образцы для обучения, чтобы все классы в наборе для обучения были представлены в равной степени. Отбор образцов из 1000 различных классов – гораздо более трудная для решения задача в сравнении с двоичной выборкой. Случай многомерной (много атрибутов) многоклассовой классификации создает дополнительные препятствия – следствие проклятия размерности (см. главу 4).

Если в вашей базе данных 100 миллионов записей, вам, естественно, хотелось бы воспользоваться всеми этими данными и задействовать содержащуюся в них информацию. На стадии проектирования классификатора следует проанализировать параметры масштабирования для этапов обучения и тестирования классификатора. Если вы удваиваете размер набора для обучения, спросите себя:

- Как возрастет время, затрачиваемое на обучение классификатора?
- Какова правильность классификатора на новом (более крупном) наборе?

Скорее всего, вам придется задействовать не только правильность, но и другие метрики качества, и рассмотреть еще несколько градаций размеров данных (в четыре, в восемь раз больше исходного набора и так далее), впрочем, идею вы поняли. Возможно, ваш классификатор работает отлично (он быстро обучился и обеспечивает высокую степень правильности) на небольшом наборе выборочных данных, но его производительность существенно падает, когда он обучается на гораздо большем наборе данных. Это имеет значение, поскольку показатель времени всегда важен для рынка, и «интеллектуальные» модули создаваемого вами приложения должны подчиняться тем же правилам производства, что и другие составляющие вашего программного обеспечения.

Этот же принцип верен для производительности, характеризующей выполнение классификатора на третьем этапе его жизненного цикла – этапе эксплуатации. Возможно, ваш классификатор быстро обучился и демонстрирует высокую степень правильности, но все это бесполезно, если он не обеспечивает хорошее масштабирование на этапе эксплуатации! На этапе тестирования классификатора следует измерить производительность и ее зависимость от размера данных. Допустим, вы используете классификатор с квадратичной зависимостью от размера данных: если размер данных удваивается, на обработку данных затрачивается в четыре раза больше времени. Далее предположим, что ваш интеллектуальный модуль будет использовать классификатор в фоновом режиме для обнаружения мошеннических транзакций. Если вы использовали 10 000 записей для тестирования и все записи были классифицированы за 10 минут, то 10 миллионов записей вы обработали бы примерно за 10 миллионов минут! Вряд ли у вас найдется столько времени, сле-

довательно, придется или выбрать другой классификатор, или повысить производительность того, что есть. Зачастую в промышленных системах ради скорости приходится жертвовать правильностью работы классификатора; если классификатор демонстрирует исключительную правильность и при этом крайне медленно работает, он, скорее всего, бесполезен!

Уделите внимание характерным особенностям вашей классификационной системы. Если вы используете систему на основе правил, то можете столкнуться с тем, что называют *проблемой эффективности*. Процесс обучения – накопление правил – может в результате привести к замедлению работы всей системы в целом на этапе эксплуатации. Есть способы, позволяющие избежать или, по крайней мере, смягчить проблему эффективности [Doorenbos, R. B.], но нужно в них разбираться и быть уверенным в том, что ваша реализация с ними совместима. Разумеется, падение производительности в этом случае не единственная проблема. Вам также потребуются способы, позволяющие управлять правилами и организовывать их; это проблема проектирования, решение которой сильно зависит от конкретной предметной области приложения. В общем случае, чем сложнее реализация классификатора, тем больше внимания следует уделить пониманию характеристик производительности (как скорости, так и качества) классификатора.

5.7. Заключение

Классификация – один из важнейших компонентов интеллектуальных приложений. Мы начали эту главу с того, что представили несколько случаев использования некоторых видов классификации. Были рассмотрены ссылочные схемы, которые относятся к разнообразным прикладным областям, от библиотечных каталогов до справочников медицинского страхования, и в результате мы установили, что классификация вездесуща и незаменима. Мы также представили три блока, из которых строятся системы классификации, – концепты, образцы и атрибуты. Эти три блока определяют онтологию – полное описание конкретной области специальных знаний. Если также доступна семантическая информация, то мы говорим о семантической онтологии. Классификацию можно рассматривать и как задачу назначения «лучшего» концепта данному образцу. Классификаторы отличаются друг от друга тем, как они отображают и измеряют такое оптимальное назначение. Тем не менее все они имеют сходный жизненный цикл, включающий три этапа – обучение, тестирование и эксплуатацию.

Вы узнали, что в общем случае все классификаторы делятся на две категории – двоичные и многоклассовые классификаторы, в зависимости от того, является ли решение, которое должен принять классификатор, выбором из двух или из нескольких вариантов, соответственно. Вы так-

же узнали, что с точки зрения базовой методики классификаторы являются или статистическими, или структурными. Обеспечив то, что нам видится литературным наибольшим общим знаменателем, мы продолжили обсуждение, рассмотрев высокоуровневое представление регрессионных алгоритмов, байесовских алгоритмов, алгоритмов на основе правил, функциональных алгоритмов, алгоритмов ближайших соседей, а также нейронных сетей.

Еще вы изучили два мощных алгоритма, применяемых для классификации текста. Первым был наивный байесовский алгоритм в применении к единственному строковому атрибуту. Вторым был процессор правил Drools, объектно-ориентированная реализация алгоритма Rete, которая позволяет объявлять и применять правила в целях классификации. Ваш клиент электронной почты, скорее всего, уже содержит в том или ином виде процессор правил; объявив, что сообщение электронной почты, содержащее конкретное слово в теме и присланное с домена **.foo.com*, следует считать спамом, вы, по сути, определяете правило. Таким образом, вы должны быть готовы к применению наших алгоритмов для решения многих других задач классификации неформатированного или частично структурированного текста.

Кроме того, мы познакомили вас с построением *вычислительных нейронных сетей* (computational neural networks) и представили элементарную, но надежную реализацию, пригодную для построения универсальных нейронных сетей. Мы сформулировали принципы проектирования, а также предоставили полученные в ходе наблюдений сведения относительно структуры данных и важности применения для обучения и тестирования таких наборов данных, которые являются репрезентативной выборкой производственных данных.

Хотя у классификации есть масса достоинств, мы обратили внимание на то, что прежде чем внедрять классификацию в свое приложение, важно также анализировать известные вопросы, связанные со степенью доверия и вычислительными требованиями классификации.

В заключение можно сказать следующее.

- Алгоритмы классификации важны для построения интеллектуального приложения, потому что они помогают нам задействовать (автоматически) и пополнять (систематически) наши знания о мире.
- Мы всегда проводим классификацию с учетом ссылочной структуры, которая могла бы быть и простой, как двоичный набор (классы истина/ложь), и обширной онтологией.
- На самом высоком уровне классификаторы можно разделить на статистические и структурные.
- Выбор классификатора зависит исключительно от данных и природы задачи классификации.

- Особое внимание требуется уделить степени доверия и стоимости классификации.
- Очень большие наборы данных, очень обширные онтологии, требования онлайн-режима или любые комбинации этих трех случаев могут затруднять классификацию.
- Каждый отдельный алгоритм из тех, что мы рассмотрели, будет хорошо делать свою работу. Но ни один взятый в отдельности классификатор не обеспечит способность безошибочного принятия решений. В сущности, если вам требуется безошибочность, вам не повезло!

В следующей главе мы собираемся рассмотреть несколько методик объединения классификаторов с целью улучшения результатов, обеспечиваемых каждым отдельным классификатором из тех, что мы уже рассмотрели.

5.8. Сделать

1. *Компромисс между специализацией и обобщением*. Любой алгоритм классификации, который вы в состоянии придумать, использует несколько переменных в качестве входных данных и вырабатывает несколько переменных в качестве выходных данных. Входные данные состоят из переменных двух видов. Первый вид – это переменные, связанные со значениями атрибутов наших образцов; второй – переменные, связанные с несколькими переменными модели, характерными для имеющегося классификатора. На этапе обучения мы оцениваем переменные модели, основываясь на входных и выходных переменных набора для обучения. Другими словами, мы калибруем эти произвольные параметры модели таким образом, что при заданных входных значениях набора для обучения выходные переменные получают требуемые значения.

   Понятно, что вы можете «сжульничать» и ввести столько параметров модели, сколько у вас точек данных, тем самым добиваясь очень высокой, если не абсолютной, правильности классификации в применении к набору для обучения. Это называется *переобучением* (overfitting) и делает ваш классификатор знатоком обучающего набора, но, скорее всего, плохим исполнителем для набора данных, достаточно отличающегося от набора для обучения. В общем случае, переобучение (мы могли бы также назвать это *специализацией*) – это плохо, и его следует избегать. И наоборот, у вас может оказаться слишком мало параметров модели, и вам не удастся собрать нужное *количество информации* (information content) из своего обучающего набора. Это называется *недооценкой* (underfitting). Используя меньшее число параметров, но все-таки в достаточном количестве для репрезентативного представления количества информации в обучающем

наборе, вы можете повысить уровень правильности для не встречавшихся ранее данных – точек, которые не были включены в набор для обучения. Способность к этому обычно называют *обобщением* (generalization).

Теперь понятно, что хороший классификатор следует ориентировать на достижение тонкого баланса между специализацией и обобщением. Поэкспериментируйте с наборами данных, которые мы предоставили в этой главе, и введите в свои данные новые объекты для тестирования. Постройте график ошибки вашего классификатора как функции от числа образцов в наборе для обучения. Постройте две кривые. Первая – график ошибки для образцов, принадлежащих набору для обучения, а вторая – график ошибки для образцов, не включенных в набор для обучения. Наблюдаете ли вы компромисс между специализацией и обобщением для данного классификатора? Можно было бы также ввести третье измерение, которое отображает сложность модели. Выражение сложности модели для правил и деревьев решений может быть простым по логике (например, число правил), но как бы вы выразили сложность модели в случае классификатора на основе теоремы Байеса, такого как наш класс `NaiveBayes`? А как насчет нейронной сети?

2. *Бритва Оккама* (Occam's razor) и *число атрибутов для обучения*. В том же духе, что и п. 1, мы можем утверждать, что чем больше обучающих атрибутов включено в нашу модель, тем лучше будут полученные нами результаты. У этого подхода есть две проблемы. Во-первых, с точки зрения практической реализации, мы, как правило, располагаем конечным объемом ресурсов и небольшим объемом времени для проведения классификации. Следовательно, наши схемы классификации должны быть такими, чтобы их легко было обслуживать, легко тестировать, и они должны довольно быстро вырабатывать результаты: вы ведь не хотите пять минут дожидаться результатов классификации, которые позволят узнать, является ваша электронная почта спамом или нет. Разумеется, есть случаи, которые требуют длительных вычислений, например обнаружение богатых нефтью месторождений или создание отчетов, помогающих вашим пользователям принять критичные (стратегические) бизнес-решения.

   Вторая проблема использования максимального возможного числа обучающих атрибутов связана с тем фактом, что «больше данных» не обязательно означает большее *количество информации*. Есть много метрик, позволяющих определить количество информации, но давайте откажемся от математического жаргона и подумаем об этом следующим образом. Как правило, мы заинтересованы в значениях некоторых переменных, релевантных для нашего приложения; это могли бы быть значения одной или нескольких акций на

торгах NASDAQ, соответствующая категория сообщений электронной почты, булевская переменная, которая отвечает, надо ли приобретать товар на сайте eBay, и так далее. Обычно мы исходим из того, что некая базовая модель описывает рассматриваемую задачу, а ее решение обеспечивают переменные, которые нам надо оценить. Предоставляя набор данных и классификатор, мы пытаемся как можно лучше аппроксимировать эту физическую модель. Если используются данные, не релевантные физической модели, или если мы «загружаем» классификатор избыточной информацией, то в результатом может стать искаженное представление этой физической модели. Ссылаясь на *количество информации*, мы имеем в виду данные, которые могут помочь нам улучшить представление базовой физической модели. Если добавленный нами обучающий атрибут не влияет на правильность классификатора, можно безошибочно утверждать, что этот новый обучающий атрибут не несет в себе существенного количества информации.

Конечно, к выбору обучающих атрибутов необходимо подходить с осторожностью, потому что разные классификаторы могут использовать большее или меньшее количество данных конкретных обучающих атрибутов. В общем случае можно использовать принцип бритвы Оккама: если два подхода обеспечивают получение одинаковых результатов, предпочтительным является простейший из них. Итак, рассмотрите пример с фильтрацией электронной почты из раздела 5.3 и добавьте больше атрибутов в набор для обучения классификатора. Можно было бы начать с того, что разбить единственный атрибут, который мы использовали в разделе 5.3.1, на два: один атрибут – для темы сообщения электронной почты, а другой – для основного сообщения. Наблюдаются ли существенные отличия в результатах классификации? В контексте количества информации, как бы вы интерпретировали применимость правил в классификации сообщений электронной почты?

3. *Универсальный класс RuleEngine*. В нашей реализации класса `RuleEngine` используется класс `Email` как аргумент метода `executeRules`. Этого вполне хватает для классификации единственного сообщения электронной почты, но недостаточно для общей реализации. В общем случае вы поместили бы все факты в рабочую память, прежде чем запускать выполнение правил. Модифицируйте существующий класс `RuleEngine`, так чтобы его можно было использовать в более общих условиях.

   Кроме того, создайте вариант применения для процессора правил, имеющего дело с более сложными правилами, условиями и действиями. Класс `ClassificationResult` послужит вам проводником для пользовательской настройки условий и действий. Обратите внимание, что благодаря объектно-ориентированной природе процессора

правил Drools вы можете построить усложненные правила, включив в них условия и довольно изощренные действия. Хорошая практика – использовать вспомогательные классы, такие как `ClassificationResult`, помещая в них весь свой Java-код и избегая добавления программного кода в сам файл с правилами системы Drools.

Как бы вы продолжили создание универсальной системы классификации на основе правил? Представьте систему, которая может классифицировать произвольный текст по нескольким классам; это могло бы быть сообщение электронной почты, документ текстового процессора Word, документ в формате PDF и так далее. Какие правила вам потребуются? Какие условия? И как бы вы выразили их в программном коде?

4. *Важность нормализации данных и влияние топологии нейронной сети.* Заметим, что все значения данных, которые мы передаем в качестве входных в нейронную сеть, являются нормализованными значениями. Сумма транзакции нормализована на основе минимального и максимального значений, определенных для допустимых транзакций, так что это значение всегда принадлежит интервалу от 0 до 1. Еще мы используем класс `JaccardCoefficient`, чтобы получить такой же результат для описания транзакции. С местами проведения транзакций мы поступаем несколько хитрее. Мы нормализуем местонахождение центроида пользователя и место проведения транзакции, а затем вычисляем расстояние между этими двумя точками. Это расстояние является одним из трех входных значений для нейронной сети `TransactionNN`. Зачем мы это делаем? Важно ли это? Поэкспериментируйте, используя этот же программный код, но с переключением на ненормализованный вход. Используйте идентичные данные для обучения и тестирования, чтобы можно было сравнить влияние, которое оказывают сами алгоритмы. Очевидно, что вам следует изменять за один раз только что-то одно, чтобы можно было установить связь между следствием и причиной изменений.

   Второй тип экспериментов, который может оказаться довольно поучительным, – влияние топологии нейронной сети на результаты классификатора. Лучшая топология нейронной сети зависит от природы входных данных и самой вашей задачи. Вариант с обнаружением мошенничества и базовая реализация класса `TransactionNN` обеспечивают основу, помогающую проанализировать эту зависимость. Реализуйте собственную топологию нейронной сети и сравните полученные результаты работы классификаторов. Например, можно было бы попробовать явно подавать координаты x и y на вход сети вместо одного только расстояния от места проведения каждой транзакции до центроида пользователя. Вы могли бы также предоставить описание в нескольких узлах. Одним из способов, позволяющих это сделать, может стать разметка (tokenizing) описания и ис-

пользование сходства каждой лексемы с первыми пятью лексемами описания. В результате были бы получены пять входных узлов, связанных с описанием, или, в более общем случае, при использовании сходства каждой лексемы с первыми N лексемами описания, N входных узлов, связанных с описанием. Как изменяются ваши результаты по мере возрастания N? Чем отличаются результаты, полученные с помощью этих нейронных сетей, от результатов, которые обеспечивает базовая реализация класса `TransactionNN`?

5. *Обучение без учителя: обучение по Хеббу* (Hebbian learning) и *самоорганизующиеся карты* (self-organizing maps, SOM). В этой главе мы рассмотрели только методики обучения с учителем, которые очень распространены и полезны. Но методики обучения без учителя также полезны и заслуживают вашего внимания. В случае обучения с учителем у вас всегда есть ощущение, что вы ввели ответ с «черного хода», – математики называют это *интерполяцией* (interpolation), термин менее броский и более достойный. В любом случае, суть в том, что мы сообщаем классификатору необходимые ему знания, а он пытается усвоить их, для чего изменяет свои параметры, либо вычисляя априорную и условную вероятности (в случае байесовских методов), либо настраивая различные веса (в случае нейронных сетей), либо беззастенчиво все «списывая» (в случае систем на основе правил). У системы с обучением без учителя есть поразительное свойство: она может «запомнить» то, что «видела», не получая обратного отклика от человека.

   Дональд Хебб в своей книге «Организация поведения: нейропсихологическая теория» («The Organization of Behavior», 1949) представил простую модель, которая хорошо иллюстрирует способность учиться без учителя. Если вы не сообщаете классификатору, что является правильным, то как он работает? Рассмотрим нейронную сеть, узлы которой полностью связаны с помощью симметричных двунаправленных связей (синапсов); «симметричные» в данном случае означает, что вес связи, ведущей от узла i к узлу j, равен весу связи, ведущей от узла j к узлу i. Какого рода правила активизации и обучения могут помочь нам построить нейронную сеть без учителя?

6. *Подсчет стоимости ошибок классификации.* В реальных условиях системы классификации часто служат для поддержки систем принятия решений, то есть ошибки классификации могут вести к неверным решениям. В некоторых случаях принятие неверных решений, хоть и нежелательно, может быть сравнительно безвредным. Но в других случаях это может составить разницу между жизнью и смертью; подумайте о враче, который не сможет диагностировать рак, или об опасности для астронавта в открытом космосе, если они полагаются на результат работы вашего классификатора. Итак, оценка систем классификации должна анализировать как степень

доверия, так и связанную с ней стоимость классификации. Для двоичной классификации можно назначить *функцию стоимости*, которая является функцией коэффициентов FP и FN. Как бы вы обобщили эту идею для многоклассовой классификации?

При многоклассовой классификации, если есть N классов и вы совершаете ошибку, имеется $N - 1$ вероятностей. Следовательно, необходим способ, позволяющий назначить стоимость для $N \times (N - 1)$ случаев. Естественно, наиболее подходящий инструмент для достижения этой цели – матрица. В случае многоклассовой классификации матрица неточностей – это матрица $N \times N$, и мы можем определить *матрицу стоимости* (cost matrix), которая также имеет размерность $N \times N$, но значениями ее диагонали являются значения 0 (не следует наказывать классификатор за правильные ответы). Проработайте детали и оцените, например, классификатор `NaiveBayes` с помощью различных матриц стоимости.

5.9. Ссылки

Схемы классификации

- The Dewey Decimal Classification (DDC) system. *http://www.oclc.org/dewey/*
- International Classification of Diseases (ICD). World Health Organization (WHO). *http://www.who.int/classifications/icd/en/*
- The Library of Congress: Cataloging Distribution Service. *http://www.loc.gov/cds/*
- Myhre, A. P., and M. L. Richardson «A Web-based Tutorial for Teaching the Schatzker Classification for Tibial Plateau Fractures». *http://uwmsk.org/schatzker/*
- «Occupational Injury and Illness Classification Manual», Bureau of Labor Statistics, U.S. Department of Labor. *http://www.bls.gov/iif/oshoiics.htm*

Книги и статьи

- Antoniou, G., and F. van Harmelen «A Semantic Web Primer», The MIT Press, 2004.
- Doorenbos, R. B. «Production Matching for Large Learning Systems», Ph. D. Thesis, Carnegie Mellon University, 1995.
- Dunham, M. H. «Data Mining: Introductory and Advanced Topics», Prentice Hall, Pearson Education Inc. 2003.

- Fawcett, T. «ROC Graphs: Notes and practical considerations for researchers», 2004. *http://home.comcast.net/~tom.fawcett/public_html/papers/ROC101.pdf*.
- Friedman-Hill, E. «Jess in Action», Manning Publications, 2003.
- Gasevic, D., D. Djuric, V. Devedzic «Model Driven Architecture and Ontology Development», Springer, 2006.
- Gómez-Pérez, A., M. Fernández-López, and O. Corcho «Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web», Springer, 2004.
- Hastie, T., R. Tibshirani, and J. Friedman «The Elements of Statistical Learning: Data Mining, Inference, and Prediction», Springer, 2001.
- Hill, T. (1996) «A note on distributions of true versus fabricated data», Perceptual and Motor Skills. Vol 83, pp. 776–778. *http://www.math.gatech.edu/~hill/publications/cv.dir/truvsfab.pdf*.
- Holmström, L., P. Koistinen, J. Laaksonen, and E. Oja «Neural and statistical classifiers – taxonomy and two case studies», IEEE Transactions on Neural Networks, Vol 8 (1), pp. 5–17, 1997.
- Maier, K. D., C. Beckstein, R. Blickhan, W. Erhard, and D. Fey «A multilayer-perceptron neural network hardware based on 3D massively parallel optoelectronic circuits», Proceedings of the 6th International Conference on Parallel Interconnects, pp. 73–80, 1999.
- MacKay, D. J. C. «Information Theory, Inference, & Learning Algorithms», Cambridge University Press, 2003.
- Neapolitan, R. E. «Learning Bayesian Networks», Prentice Hall, 2003.
- Papoulis, A., and S. U. Pillai «Probability, Random Variables, and Stochastic Processes», Fourth Edition, McGraw-Hill, 2002.
- Rish, I. «An empirical study of the naïve Bayes classifier», IBM Research Report, RC22230 (W0111-014). *http://www.cc.gatech.edu/~isbell/classes/reading/papers/Rish.pdf*.
- Russell, S., and P. Norvig «Artificial Intelligence: A Modern Approach»[1] (Second Edition), Prentice Hall, 2002.
- Staab, S., and R. Studer «Handbook on Ontologies», Springer, 2004.
- Witten, I. and Frank, E. «Data Mining: Practical Machine Learning Tools and Techniques», 2nd Edition, Morgan Kaufmann, 2005.

[1] Рассел С., Норвиг П. «Искуственный интеллект: Современный подход», 2-е изд., Вильямс, 2007.

6

Объединение классификаторов

- Основы оценки классификаторов
- Сравнение классификаторов и понимание сложных наборов данных
- Практика применения самонастраиваемого объединения
- Основы применения метода улучшения

Античный греческий философ Эпиктет сказал: «Не на одном якоре стоит корабль, не на одной только надежде покоится жизнь»[1]. Нам также не следует полагаться на единственный классификатор. В одиночку ни один классификатор не в состоянии наделить нас способностью безошибочно принимать решения. В самом деле, есть масса примеров, демонстрирующих огромный потенциал объединения классификаторов, и настоящая глава познакомит вас с этой интересной темой. [Bell R., Y. Koren, and C. Volinsky] недавно с большим успехом применили аналогичные идеи в контексте систем выработки рекомендаций (см. главу 3).

Основная идея, ради которой объединяют классификаторы, – добиться лучших результатов классификации ценой более сложных и дорогостоящих (например, требующих более длительного времени или дополнительных ресурсов) вычислений. Методики объединения классификаторов делятся на две главные категории – *слияние решений* (classifier fusion) и *подбор классификаторов* (classifier selection). В методиках из

[1] «Пифагоровы законы и нравственные правила. Афоризмы Эпиктета», М: Олма-Пресс, СПб: Нева, 2000.

категории «слияние решений» все классификаторы вносят свой вклад в данную классификацию, следовательно, каждый классификатор должен охватывать всю область возможных точек данных. В методиках из категории «подбор классификаторов» каждый классификатор отвечает за конкретную область точек данных и, предположительно, хорошо работает только в зоне своего влияния.

Такое различие между типами объединения классификаторов полезно в качестве «ориентира на местности», но не является абсолютным. Мы увидим (см. п. 5 раздела «Сделать»), что понятия слияния и подбора классификаторов не всегда легко разграничить.

Если вам захочется почитать что-либо о теории объединения классификаторов, обратитесь к отчету [Dietterich, T. G.]. Причины объединения классификаторов он делит на три большие группы.

Причины из первой группы обусловлены статистическим анализом; главным аргументом здесь является то, что объединение классификаторов, возможно, не превзойдет по результатам какой-то один, самый лучший классификатор в ансамбле, но значительно снизит риск применения классификатора, который может оказаться нерелевантным по отношению к ранее не встречавшимся данным.

Причины объединения из второй группы опираются на вычислительный анализ, указывающий на то, что классификаторы нередко оказываются восприимчивы к обучению. Так, [Dietterich, T. G.] утверждает, что объединение классификаторов может обеспечить лучшие результаты за счет выравнивания уровня восприимчивости каждого классификатора в ансамбле во время обучения.

Причины из третьей группы связаны с таким стимулирующим фактором, как представление предметных сущностей. Для пояснения предположим, что нам потребовалось классифицировать какие-то краткие текстовые описания на французском языке (например, *porte, fenêtre* и *guerre*). Но вот незадача: у нас нет классификатора, «понимающего» французский, а есть только тот, что «понимает» английский и испанский. Мотивация с точки зрения представления предметных сущностей сводится к тому, что «объединенными усилиями» английский и испанский языки обеспечат лучшее представление французского языка, хотя ни один из этих языков, взятый в отдельности, обычно не в состоянии достойно справиться с этой задачей. Слово *guerre* в переводе на английский язык – war (война), в то время как в испанском языке это будет слово *guerra*; и наоборот, можно найти французские слова, по написанию более близкие к английскому аналогу, чем к испанскому. В ходе обсуждения у вас будет возможность распознать мотивационные факторы из указанных групп.

Начнем с представления конкретного случая оценки кредитоспособности лиц, желающих получить ипотечный кредит. Эта тема актуальна, поскольку связана с экономическим спадом, который мы переживаем

сегодня (в 2008 году). Будем рассматривать случай оценки кредитоспособности лиц, желающих получить ипотечный кредит, а не риска предоставления кредита. Нужные нам данные будут искусственными (поскольку сведения об оценке риска предоставления кредита и соответствующие модели защищены правом собственности), но реалистичными. Естественно, первый вопрос, с которым необходимо разобраться, – применение в рассматриваемом случае единственного классификатора. Итак, создаем три классификатора для классификации лиц, желающих получить кредит, по пяти категориям их кредитоспособности. А именно, предоставляем классификатор на основе наивного байесовского алгоритма, классификатор на базе нейронной сети, а также недавно введенный классификатор на основе дерева решений.

Следующий шаг – исследование на предмет того, является ли какой-либо из этих классификаторов лучшим. Для этого покажем, как сравнить два и более классификаторов с помощью четырех статистических методик. Чтобы сравнить два классификатора, воспользуемся тестом Макнемара (McNemar) и тестом *на разность пропорций* (difference of proportions). Для сравнения большего числа классификаторов будем использовать Q-тест Кохрана (Cochran's Q test) и F-тест. Не переживайте по поводу их несколько пугающих названий: мы свели практическое применение этих тестов к нескольким обращениям к методам класса.

Мы познакомим вас с двумя категориями методик объединения классификаторов. Во-первых, вы увидите, как реализовать объединения из категории *bagging* (от англ. bootstrap aggregating – самонастраиваемое объединение). Этот метод будет использован для того, чтобы улучшить показатели правильности классификатора на основе дерева решений и проанализировать, как меняются результаты классификации с увеличением числа объединяемых классификаторов. Второй подход к объединению классификаторов, который мы изучим, называется boosting (улучшение). Мы представим алгоритм с довольно загадочным названием *arc-x4* и рассмотрим его предшественника – алгоритм AdaBoost. Естественно, эти методики объединения будут использованы в контексте представленной нами задачи оценки кредитоспособности.

6.1. Кредитоспособность: анализ примера объединения классификаторов

Начнем с описания конкретного случая оценки кредитоспособности пользователя в приложении, обслуживающем ипотечное кредитование. Мы анализируем применение трех различных классификаторов, разработанных специально для решения этой задачи. В частности, рассматриваются: классификатор на основе наивного байесовского алгоритма, классификатор на основе нейронной сети и недавно введенный классификатор на основе дерева решений. Итак, что такое кредитоспособность и почему важно ее оценивать? Вот некоторые исходные

предпосылки, которые создадут определенный контекст и подчеркнут (возможно, неожиданно) влияние, которое хорошие интеллектуальные приложения могут оказать на глобальную экономику!

В 2007 году в Соединенных Штатах произошел эпохального масштаба обвал ипотечного рынка с глобальными последствиями. Населению предоставили возможность – если не сказать, что его к этому подталкивали, – покупать дома, которые люди не могли себе позволить, с помощью кредитных лимитов, которых они не заслуживали. Когда цены на недвижимость начали падать и поток денежных средств на рынке сократился, не обладающие предполагаемой покупательной способностью заемщики перестали выплачивать полученные кредиты. Сколько всего случаев отчуждения заложенного имущества и продаж с молотка породила эта ситуация, точно неизвестно: на момент написания этой книги кризис продолжается.

Во избежание дальнейших экономических бедствий и социальной нестабильности правительственные органы США вмешались в ситуацию и попытались найти способы, позволяющие оставить многим покупателям их дома. По разным оценкам, насчитывается около 2 миллионов домовладельцев с регулируемой ипотекой, многие из которых изначально имели низкие – в рекламных целях – ипотечные ставки. Не получив от кого-либо прямой финансовой поддержки, эти люди неизбежно столкнутся с повышением ставки по кредиту независимо от величины ставки по централизованным кредитным средствам Федеральной резервной системы США[1]. Отчуждение заложенного имущества и продажа с молотка неминуемо ведут к общему падению цен на недвижимость и снижению благосостояния и потребительской активности большинства владельцев недвижимости.

Эта проблема может иметь глобальные последствия по двум причинам. Эксперты полагают, что такая же финансовая нестабильность может возникнуть и на ипотечных рынках Европы, и тогда государствам, являющимся крупными экспортерами, особенно развивающимся странам, таким как Китай, чьи товары заполняют полки американских магазинов, придется искать новых покупателей, поскольку оставшиеся без денег американцы больше не смогут покупать их продукцию.

Вы удивлены: какое отношение все это имеет к программным интеллектуальным приложениям? Как выясняется, довольно близкое! В целом возникшая финансовая проблема усложняется и может сохраняться довольно долго. Тем не менее, пока ясно одно: если исключить мошеннические и фиктивные схемы ипотечного финансирования, влияющие

[1] Ставка по централизованным кредитным средствам Федеральной резервной системы США – это процентная ставка, по которой одни депозитные учреждения предоставляют однодневный кредит из Федерального резерва другим депозитным учреждениям.

на входные данные, невозможность оценить кредитоспособность (или риск кредитования) отдельной транзакции сыграла роль катализатора, ускорявшего распространение последствий кризиса, и способствовала вовлечению в этот кризис все новых социальных структур.

В этой главе мы изучим задачу оценки кредитоспособности отдельного лица по ряду критериев. Другими словами, рассмотрим значения ряда атрибутов, описывающих финансовое положение, социальный статус и кредитную историю конкретного человека, позволяя отнести его кредитоспособность к одной из 5 категорий: превосходная (excellent), очень хорошая (very good), хорошая (good), плохая (bad) и опасная (dangerous).

Будем считать, что кредитоспособность является функцией 11 произвольно выбранных атрибутов. Мы взяли достаточно атрибутов, чтобы рассмотрение было интересным и реалистичным, но вы не должны воспринимать наш вариант как самую сложную модель оценки кредитоспособности. Поучительно было бы проанализировать, какие из этих атрибутов можно исключить или какие еще атрибуты можно добавить. В п. 1 раздела «Сделать» этой главы представлена соответствующая мотивация и даны дополнительные указания по немаловажному вопросу выбора атрибутов.

6.1.1. Краткое описание данных

Проанализируем упомянутые 11 атрибутов один за другим в алфавитном порядке. Первый атрибут – это *календарный возраст* (Age) человека (в противоположность его умственному развитию, о котором, как правило, нельзя судить по официальным документам). Возраст, безусловно, является непрерывной величиной, поскольку время – величина непрерывная. Следовательно, наше первое исходное условие заключается в том, что возраст измеряется числом прожитых лет. Мы будем использовать целые значения от 0 до 100; если вы посмотрите на реализацию, то поймете, что сами по себе точные границы возрастного диапазона не очень важны. Мы делим интервал возрастов от 18 до 100 лет на 10 диапазонов, как указано в табл. 6.1.

Таблица 6.1. 10 диапазонов для значений атрибута Age (возраст). Это разбиение не является ни очень точным, ни очень грубым: возрастные диапазоны вводятся для того, чтобы адекватно отобразить различные этапы финансовой динамики в жизни частного лица и сопряженные с этим риски

Идентификатор атрибута	От	До
1	18	25
2	26	30
3	31	35
4	36	40

Таблица 6.1 (продолжение)

Идентификатор атрибута	От	До
5	41	50
6	51	60
7	61	70
8	71	75
9	76	80
10	81	100

Второй учитываемый нами атрибут определяет, проходило ли данное лицо процедуру *банкротства* (`Bankruptcy`). Значимость банкротства варьируется от страны к стране: объявление банкротства в Соединенных Штатах – это совсем не то же самое, что объявление банкротства в Германии. В первом случае банкротство свидетельствует о наличии предпринимательской жилки, чем можно гордиться. Во втором случае банкротство может стать финансовым и социальным клеймом, от которого не избавиться до конца своих дней! Помимо этих есть еще много разных видов банкротства. Оценивая кредитоспособность, важно понимать причину банкротства: было ли оно вызвано обстоятельствами, связанными с состоянием здоровья, или личным расточительством. В нашем примере атрибут «банкротство» получает булевское значение.

Третий атрибут – это *владение автомобилем* (`CarOwnership`). И хотя одно дело – иметь ламборгини и совсем другое – автомобиль фирмы KIA, мы будем учитывать владение автомобилем как булевский атрибут. Поскольку нас интересует кредитоспособность, сам факт, что у человека есть автомобиль, очень показателен и является очень простым отличительным признаком. Как по-вашему, когда было бы полезно назначить этому атрибуту целое значение, скажем, указав конкретные ценовые диапазоны для автомобилей? Рассмотрите, например, случай, когда требуется более точно классифицировать кредитоспособность, определив для нее 20, а не 5 категорий. Кроме того, следовало бы учитывать исключения – тех, кто не владеет автомобилем, потому что не в состоянии им управлять (физически), либо не испытывает потребности или желания иметь автомобиль, даже если может его себе позволить.

Четвертый атрибут – *кредитный балл* (CreditScore). Есть три основные компании, которые собирают информацию, связанную с использованием кредитных карт, ипотечных займов, кредитных линий под залог жилой недвижимости и так далее. Эти компании назначают каждому лицу кредитный балл из некоторого интервала, который варьируется в зависимости от компании, но, как правило, является числом от 0 до 800. Мы будем использовать для представления кредитного балла 8 различных «вилок», как указано в табл. 6.2.

Таблица 6.2. Диапазон значений для атрибута CreditScore (кредитный балл)

Идентификатор атрибута	От	До
1	0	500
2	501	550
3	551	600
4	601	650
5	651	700
6	701	750
7	751	800
8	801	–

Пятый атрибут – это индикатор того, имеет ли человек *судимость* (`CriminalRecord`). Мелкое преступление или обычная жалоба на человека в корне отличаются от вооруженного ограбления. Однако использование для этого атрибута булевского значения не имеет такого ограничительного характера, как это может показаться на первый взгляд: значение данного атрибута – это просто соответствующий уровень порога «криминальности». Уголовные преступления или какой-то другой подробный список преступлений послужат основанием для присваивания этой булевской переменной значения «истина», тогда как более «мягкие» проступки оставляют для атрибута судимости значение, принятое по умолчанию («ложь»).

Шестой атрибут – *доля в процентах авансового платежа от суммы займа* (`DownPayment`), которую желающий получить кредит согласен выплатить к моменту погашения кредита. В табл. 6.3 показаны четыре диапазона для значения авансового платежа. Если желающий получить кредит способен выплатить авансом свыше 25% от суммы займа, не имеет значения, сколько именно он может выплатить. Интерес представляет область дифференциации от 0% до 25%.

Таблица 6.3. Диапазон значений для атрибута DownPayment (авансовый платеж, исчисляется как доля в процентах от общей суммы займа)

Идентификатор атрибута	От	До
1	0	5
2	6	10
3	11	20
4	21	–

Седьмой атрибут – *сумма дохода* (`Income`). По аналогии с атрибутом `DownPayment`, когда мы учитываем суммы доходов, нас не волнует, имеет ли кто-то доход в 5 или 100 млн долл. Нас больше интересует расслоение доходов в нижней части спектра. Кроме того, разброс может быть таким, что сумму дохода представляет любое число, от 0 до нескольких миллиардов. Для дифференциации лиц, желающих получить кредит, мы будем использовать укрупненные показатели доходов. В табл. 6.4 перечислены 10 возможных значений этого атрибута и соответствующие диапазоны; сумма дохода исчисляется в тысячах долларов.

Таблица 6.4. Диапазон значений для атрибута Income (сумма дохода), в тыс. долл.

Идентификационный код (ID) атрибута	От	До
1	0	25
2	26	35
3	36	45
4	46	60
5	61	80
6	81	100
7	101	125
8	126	150
9	151	200
10	201	–

Восьмой атрибут определяет *род трудовой деятельности* (`JobClass`) человека. В целях рассматриваемого примера мы предлагаем использовать пять «классов деятельности», но можно применить и более подробное деление. В табл. 6.5 показаны пять возможных значений этого атрибута.

Таблица 6.5. Диапазон значений для атрибута JobClass (род трудовой деятельности)

Идентификатор атрибута	Класс деятельности
1	C-level executive (руководство высшего звена)
2	Professional (специалист)
3	Employee (наемный работник)
4	Business owner (владелец предприятия)
5	Contractor (работник по контракту)

Девятый атрибут связан с *владением мотоциклом* (`MotorcycleOwnership`); принимает булевские значения. В некоторых странах мотоцикл в собственности – это роскошь, в других это главное (если не единственное) транспортное средство. Этот атрибут не должен сильно влиять на результаты, поскольку, генерируя данные, мы присваиваем ему значение исходя из равной вероятности владения мотоциклом представителями всех категорий кредитоспособности (кроме опасной). Позже мы рассмотрим определение категорий претендентов на получение займа в зависимости от значений приведенных атрибутов и подробнее обсудим это исходное допущение.

Десятый атрибут связан с *владением недвижимостью* (`PropertyOwnership`), то есть землей, и он также является булевским. Естественным развитием этого атрибута было бы отображение стоимости земли, но в примере мы будем трактовать его как булевскую переменную.

Одиннадцатый атрибут относится к *сумме финансовых активов, связанных с пенсионными выплатами* (`RetirementAccounts`). Разумеется, это непрерывная величина, поэтому мы подвергаем ее дискретизации с помощью конкретных диапазонов. В табл. 6.6 показаны 8 значений этого атрибута и назначенные им диапазоны; выплаты исчисляются в тысячах долларов.

Напоминаем, что ни в числе 11, ни в выборе этих конкретных атрибутов нет ничего магического. В реальных условиях вам придется выбирать из множества атрибутов, их выбор является частью постановки и решения задачи классификации (см. п. 1 раздела «Сделать»). Важнее всего, чтобы вы понимали природу ваших данных. Можно выучить все известные человечеству алгоритмы, но если вы не понимаете природу данных, обеспечить удовлетворительное решение вашей задачи будет нелегко.

Таблица 6.6. Диапазон значений для атрибута RetirementAccounts (пенсионные выплаты)

Идентификатор атрибута	**От**	**До**
1	0	25
2	26	75
3	76	150
4	151	300
5	301	500
6	501	1000
7	1001	2000
8	2001	–

6.1.2. Создание искусственных данных для реальных задач

Проанализируем ряд решений, которые необходимо принять для определения наших данных, чтобы вы получили представление о том, что мы понимаем под природой данных. Как уже говорилось, для представления каждого пользователя будут использованы значения 11 атрибутов. Чтобы определить категорию кредитоспособности пользователя, каждой категории назначается набор допустимых значений атрибутов. Иначе говоря, мы «выделяем» пять областей в 11-мерном пространстве и помечаем каждую из этих областей как одну из пяти категорий. Если требуется создать пользователя конкретной категории, мы случайным образом выбираем подходящие кортежи – для каждого атрибута выбирается значение из области соответствующей категории.

Получить допустимые значения атрибутов для каждой категории кредитоспособности можно с помощью Java-классов `ExcellentUserType` («превосходная»), `VeryGoodUserType` («очень хорошая»), `GoodUserType` («хорошая»), `BadUserType` («плохая»), `DangerousUserType` («опасная»), каждый из которых является расширением абстрактного класса `UserType`, содержащего общую функциональность. В листинге 6.1 приведен программный код класса `GoodUserType`, позволяющего создать пользователя наиболее часто встречающейся категории кредитоспособности. Как видим, согласно этому определению пользователь с хорошей кредитоспособностью может иметь любой возраст, не должен быть объявлен банкротом, должен владеть автомобилем и так далее.

Листинг 6.1. Класс GoodUserType: определение категории кредитоспособности «хорошая» с помощью значений атрибутов

```
public class GoodUserType extends UserType {
  {
    setAge(new int[] { 2, 3, 4, 5, 6, 7, 8 });
    setBancruptcy(new int[] { 0 });
    setCarOwnership(new int[] { 1 });
    setCreditScore(new int[] { 3, 4, 5, 6 });
    setCriminalRecord(new int[] { 0 });
    setDownPayment(new int[] { 2, 3 });
    setIncome(new int[] { 5, 6, 7, 8 });
    setJobClass(new int[] {2, 3, 4, 5});
    setMotorcycleOwnership(new int[] { 0, 1});
    setPropertyOwnership(new int[] { 0, 1 });
    setRetirementAccounts(new int[] { 1, 2, 3, 4 });
  }

  @Override
  public String getUserType() {
    return UserType.GOOD;
  }
}
```

Взгляните на другие классы семейства UserType. Что вы видите? Как и следовало ожидать, соседние классы частично перекрывают друг друга. Эти размытые границы делают данные более реалистичными. Возможно, вас интересует, не может ли случиться так, что атрибуты пользователя из категории кредитоспособности «плохая» будут иметь значения, которые полностью принадлежат области, определяющей категорию кредитоспособности «хорошая»? В реальных данных такое было бы не только возможно, но и наиболее вероятно. Именно эту проблему мы и пытаемся решить! Мы хотим исходя из некоторого набора значений атрибутов автоматически оценивать кредитоспособность пользователя. Если бы данные реального мира можно было «вписать» в некоторые заранее известные модели, «обучение» было бы не слишком полезно; всю интеллектуальность можно было бы загрузить в априорную модель и привести эту модель в действие. Чтобы избавиться от такой особенности данных реального мира, мы вводим понятие *уровней шума* (noise levels).

Уровни шума определяют вероятность того, что пользователь, значения атрибутов которого принадлежат области категории *X*, на самом деле будет принадлежать категории *Y*. Уровни шума позволяют в произвольной степени смешивать определения классов. В листинге 6.2 показаны важнейшие шаги, реализованные в классе DataGenerator.

Листинг 6.2. Класс DataGenerator: создание наборов пользователей для представления всех категорий кредитоспособности

```
public List<User> generateUsers(List<UserType> userTypes) {

   List<User> allUsers = new ArrayList<User>();

   for(UserType userType : userTypes) {
       allUsers.addAll(generateUsers(userType, userType.getNUsers() ));
   }
   return allUsers;
}

public List<User> generateUsers(UserType userType, int n) {

   List<User> users = new ArrayList<User>();

   userTypeDistributions.put(userType, n);

   for(int i = 0; i < n; i++) {  ❶
      User u = generateUser(userType);
      users.add(u);
   }
   return users;
}

public User generateUser(UserType userType) {
```

```
    User user = new User();

    long userId = generateNextUniqueUserId();

    String username;

    if (isNoiseOn) {  ❷
       username = userType.getNoisyType();
    } else {
       username = userType.getUserType();
    }

    username = username + String.valueOf(userId);

    user.setUsername(username);
    user.setAge(userType.pickAge());
    user.setCarOwnership(userType.pickCarOwnership());
    user.setCreditScore(userType.pickCreditScore());
    user.setIncome(userType.pickIncome());
    user.setJobClass(userType.pickJobClass());
    user.setDownPayment(userType.pickDownPayment());
    user.setBicycleOwnership(userType.pickMotorcycleOwnership());
    user.setPropertyOwnership(userType.pickPropertyOwnership());
    user.setCriminalRecord(userType.pickCriminalRecord());
    user.setBankruptcy(userType.pickBancruptcy());
    user.setRetirementAccount(userType.pickRetirementAccounts());

    return user;
  }
```

❶ Создаем несколько пользователей каждого типа согласно заданному распределению пользователей по типам.

❷ Если присутствует шум, смешиваем типы пользователей, как это определено в методе `getNoisyType`. В противном случае, используем только заданное распределение по типам пользователей.

Поскольку данные для нашего примера являются искусственными, соотношение числа пользователей в различных категориях кредитоспособности находится под полным контролем. Принятое по умолчанию распределение пользователей по категориям показано в табл. 6.7.

Таблица 6.7. Распределение пользователей по категориям кредитоспособности, в процентах от общего числа пользователей

Категория кредитоспособности	**Число пользователей (%)**
Превосходная	5
Очень хорошая	15
Хорошая	50

Категория кредитоспособности	Число пользователей (%)
Плохая	25
Опасная	5

Разумеется, вы можете изменить долю пользователей в каждой категории. Это можно сделать, изменив реализацию метода `createUserTypes` в классе `UseCaseData`. Важно отметить, что данное распределение вряд ли является однородным. Большинство пользователей принадлежит категории кредитоспособности «хорошая», а вероятность того, что пользователи, не принадлежащие этой категории, скорее всего будут иметь худшую кредитоспособность, можно оценить как 3 к 2. Две крайних категории кредитоспособности с равной вероятностью соответствуют этому распределению.

Вы можете создать свои искусственные данные, выполнив команды из листинга 6.3.

Листинг 6.3. Создание искусственных данных для случая оценки кредитоспособности

```
UseCaseData useCaseData = new UseCaseData(40000,20000);

UserType.addNoiseLevel("EX",
➥  new Double[] {1.0d, 5.0d, 8.0d, 10.0d});
UserType.addNoiseLevel("VG",
➥  new Double[] {1.0d, 2.5d, 6.0d, 10.0d});
UserType.addNoiseLevel("GD",
➥  new Double[] {1.0d, 3.0d, 4.0d, 8.0d});
UserType.addNoiseLevel("BD",
➥  new Double[] {1.0d, 3.0d, 7.5d, 10.0d});
UserType.addNoiseLevel("DN",
➥  new Double[] {1.0d, 6.0d, 10.0d, 14.0d});

useCaseData.create(false);
```

Определение нового набора данных

Определение уровней шума для каждого класса

Задание для аргумента значения true обеспечит переопределение файлов

Если вы хотите использовать уровни шума, принятые по умолчанию, можете пропустить обращения к методу `UserType.addNoiseLevel`. Будьте осторожнее с обращением к последнему методу. Если присвоить его булевскому аргументу значение `true`, существующий набор данных будет уничтожен и результаты выполнения программного кода из предоставленных листингов будут отличаться от тех, которые вы видите в этой книге. Оригинальные файлы всегда можно восстановить, распаковав их из файла *clean_40k_20k.7z*, который находится в дистрибутивном каталоге *data/ch06/samples/clean*.

Прежде чем двигаться дальше, давайте подробнее рассмотрим определение уровней шума. В листинге 6.2 важнейшим шагом было обраще-

ние к методу getNoisyType. В листинге 6.4 приведен соответствующий программный код для получения пользователей с типом кредитоспособности «превосходная», помогающий интерпретировать семантику обращений к методу UserType.addNoiseLevel.

Листинг 6.4. Метод UserType.getNoisyType: добавление шума к типу пользователя с «превосходной» кредитоспособностью

```
public String getNoisyType() {

    double gaussian = rnd.nextGaussian();

    String noisyType=null;

    String userType= getUserType();

    Double[] nLevels = noiseLevels.get(userType);

    if (getUserType().equals(EXCELLENT)) {

        if (gaussian <= nLevels[0]) {

            noisyType = EXCELLENT;

        } else if (gaussian >  nLevels[0] &&
           gaussian <= nLevels[1]) {

            noisyType = VERY_GOOD;

        } else if (gaussian >  nLevels[1] &&
           gaussian <= nLevels[2]) {

            noisyType = GOOD;

        } else if (gaussian >  nLevels[2] &&
           gaussian <= nLevels[3]) {

            noisyType = BAD;

        } else {
            noisyType = DANGEROUS;
        }

    }
```

Класс Random из пакета java.util служит для вычисления значения типа double согласно стандартному нормальному распределению. Это значит, что вычисляются значения из окрестности нуля и примерно 68,2% этих значений принадлежат интервалу от –1 до 1, 95% – интервалу от –2 до 2 и 99,7% – интервалу от –3 до 3. Если мы создаем набор данных, определяя уровни шума, как это показано в листинге 6.3, то согласно программному коду из листинга 6.4 пользователь из категории кредитоспособности «превосходная» может оказаться, в итоге, пользователем из категории кредитоспособности «очень хорошая» с вероятностью примерно 16%. Из этого программного кода также следует, что веро-

ятность принадлежности такого пользователя категории кредитоспособности «плохая» или «опасная» практически равна нулю. Чем ниже уровни шума в наборах данных, тем лучше показатели правильности, которых может добиться классификатор.

6.2. Оценка кредитоспособности с помощью единственного классификатора

Применим каждый из наших классификаторов к данным рассматриваемого случая и оценим правильность их работы. В разделе 6.2.1 будет использован наивный байесовский классификатор. В разделе 6.2.2 мы представим новый классификатор на основе дерева решений. В разделе 6.2.3 мы воспользуемся классификатором на основе нейронной сети. Наша цель – создать базис, позволяющий оценить, добились мы улучшения за счет объединения классификаторов или нет.

6.2.1. Основы применения наивного байесовского классификатора

В листинге 6.5 показаны шаги, позволяющие быстро загрузить, обучить и оценить наивный байесовский классификатор. По умолчанию эти команды обеспечивают загрузку данных для обучающего набора из файла *c:/iWeb2/data/ch06/training-users.txt*. Чтобы загрузить другой файл, используйте имя нужного файла как аргумент. Данные для тестового набора эти команды загрузят из файла *c:/iWeb2/data/ch06/test-users.txt*.

Листинг 6.5. Оценка кредитоспособности с помощью наивного байесовского классификатора

```
UserDataset ds = UserLoader.loadTrainingDataset();

NBCreditClassifier naiveBayes = new NBCreditClassifier(ds);   ← Создание классификатора на основе наивного байесовского алгоритма

naiveBayes.useDefaultAttributes();

naiveBayes.train();   ← Обучение классификатора

UserDataset testDS = UserLoader.loadTestDataset();

CreditErrorEstimator nb_err =
➥  new CreditErrorEstimator(testDS, naiveBayes);

nb_err.run();   ← Оценка ошибки
```

На рис. 6.1 показаны результаты выполнения команд из листинга 6.5 для тестового набора из 20 000 транзакций. Как видите, для этого конкретного набора данных наивный байесовский классификатор позволяет получить достойные результаты с показателем правильности 0,826. Из полученных 20 000 оценок кредитоспособности: 16 520 оказались правильными, а 3480 – ошибочными. *Матрица неточностей* помога-

ет визуально оценить, насколько серьезны наши ошибки. Числа, расположенные вдоль главной диагонали этой матрицы, характеризуют правильные классификации. Числа, отклоняющиеся от главной диагонали, свидетельствуют об ошибочных классификациях. Если результат классификации расположен слева от диагонали (как в полученной нами матрице), то кредитоспособность была завышена из-за ошибки классификации. Если результат расположен справа от диагонали, то классификатор недооценил кредитоспособность пользователя.

```
Classification completed in 1.575 seconds.

 Total test dataset txns: 20000
    Classified correctly: 16520, Misclassified: 3480
                Accuracy: 0.826
-------------------------------------------------------------

                CONFUSION MATRIX
-------------------------------------------------------------
         EX     VG     GD     BD     DN
  EX    828     24     18      0      0
  VG    161   2149   1900      4      0
  GD      1    418   8482    800      0
  BD      0      0      0   4208    147
  DN      0      0      0      7    853
```

Рис. 6.1. *Результаты классификации кредитоспособности, полученные с помощью наивного байесовского классификатора*

Визуализация матрицы неточностей, полученной в результате классификации, важна потому, что, как было рассмотрено в главе 5, стоимость ошибочных классификаций в задачах реального мира не является соизмеримой. Кроме того, на основе полученных матриц неточностей двух разных классификаторов можно сформировать метрики для сравнения этих классификаторов; подробнее мы поговорим об этом в разделе 6.3. Класс `CreditErrorEstimator` сохраняет результаты классификации и отвечает за выработку выходных данных, показанных на рис. 6.1.

В листинге 6.6 приведен программный код класса `NBCreditClassifier` без некоторых малозначительных вспомогательных методов. Суть этого класса понятна из представленного в листинге программного кода. Опять-таки, мы видим, что создать специализированный классификатор как расширение класса `NaiveBayes` очень легко; вот небольшой фрагмент программного кода, который вам придется написать. Рассмотрим его подробнее.

Листинг 6.6. Классификатор NaiveBayes для оценки кредитоспособности

```
public class NBCreditClassifier extends NaiveBayes {

  private UserInstanceBuilder instanceBuilder;  ❶

  public NBCreditClassifier(String name, TrainingSet ts,
                            UserInstanceBuilder instanceBuilder) {  ❷
      super(name, ts);
      this.instanceBuilder = instanceBuilder;
  }

  public Concept classify(Instance instance) {  ❸
      return super.classify(instance);
  }

  public Concept classify(User user) {  ❹
      return classify(instanceBuilder.createInstance(user));
  }

  public void useDefaultAttributes() {  ❺
      trainOnAttribute(CreditInstance.ATTR_NAME_JOB_CLASS);
      trainOnAttribute(CreditInstance.ATTR_NAME_INCOME_TYPE);
      trainOnAttribute(CreditInstance.ATTR_NAME_AGE);
      trainOnAttribute(CreditInstance.ATTR_NAME_CAR_OWNERSHIP);
      trainOnAttribute(CreditInstance.ATTR_NAME_CREDIT_SCORE);
      trainOnAttribute(
➥       CreditInstance.ATTR_NAME_MORTGAGE_DOWN_PAYMENT);
      trainOnAttribute(
➥       CreditInstance.ATTR_NAME_MOTOR_BICYCLE_OWNERSHIP);
      trainOnAttribute(
➥       CreditInstance.ATTR_NAME_OTHER_PROPERTY_OWNERSHIP);
      trainOnAttribute(CreditInstance.ATTR_NAME_CRIMINAL_RECORD);
      trainOnAttribute(CreditInstance.ATTR_NAME_BANKRUPTCY);
      trainOnAttribute(
➥ CreditInstance.ATTR_NAME_RETIREMENT_ACCOUNT);
  }
}
```

❶ Класс `UserInstanceBuilder` отвечает за преобразование объекта `User` (пользователь) в объект `Instance` (образец). Если вы работаете над другой задачей (например, оценка автомобилей), ваш специализированный классификатор должен будет иметь похожий вспомогательный класс для преобразования в объект `Instance` объекта `Car` (автомобиль). Именно поэтому мы внесли дополнение в классификатор `NaiveBayes`, который ничего не знает об универсальных объектах, – он имеет дело только с концептами `Concepts` и образцами `Instances`.

Важный шаг, на котором принимается решение, как следует трактовать различные атрибуты – как числовые или как категорийные переменные, – обеспечивает класс `UserInstanceBuilder`.

❷ Конструктору требуются аргументы: имя, набор данных для обучения и экземпляр класса `UserInstanceBuilder`.

❸ Мы подчеркиваем, что в итоге все реализации делегируют классификацию базовому методу классификатора `NaiveBayes`.

❹ Значение класса `UserInstanceBuilder`. Переопределенный метод `classify` задействует этот класс, чтобы создать образец `Instance`, который он передает в базовый метод классификации.

❺ Вы можете настраивать этот метод, помечая некоторые его строки как комментарий. По умолчанию для классификации используются все 11 атрибутов. Если вы закомментируете некоторые из строк этого метода, соответствующие атрибуты не будут задействованы в процессе классификации.

Наивному байесовскому классификатору можно доверять: во многих случаях он обеспечит вам получение приемлемых результатов. Кроме того, этот классификатор очень устойчив, то есть если в вашем обучающем наборе есть какие-то нетипичные образцы, или если какие-то образцы введены в данные ошибочно, это не окажет заметного влияния на результаты. Но, как мы увидим, в случае объединения классификаторов устойчивость не является желательным свойством. Напротив, для тех методов объединения, которые мы представим, необходимо, чтобы базовые классификаторы были *изменчивыми* (*unstable*) (см. раздел 6.4). Мы можем создать изменчивый классификатор с помощью дерева решений. Значит, пойдем дальше и рассмотрим создание и применение дерева решений в контексте нашей задачи.

6.2.2. Основы применения дерева решений

В листинге 6.7 показаны шаги, необходимые для создания, обучения и оценки классификатора на основе дерева решений. Мы не представили деревья решений в главе 5, но включили их краткое описание в пункт 2 раздела «Сделать» этой главы. С точки зрения материала данной главы, конкретная реализация алгоритма на самом деле не имеет особого значения; наш классификатор мог быть построен на основе любого алгоритма классификации. Важно, что эти алгоритмы отличаются друг от друга.

Шаги, представленные в листинге 6.7, похожи на шаги, реализованные в листинге 6.5. Если вы выполняете программный код из листинга 6.7 в том же сеансе оболочки (shell), что и код из листинга 6.5, загрузку обучающего и тестового наборов данных можно пропустить.

Листинг 6.7. Оценка кредитоспособности с помощью классификатора на основе дерева решений

```
UserDataset ds = UserLoader.loadTrainingDataset();

DTCreditClassifier decisionTree =
➥  new DTCreditClassifier(ds);          ← Создание классификатора
                                           на основе дерева решений
decisionTree.useDefaultAttributes();

decisionTree.train();   ← Обучение классификатора

UserDataset testDS = UserLoader.loadTestDataset();

CreditErrorEstimator dt_err =
➥  new CreditErrorEstimator(testDS, decisionTree);

dt_err.run();   ← Оценка ошибки
```

На рис. 6.2 показаны результаты выполнения команд из листинга 6.7. Как видим, классификатор на основе дерева решений работает быстро – на порядок быстрее, чем реализация наивного байесовского алгоритма. Он вырабатывает результаты с показателем правильности 0,8262, чуть лучше показателя правильности, полученного для реализации наивного байесовского алгоритма в применении к одному и тому же набору данных. В частности, при том, что всего было получено 20 000 оценок кредитоспособности, 16 524 случая были классифицированы правильно, а 3476 случаев – неверно. Как мы увидим в следующих разделах, разница в результатах работы наивного байесовского классификатора и классификатора на основе дерева решений не является статистически значимой.

```
Classification completed in 0.132 seconds.

Total test dataset txns: 20000
    Classified correctly: 16524, Misclassified: 3476
                Accuracy: 0.8262
-------------------------------------------------------------

                CONFUSION MATRIX
-------------------------------------------------------------
         EX     VG     GD     BD     DN
  EX    831     24     15      0      0
  VG    164   2321   1725      4      0
  GD      0    585   8319    797      0
  BD      0      0      8   4200    147
  DN      0      0      0      7    853
```

Рис. 6.2. *Результаты классификации кредитоспособности, полученные с помощью классификатора на основе дерева решений*

Если вы выполните команду `decisionTree.printTree()` в командной оболочке, то увидите, что собой представляет дерево решений. На рис. 6.3 показаны первые несколько строк выходных данных метода `printTree()`. Убедитесь, что длина буфера экрана, заданная для вашей командной оболочки, позволяет отобразить несколько тысяч строк. Также обратите внимание на то, что из-за ограничений книжного формата древовидная структура на рисунке не видна. Первый атрибут, который рассматривает классификатор, – это наличие предшествующих записей о судимости. Согласно дереву решений, если вас идентифицируют как человека с судимостью, об ипотеке можно забыть! Если образец ссылается на пользователя без криминального прошлого, то по алгоритму дерева решений выясняется, относится ли этот образец к пользователю, который был объявлен банкротом. Если банкротство объявлялось, кредитоспособность этого пользователя будет классифицирована как «плохая». В противном случае анализ атрибутов продолжается, как это видно по выходным данным на вашем экране.

```
Node:attrName=priorCriminalRecord,isLeaf=false,concept=null
-> Branch: [priorCriminalRecord=1]
     Node:attrName=null,isLeaf=true,concept=DN
-> Branch: [priorCriminalRecord=0]
Node:attrName=priorDeclaredBankruptcy,isLeaf=false,concept=null
 -> Branch: [priorDeclaredBankruptcy=1]
      Node:attrName=null,isLeaf=true,concept=BD
-> Branch: [priorDeclaredBankruptcy=0]
   Node:attrName=carOwnership,isLeaf=false,concept=null
-> Branch: [carOwnership=1]
   Node:attrName=mortgageDownPayment,isLeaf=false,concept=null
-> Branch: [mortgageDownPayment=3]
   Node:attrName=otherPropertyOwnership,isLeaf=false,concept=null
-> Branch: [otherPropertyOwnership=1]
   Node:attrName=retirementAccount,isLeaf=false,concept=null
-> Branch: [retirementAccount=3]
   Node:attrName=creditScore,isLeaf=false,concept=null
-> Branch: [creditScore=3]
   Node:attrName=null,isLeaf=true,concept=GD
```

Рис. 6.3. *Узлы верхнего уровня дерева решений для данных о кредитоспособности*

Простота интерпретации результатов классификации – одна из многих причин популярности деревьев решений. Но ценность прямой интерпретации сомнительна, если мы имеем дело с объединением 10 или 100 классификаторов. В такой ситуации нас в первую очередь интере-

сует собственно изменчивость деревьев решений. Разумеется, деревья решений – не единственный изменчивый алгоритм классификации. Еще одна разновидность, как правило, изменчивых классификаторов – классификаторы на основе нейронных сетей.

6.2.3. Основы применения нейронных сетей

Добавим в наш набор еще один классификатор, основанный на реализации нейронной сети. В листинге 6.8 показано, как можно оценить кредитоспособность пользователей с помощью созданного нами классификатора на основе нейронной сети. Если вы выполняете программный код из листинга 6.8 в том же сеансе командной оболочки, в котором выполняли код из листингов 6.5 или 6.7, можете пропустить загрузку обучающего и тестового наборов данных.

Листинг 6.8. Оценка кредитоспособности с помощью классификатора на основе нейронной сети

```
UserDataset ds = UserLoader.loadTrainingDataset();

NNCreditClassifier neuralNet =
➥  new NNCreditClassifier(ds);          ← Создание классификатора на базе нейронной сети

neuralNet.setLearningRate(0.025);        ← Задание коэффициента обучения для обратного распространения
neuralNet.useDefaultAttributes();

neuralNet.train();                       ← Обучение классификатора

UserDataset testDS = UserLoader.loadTestDataset();

CreditErrorEstimator nn_err =
➥  new CreditErrorEstimator(testDS, neuralNet);

nn_err.run();                            ← Оценка ошибки
```

Как видите, единственное различие этих двух листингов – настройка классификатора. Это преднамеренная и важная повторяемость. Когда речь идет о сравнении классификаторов (или чего-то еще), вы должны по возможности стремиться к методичности и систематичности. С увеличением числа классификаторов есть шанс утратить уверенность в том, что вы сравниваете яблоки с яблоками. На рис. 6.4 показаны результаты выполнения программного кода из листинга 6.8.

Класс `iweb2.ch6.usecase.credit.NNCreditClassifier` инкапсулирует пользовательский классификатор на основе нейронной сети для рассматриваемого нами случая оценки кредитоспособности. Сама по себе нейронная сеть реализована в классе `iweb2.ch6.usecase.credit.UserCreditNN` и показана в листинге 6.9. Проанализируем этот класс (определения всех связей или маловажных методов здесь опущены).

```
Classification completed in 0.266 seconds.

 Total test dataset txns: 20000
    Classified correctly: 14330, Misclassified: 5670
                Accuracy: 0.7165
_____________________________________________________________

                CONFUSION MATRIX
_____________________________________________________________
         EX     VG     GD     BD     DN
  EX    498      0    372      0      0
  VG     91      0   4100     23      0
  GD      0      0   8804    897      0
  BD      0      0     33   4175    147
  DN      0      0      0      7    853
```

Рис. 6.4. *Результаты классификации кредитоспособности, полученные с помощью классификатора на основе нейронной сети*

Листинг 6.9. Специализированная нейронная сеть для оценки риска ипотечного кредитования

```
public class UserCreditNN extends BaseNN {

    public UserCreditNN(String name) {
        super(name);
        create();
    }

    public void create() {
        createNN_11_7_5();
    }

    private void createNN_11_7_5() {

        Layer inputLayer = createInputLayer(0, 11);

        Layer hiddenLayer = createHiddenLayer(1, 7,
        new double[] { 0.5, -1, 1.5, 0.5, 1, -0.2, 0.1 });

        Layer outputLayer = createOutputLayer(2, 5,
                new double[] {-1.5, 0.5, -1, 0.5, 1});

        setInputLayer(inputLayer);
        setOutputLayer(outputLayer);
        addHiddenLayer(hiddenLayer);

        setLink("0:0", "1:0", 0.25);
        setLink("0:0", "1:1", -0.7);
        setLink("0:0", "1:2", 0.25);
```

Определение уровня ввода с 11 узлами

Определение скрытого уровня с 7 узлами

Определение уровня вывода с 5 узлами

Определение связанности узлов

```
        setLink("0:0", "1:3", 0.25);
        setLink("0:0", "1:4", -0.3);
        setLink("0:0", "1:5", 0.25);
        setLink("0:0", "1:6", -0.5);

        setLink("0:1", "1:0", 0.25);
        setLink("0:1", "1:1", -0.5);
        setLink("0:1", "1:2", 0.25);
        setLink("0:1", "1:3", 0.25);
        setLink("0:1", "1:4", 0.50);
        setLink("0:1", "1:5", 0.25);
        setLink("0:1", "1:6", 0.50);

    [... Вырезано ...]

    }
}
```

Определение связанности узлов

Выбор входного уровня, содержащего 11 узлов, продиктован нашим решением использовать 11 атрибутов, а выбор 7 узлов для скрытого уровня и 5 узлов для выходного уровня является произвольным. В сущности, в этой сети могло бы быть 2 или больше скрытых уровней. Пусть это послужит напоминанием о том, что в главе 5 мы обсуждали вопрос о чрезвычайной сложности моделирования нейронной сети. Высочайшая степень универсальности оплачивается сложностью. Хотя есть правила, выработанные практикой, следование принципам проектирования нейронных сетей не должно быть безоглядным, а созданную согласно этим принципам сеть необходимо будет подвергнуть проверке в контексте ее применения. В одном из пунктов раздела «Сделать» мы приглашаем вас создать свою нейронную сеть, определив собственную архитектуру в классе `UserCreditNN`. После этого вы сможете воспользоваться рассматриваемыми в этой главе методами сравнения двух и более нейронных сетей, на практике наблюдая последствия принятых вами проектных решений.

В этом разделе представлены три классификатора, которые мы будем использовать далее в этой главе. Еще одно последнее замечание, которое мы хотим сделать, касается времени обучения нейронной сети. Помимо характеристик, гарантирующих качество классификатора, всегда следует учитывать две важнейшие характеристики производительности – время, затрачиваемое на обучение, и время выполнения классификации. Временные показатели, полученные на стадии обучения, выводятся на экран в командной оболочке после обращения к методу `train()`, которое инициируется для всех классификаторов (листинги 6.5–6.8). Для наивного байесовского классификатора, классификатора на основе дерева решений и нейронной сети эти временные показатели равны 0,5, 5,6 и 265,6 секунды соответственно. Время, затрачиваемое на обучение, как правило, не создает проблем; более ценным обычно является время выполнения классификации. Тем не менее, при расхождении показате-

лей времени обучения двух классификаторов на несколько порядков, фактически, можно исключить более медленный (в обучении) классификатор из рассмотрения, особенно в том случае, если ожидается возрастание объема обучающих данных в рабочей системе.

Как мы уже говорили, правильность результатов классификации, выполненной классификатором на основе дерева решений, оценивается как 0,8262; этот показатель лучше показателя правильности, полученного реализацией наивного байесовского алгоритма в применении к этому же набору данных. Оба классификатора, наивный байесовский и на основе дерева решений, выглядят лучше, чем классификатор на основе нейронной сети, но так ли это? Является ли разброс в показателях правильности статистически значимым? Это важные вопросы, на которые необходимо ответить, прежде чем перейти к рассмотрению вариантов объединения любых классификаторов. В следующем разделе мы разработаем инструменты, которые могут помочь ответить на эти вопросы.

6.3. Сравнение классификаторов в применении к одним и тем же данным

Мы представим четыре теста, позволяющих сравнить классификаторы. *Тест Макнемара* (McNemar's test) и *тест на разность пропорций* (difference of proportions test) можно использовать для сравнения двух классификаторов. *Q-тест Кохрана* (Cochran's Q test) и *F-тест* (F test) можно использовать для сравнения трех и более классификаторов. Это статистические тесты, которые требуют знаний в области математической статистики, не предполагаемых у читателей этой книги; тем не менее, общее представление об этих тестах получить нетрудно. Все статистические тесты имеют исходную предпосылку (формальное название этой предпосылки – *нуль-гипотеза* (null hypothesis)), которая доказывается или опровергается на основе численного сравнения двух значений – *статистики* (statistic) и *порогового значния* (threshold value); эти значения можно взять из таблиц или вычислить по известной формуле. Итак, вы увидите, что во всех представленных тестах выполняются два шага. Во-первых, мы вычисляем подходящую статистику. Во-вторых, сравниваем полученное значение с жестко запрограммированным пороговым значением. Если вас интересуют математические подробности, отличной отправной точкой является книга [Kuncheva, L. I.], в данном разделе мы придерживаемся структуры изложения материала в этой книге.

Хотим ли мы выбрать один классификатор из нескольких или создать классификатор на базе нескольких классификаторов, сравнивать их можно только попарно. Есть практическая причина, по которой мы рассматриваем тесты, применяемые только для сравнения трех классифи-

каторов и более: число возможных пар быстро возрастает с ростом числа классификаторов. Фактически, число таких пар равно $N\ (N - 1)/2$, где N – число классификаторов. Следовательно, нужен способ, позволяющий быстро сказать, есть ли у классификаторов в некотором наборе статистически значимые отличия. В конце концов, чтобы выяснить, на какие из классификаторов следует возложить вину, пришлось бы выполнить их попарные сравнения!

Теоретическое обоснование этих тестов выходит за рамки данной книги: заинтересованный читатель найдет ссылки на литературу в конце этой главы и в *приложении C*. С точки зрения практики, вам необходимы три вещи:

- Понимать, что расхождения в показателях правильности классификаторов не всегда являются значимыми, и помнить о том, что существуют тесты для попарных сравнений, а также тесты для сравнения большего числа классификаторов
- Знать, как можно вычислить соответствующую статистику для каждого теста; наш программный код обеспечивает реализацию четырех рассматриваемых здесь тестов, а также типовой абстрактный класс, который вы можете расширить, чтобы создать собственный тест
- Знать, как найти соответствующее пороговое значение для каждого теста. Как правило, такие значения для конкретных уровня значимости и числа степеней свободы (если они применимы) можно взять из общедоступных таблиц. Кроме того, вы можете написать собственные классы для вычисления этих значений на основе их математических определений

6.3.1. Тест Макнемара

При условии, что вы выполняли программный код из листингов 6.5, 6.7 и 6.8 в одном и том же сеансе командной оболочки, продолжайте работу и выполните сценарий, приведенный в листинге 6.10, чтобы сравнить три возможных пары классификаторов. Как и следовало ожидать, тест Макнемара инкапсулирован в классе `McNemarTest`, который требует получения двух аргументов в конструкторе – по одному образцу `ClassifierResults` от каждого классификатора.

Листинг 6.10. Тест Макнемара для сравнения двух классификаторов

```
McNemarTest mnTest1 = new McNemarTest(nb_err.getResults(),
➥  nn_err.getResults());

McNemarTest mnTest2 = new McNemarTest(nb_err.getResults(),
➥  dt_err.getResults());

McNemarTest mnTest3 = new McNemarTest(dt_err.getResults(),
➥  nn_err.getResults());
```

По одному тесту для каждой пары классификаторов

```
mnTest1.evaluate();
mnTest2.evaluate();      Оценка каждого теста
mnTest3.evaluate();
```

Результаты показаны на рис. 6.5, они совпадают с тем, что можно было бы интуитивно предположить, глядя на отчеты о тестовой классификации каждого классификатора. Нейронная сеть обеспечила получение результатов с более низким показателем правильности, что статистически значимо, согласно проводимому тесту. Переменная `Chi2`, которую вы видите в выходных данных, называется статистикой χ^2 (хи-квадрат), и ее вероятностное распределение можно определить для n степеней свободы; в нашем случае мы принимаем $n = 1$. Подробно статистика χ^2 описана в п. 4 раздела «Сделать» этой главы, а также в литературе, ссылки на которую приведены в *приложении C*.

Статистику χ^2 можно использовать несколькими разными способами. В нашей реализации мы, следуя за [Dietterich, T. G.] и [Kuncheva, L. I.], вычисляем χ^2, как показано в листинге 6.11. Переменная `n01` означает, сколько раз первый классификатор выполнил ошибочную классификацию, тогда как второй классификатор выполнил эту же классификацию правильно, переменная `n10` означает, сколько раз было верно обратное.

Листинг 6.11. Оценка статистики χ2 в тесте Макнемара

```
protected void calculate() {
  int n = c1.getN();

  for(int i = 0; i < n; i++) {

    if( c1.getResult(i) && c2.getResult(i) ) {

        n11++;        ← Оба классификатора обеспечили получение правильных результатов

    } else if( c1.getResult(i) && !c2.getResult(i) ) {

            n10++;    ← Результаты первого классификатора правильны, второй классификатор ошибся

    } else if( !c1.getResult(i) && c2.getResult(i) ) {

            n01++;    ← Первый классификатор ошибся, результаты второго классификатора правильны

    } else {
            n00++;    ← Ошиблись оба классификатора
    }
  }
  double a = Math.abs(n01 - n10) - 1;
  chi2 = a * a / (n01 + n10);   ← Значение статистики χ2
}
```

Обратите внимание: абсолютное значение гарантирует, что результат операции вычитания для значений `n01` и `n10` является симметричным.

```
bsh % mnTest1.evaluate();
      Evaluating classifiers
      NBCreditClassifier and NNCreditClassifier:
      ------------------------------------------------------
      NBCreditClassifier accuracy: 0.826
      NNCreditClassifier accuracy: 0.7165
      N = 20000, n00=3050, n10=2620, n01=430, n11=13900
      ------------------------------------------------------
      Confidence Interval             : 0.05
      Degrees of Freedom              : 1
      Statistic threshold (Chi-square): 3.841
      ------------------------------------------------------
      Chi2 = 1571.0560655737704 > 3.841
      The two classifiers are different: TRUE

bsh % mnTest2.evaluate();
      Evaluating classifiers
      NBCreditClassifier and DTCreditClassifier:
      ------------------------------------------------------
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262
      N = 20000, n00=3252, n10=224, n01=228, n11=16296
      ------------------------------------------------------
      Confidence Interval             : 0.05
      Degrees of Freedom              : 1
      Statistic threshold (Chi-square): 3.841
      ------------------------------------------------------
      Chi2 = 0.01991150442477876 <= 3.841
      The two classifiers are different: FALSE

bsh % mnTest3.evaluate();
      Evaluating classifiers
      DTCreditClassifier and NNCreditClassifier:
      ------------------------------------------------------
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165
      N = 20000, n00=2872, n10=2798, n01=604, n11=13726
      ------------------------------------------------------
      Confidence Interval             : 0.05
      Degrees of Freedom              : 1
      Statistic threshold (Chi-square): 3.841
      ------------------------------------------------------
     Chi2 = 1413.6534391534392 > 3.841
      The two classifiers are different: TRUE
```

Рис. 6.5. *Результаты применения теста Макнемара для трех пар классификаторов*

Кроме того, возведение числителя в квадрат гарантирует, что и окончательный результат будет симметричным. Другими словами, порядок, в котором результаты работы классификаторов передаются в конструктор класса McNemar, значения не имеет. После того как вычислено значение статистики, сравниваем его с пороговым значением 3,841, которое совпадает со значением вероятностного распределения χ^2 для *уровня значимости* (level of significance), равного 0,05; если смысл этого вам не понятен, обратитесь к литературе, ссылки на которую даны в *приложении C*, чтобы разобраться в том, как определяется уровень значимости статистического теста. Другими словами, если значение статистики χ^2 больше, чем 3,841, мы вполне уверены в том, что эти два классификатора отличаются статистически значимым образом.

В одном из пунктов раздела «Сделать» мы предлагаем вам применить статистику χ^2 несколько иным способом. Основная идея заключается в том, чтобы детально исследовать, были ли результаты классификации двух разных классификаторов получены из одного и того же вероятностного распределения. В случае задач, подобных той, которую мы здесь рассматриваем, – оценка кредитоспособности претендентов на получение ипотечного займа с распределением их по пяти категориям, – предлагаемый нами подход может оказаться более подходящей метрикой для сравнения, чем тест Макнемара. Подробности – в соответствующем пункте раздела «Сделать»!

6.3.2. Тест на разность пропорций

Опять-таки, предполагаем, что вы выполнили программный код из листингов 6.5, 6.7 и 6.8 в одном сеансе командной оболочки. Если это так, продолжайте и выполните сценарий, показанный в листинге 6.12, чтобы сравнить три возможных пары классификаторов с помощью теста на разность пропорций. Этот тест инкапсулирован в классе Diff2PropTest, конструктор которого требует двух аргументов – точно так же, как и класс McNemarTest.

Листинг 6.12. Тест на разность пропорций для сравнения двух классификаторов

```
Diff2PropTest d2pTest1 = new Diff2PropTest(nb_err.getResults(),
➥  nn_err.getResults());
Diff2PropTest d2pTest2 = new Diff2PropTest(nb_err.getResults(),
➥  dt_err.getResults());

Diff2PropTest d2pTest3 = new Diff2PropTest(dt_err.getResults(),
➥  nn_err.getResults());

d2pTest1.evaluate();
d2pTest2.evaluate();
d2pTest3.evaluate();
```

По одному тесту для каждой пары классификаторов

Оценка каждого теста

Результаты, показанные на рис. 6.6, подтверждают результаты, полученные для теста Макнемара. Нейронная сеть вырабатывает результаты с более низким показателем правильности, который согласно также и этому тесту является статистически значимым. В переменной z, которую вы видите в выходных данных, хранится значение статистики, чье

```
bsh % d2pTest1.evaluate();
      Evaluating classifiers
      NBCreditClassifier and NNCreditClassifier:

      ------------------------------------------------------
      NBCreditClassifier accuracy: 0.826
      NNCreditClassifier accuracy: 0.7165

      ------------------------------------------------------
      Confidence Interval             : 0.05
      Statistic threshold (Std Normal): 1.96

      ------------------------------------------------------
      |z| = 26.069696745398772 > 1.96
      The classifiers are different: TRUE

bsh % d2pTest2.evaluate();
      Evaluating classifiers
      NBCreditClassifier and DTCreditClassifier:

      ------------------------------------------------------
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262

      ------------------------------------------------------
      Confidence Interval             : 0.05
      Statistic threshold (Std Normal): 1.96

      ------------------------------------------------------
      |z| = -0.05276718103090302 <= 1.96
      The classifiers are different: FALSE

bsh % d2pTest3.evaluate();
      Evaluating classifiers
      DTCreditClassifier and NNCreditClassifier:

      ------------------------------------------------------
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165

      ------------------------------------------------------
      Confidence Interval             : 0.05
      Statistic threshold (Std Normal): 1.96

      ------------------------------------------------------
      |z| = 26.12132981820125 > 1.96
      The classifiers are different: TRUE
```

Рис. 6.6. *Результаты применения теста на разность пропорций для трех пар классификаторов*

вероятностное распределение, как предполагается, будет стандартным нормальным распределением; следовательно, его значение равно нулю, а дисперсия – единице. Более подробные сведения о стандартном нормальном распределении ищите в литературе, ссылки на которую приведены в *приложении C*.

В листинге 6.13 показано вычисление z-статистики, распределение которой, как ожидается, будет соответствовать стандартному нормальному распределению. Эта статистика обычно используется в литературе по теме машинного обучения, но допущение о независимости вызывает вопросы. Можете ли вы выяснить причину? (Подсказка: см. [Dietterich, T. G.])

Листинг 6.13. Оценка z-статистики в тесте на разность пропорций

```
double diff = c1.getAccuracy() - c2.getAccuracy();
double mean = 0.5 * (c1.getAccuracy() + c2.getAccuracy());
double b = ( 2.0 * mean * ( 1 - mean ) ) / n;
z = diff / Math.sqrt(b);
```

← Вычисление разности показателей правильности

← Вычисление среднего значения показателя правильности

← Определение z-статистики

В листинге 6.13 в переменной `n` хранится число образцов в тестовом наборе, одном и том же для обоих классификаторов. После того как z-статистика вычислена, ее абсолютное значение сравнивается с числом 1,96: это значение соответствует двустороннему тесту (какой классификатор использовался первым, не должно иметь значения) с уровнем значимости 0,05.

6.3.3. Q-тест Кохрана и F-тест

Перейдем к сравнению трех и более классификаторов. Как и следовало ожидать, эти тесты сложнее тех, которые позволяют сравнить два классификатора. Но основная идея та же: мы оцениваем соответствующую статистику и сравниваем ее с подходящим пороговым значением, которое выбирается с помощью вероятностного распределения самой статистики. Разумеется, в таких случаях используемая статистика будет включать данные всех классификаторов. В листинге 6.14 показаны команды длиной в одну строку, которые вам сейчас надо выполнить, чтобы сравнить три классификатора с помощью двух тестов: Q-теста Кохрана и F-теста (*F* означает *Fisher*, поскольку эту статистику ввел в 1920-х годах Рональд А. Фишер). И снова мы исходим из предположения, что программный код из листингов 6.5, 6.7 и 6.8 вы выполнили в одном сеансе командной оболочки.

Листинг 6.14. Q-тест Кохрана и F-тест для сравнения трех и более классификаторов

```
CochransQTest cqTest = new CochransQTest(
➥  nb_err.getResults(),dt_err.getResults(), nn_err.getResults());
```

```
cqTest.evaluate();

FTest fTest = new FTest(nb_err.getResults(),
➥  dt_err.getResults(), nn_err.getResults());
fTest.evaluate();
```

Q-тест Кохрана основан на следующем допущении: если между классификаторами нет отличий, то статистика q, вычисленная, как показано в листинге 6.15, должна иметь распределение χ^2 с двумя степенями свободы; в общем случае, для N классификаторов, было бы $(N - 1)$ степеней свободы.

Листинг 6.15. Вычисление Q-статистики Кохрана

```
protected void calculate() {

  int n = c1.getN();

  double T = calculateT();   ← Суммарное число правильных классификаций
  double T2 = 0.0;

  for(int i = 0; i < n; i++) {
                                  Число классификаторов,
    double x = 0.0;               правильно классифицировавших
                                  данную предметную сущность
    if( c1.getResult(i) ) {
        x++;
    }
    if( c2.getResult(i) ) {
        x++;
    }
    if( c3.getResult(i) ) {
        x++;
    }
    T2 += (x * x);
  }
  double sum = 0.0;
  sum  =   (double)c1.getNCorrect() * c1.getNCorrect() +
           (double)c2.getNCorrect() * c2.getNCorrect() +
           (double)c3.getNCorrect() * c3.getNCorrect() ;

  double a = L * sum;

  q = (L - 1) * (a - T * T) / (L * T - T2);
}
```

В контексте сравнения классификаторов F-тест был предложен Лунли (Looney). Вычисление F-статистики является трудоемким делом. Вы можете внимательно изучить его особенности в методе calculate() класса FTest; за математическим истолкованием вычисления этой величины мы отсылаем вас к [Looney, S. W.]. С точки зрения практики, важно знать, что F-статистика сравнивается со значением из распределения Фишера–Снедекора (Fisher–Snedecor). В наших целях по-

следнее можно рассматривать как отношение двух распределений χ^2, где каждое распределение χ^2 сначала делится на число его степеней свободы. Одно распределение χ^2 имеет $(N - 1)$ степеней свободы, а другое – $(N - 1) \times (M - 1)$ степеней свободы, где N – число классификаторов, а M – число образцов в тестовом наборе. Пороговые значения из распределения Фишера–Снедекора или, сокращенно, F-распределения, для различных уровней статистической значимости и пар степеней свободы можно найти в онлайн-справочнике «Engineering Statistics Handbook» (*http://www.itl.nist.gov/div898/handbook/*).

Результаты выполнения Q-теста Кохрана и F-теста показаны на рис. 6.7. Тесты не пройдены: сравниваемые классификаторы отличаются – потому что значение каждой статистики больше соответствующего порого-

```
bsh % cqTest.evaluate();
      Evaluating classifiers NBCreditClassifier,DTCreditClassifier,NNCred
itC

      ----------------------------------------------------
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165

      ----------------------------------------------------
      Confidence Interval             : 0.05
      Degrees of Freedom              : 2
      Statistic threshold (chi-square): 5.991

      ----------------------------------------------------
      Q = 2783.821552723059 > 5.991
      The classifiers are different: TRUE

bsh % fTest.evaluate();
      Evaluating classifiers NBCreditClassifier,DTCreditClassifier,NNCred
itC

      ----------------------------------------------------
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165

      ----------------------------------------------------
      Confidence Interval     : 0.05
      Degrees of Freedom (1st): 2
      Degrees of Freedom (2nd): 39998
      Statistic threshold     : 3.08

      ----------------------------------------------------
      F = 264.49439710228575 > 3.08
      The classifiers are different: TRUE
```

Рис. 6.7. *Результаты Q-теста Кохрана и F-теста, выполненных сразу для трех классификаторов*

вого значения. Так и предполагалось, поскольку ранее мы обнаружили, что отличия в показателях правильности, полученных для классификатора на основе нейронной сети и классификаторов на основе наивного байесовского алгоритма и дерева решений были статистически значимыми.

Теперь вы знаете, как сравнить друг с другом любое число классификаторов, следовательно, пора узнать о том, как их можно объединить. В следующем разделе мы рассмотрим методы из категории «слияние решений», то есть тот случай, когда все классификаторы вносят свой вклад в данную классификацию. Подход, предусматривающий отбор классификаторов, когда каждый классификатор отвечает за конкретную область точек данных и, предположительно, успешно работает только в зоне своего влияния, рассматриваться не будет. Однако обратитесь к пункту раздела «Сделать», где говорится о смеси экспертов – методике, попадающей в эту категорию.

6.4. Bagging – самонастраиваемое объединение

Термин bagging, как мы уже сказали, является акронимом двух слов – *bootstrap aggregating* и введен Лео Брейманом [Breiman, L.]. В статистике термин bootstrap (самонастраиваемый) относится к непараметрическому методу оценки выборочного распределения статистики. Что это значит? Допустим, у нас есть набор данных, который содержит числа, и мы хотим вычислить среднее значение распределения, выборкой из которого является этот набор. Обратите внимание: просуммировав все эти числа и разделив полученную сумму на размер набора, вы получите оценку «истинного» среднего значения распределения. Метод bootstrap предназначен для улучшения оценки среднего значения путем создания нескольких наборов данных, которые благодаря *выборке с замещением* (sampling with replacement) частично повторяют исходный набор.

В контексте классификации выборочный набор данных – это помеченные данные, которые мы использовали для обучения и тестирования. Следовательно, главная идея метода самонастраиваемого объединения логически проста. Во-первых, из этого помеченного набора данных берутся случайные выборки, и для каждой выборки обучается один классификатор. Пусть данный образец классифицирует каждый классификатор, и пусть классификатор сохранит доминирующий результат (большинство голосов). Повторяем, случайные выборки создаются не путем получения от нашей системы новых обучающих и тестовых наборов, но посредством выборки с замещением из существующего набора данных.

Несмотря на теоретическую обоснованность, будьте бдительны: тут есть ловушка. Исходная предпосылка метода bagging заключается в том, что результаты разных классификаторов, полученные на повторной выборке из одних и тех же данных, будут разными. В противном

случае не было бы никакого смысла в повторной выборке! Классификаторы, которые вырабатывают разные выходные данные, когда входные данные имеют незначительные отклонения, называются *изменчивыми* (unstable). Следовательно, следует ожидать, что метод bagging будет хорошо работать с изменчивыми классификаторами, такими как нейронные сети и деревья решений. Это связано со сделанным нами ранее наблюдением относительно важности понимания и изучения природы обрабатываемых данных. Кроме того, это связано с опасностью переобучения, о которой мы говорили в главе 5.

Bagging-алгоритм пытается проанализировать разброс данных. Выбирая разные наборы данных для обучения классификаторов, мы создаем классификаторы, которые выделяют разные аспекты данных. Следовательно, если наши данные насыщены информацией, метод bagging гарантирует, что классификаторы зафиксируют многие аспекты этих данных. Метод bagging окажется наиболее уместным в тех случаях, когда у нас есть небольшой набор данных для тестирования и мы хотим классифицировать набор данных гораздо большего размера. Чтобы проиллюстрировать эту особенность, возьмем набор данных, где есть 100 образцов для обучения и 10 000 образцов для тестирования. В листинге 6.16 приведен сценарий, использованный для создания этих данных: он почти идентичен листингу 6.3, однако обратите внимание на отличающиеся размер набора данных для тестирования и уровни шума.

Листинг 6.16. Создание искусственных данных с небольшим набором для обучения и высоким уровнем шума

```
UseCaseData useCaseData = new UseCaseData(100,10000);   ❶

UserType.addNoiseLevel("EX",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("VG",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("GD",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("BD",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("DN",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});

useCaseData.create(false);   ❷
```

Определение уровня шума для каждого класса

❶ Первый аргумент конструктора класса `UseCaseData` – это размер набора данных для обучения. Второй аргумент – размер набора данных для тестирования.

❷ Этот метод инициирует создание данных. Передача в качестве аргумента значения `true` приведет к перезаписи всех существующих файлов. Передача значения `false` означает, что данные будут создаваться только в том случае, если файлы не существуют.

Мы включили наборы данных, созданные с помощью программного кода из листинга 6.16, в сопровождающий эту книгу дистрибутив. Вы можете создать другие наборы, а еще лучше – применить этот алгоритм

к собственным данным. Результаты будут меняться в зависимости от разнообразия данных, которое удастся обеспечить и проанализировать. В общем случае, по мере того как размер набора данных для обучения возрастает, улучшение показателя правильности классификации становится статистически незначащим. Самые серьезные улучшения в результате применения метода bagging обеспечивает разумный выбор наборов данных для обучения, которые используются в процессе самонастройки (bootstrap), потому что, поступая указанным образом, мы обеспечиваем максимальное разнообразие. В связи с этим читайте работу [Friedman, J., T. Hastie, and R. Tibshirani], посвященную методу boosting: изложенные там основные идеи также относятся к методу bagging.

6.4.1. Bagging-классификатор в действии

Итак, пора воспользоваться bagging-алгоритмом и проанализировать результаты, которые он обеспечивает для нашего набора данных. В листинге 6.17 показаны шаги, необходимые для создания и выполнения класса `BaggingCreditClassifier`, реализующего bagging-алгоритм.

Листинг 6.17. Оценка классификатора `BaggingCreditClassifier`

```
UserDataset ds = UserLoader.loadTrainingDataset();

BaggingCreditClassifier bagClassifier =
➥  new BaggingCreditClassifier(ds);          ❶

bagClassifier.setVerbose(false);      ← Чтобы увидеть результаты, передайте значение true

TrainingSet ts1 = bagClassifier.getBootstrapSet();    ❷
DTCreditClassifier dt1 = new DTCreditClassifier(ts1);
dt1.useDefaultAttributes();
dt1.setPruneAfterTraining(true);
bagClassifier.addMember(dt1);
bagClassifier.train();

UserDataset testDS = UserLoader.loadTestDataset();
CreditErrorEstimator bagee1 =
➥  new CreditErrorEstimator(testDS, bagClassifier);
bagee1.run();                ← Создание для оценки экземпляра объекта класса CreditErrorEstimator

TrainingSet ts2 = bagClassifier.getBootstrapSet();    ← Повторение шагов для всех
DTCreditClassifier dt2 = new DTCreditClassifier(ts2);   участников ансамбля
dt2.useDefaultAttributes();
dt2.setPruneAfterTraining(true);
bagClassifier.addMember(dt2);
bagClassifier.train();

CreditErrorEstimator bagee2 =
➥  new CreditErrorEstimator(testDS, bagClassifier);
bagee2.run();
```

❶ Создаем экземпляр объекта bagging-классификатора и указываем исходный набор данных для обучения.

❷ Создаем первое дерево решений, добавляем его в ансамбль и обучаем классификатор.

Вы можете использовать любой из трех классификаторов, представленных нами в разделе 6.2. Мы выбрали классификатор `DTCreditClassifier`, реализующий алгоритм дерева решений, на основании того факта, что деревья решений являются изменчивыми. Можно также использовать класс `NNCreditClassifier`, реализующий классификатор на основе нейронной сети и также изменчивый. Вы можете использовать класс `NBCreditClassifier`, реализующий наивный байесовский алгоритм, или создать класс `BaggingCreditClassifier`, содержащий смесь всех этих отдельных классификаторов. В сопровождающем настоящую книгу дистрибутиве мы предоставляем дополнительные сценарии, которые вы можете проанализировать, чтобы убедиться в том, что основные шаги всегда одинаковы.

Выходные данные достаточно объемны, поскольку включают матрицы неточностей. В табл. 6.8 мы предоставили сводку полученных результатов с точки зрения правильности и времени выполнения (в миллисекундах). На обучение каждого классификатора на основе дерева решений затрачивалось примерно от 5 до 10 миллисекунд. Что касается измерений времени, важно заметить, что как обучение, так и выполнение осуществляются исключительно быстро: 100 000 образцов при использовании значений 11 атрибутов были классифицированы за пару секунд.

Созданные нами данные характеризует очень высокий уровень зашумленности, отсюда низкие показатели правильности. Однако обратите внимание на их улучшение по мере увеличения числа классификаторов в ансамбле. Заметим также, что это улучшение не обязательно является монотонным. Другими словами, значение показателя правильности, полученного для ансамбля, меняется как функция возрастания числа участников ансамбля, даже при том, что общий показатель правильности улучшается.

Таблица 6.8. Правильность и время выполнения как функция числа классификаторов для классификатора BaggingCreditClassifier из листинга 6.17

Участники классификатора	Правильность	Время выполнения (мс)
1	0,60517	752
2	0,62158	968
3	0,63714	1173
4	0,62955	1327

Участники классификатора	Правильность	Время выполнения (мс)
5	0,646	1540
6	0,63719	1741
7	0,64258	1945
8	0,63536	2194
9	0,64129	2435
10	0,63625	2662
11	0,64305	2870

Это значит, что если вы применяете метод bagging как стратегию улучшения результатов работы классификатора, вам придется отслеживать получаемые результаты и, возможно, удалять участников из ансамбля, если они не улучшают общий показатель правильности. В одном из пунктов раздела «Сделать» предлагается подобное упражнение, а также еще несколько приемов тонкой настройки метода bagging.

Полученные вами результаты вряд ли в точности совпадут с теми, что приведены в табл. 6.8, потому что при каждом выполнении сценария из листинга 6.17 вы будете получать другие выборки данных. Тем не менее порядок показателей времени и тенденции улучшения должны быть очевидны, и они не должны противоречить нашим результатам; мы неоднократно выполняли этот сценарий и каждый раз получали согласованные результаты.

6.4.2. Заглянем внутрь bagging-классификатора

Наш главный класс – это класс `BaggingCreditClassifier`, полученный из абстрактного класса `ClassifierEnsemble`, который вы можете самостоятельно расширить и создать другой ансамбль классификаторов, руководствуясь идеями, на которых основан метод bagging. Программный код класса `BaggingCreditClassifier` приведен в листинге 6.18.

Листинг 6.18. Реализация метода bagging *как расширение класса ClassifierEnsemble*

```
public class BaggingCreditClassifier extends ClassifierEnsemble {

    private UserInstanceBuilder instanceBuilder;   ← Генерация образцов на основе
    private BootstrapTrainingSetBuilder                данных о пользователях
➥   bootstrapTSetBuilder;   ← Создание наборов данных для обучения с помощью самонастройки

    public BaggingCreditClassifier(UserDataset ds) {

        super(BaggingCreditClassifier.class.getSimpleName());
        instanceBuilder = new UserInstanceBuilder(false);
```

```
        TrainingSet originalTSet = instanceBuilder.createTrainingSet(ds);

        bootstrapTSetBuilder =
➥       new BootstrapTrainingSetBuilder(originalTSet);
    }

    public TrainingSet getBootstrapSet() {
        return bootstrapTSetBuilder.buildBootstrapSet();
    }

    public UserInstanceBuilder getInstanceBuilder() {
        return instanceBuilder;
    }

    public Concept classify(User user) {
        return classify(instanceBuilder.createInstance(user));
    }
}
```

Заметим, что до тех пор, пока речь идет об алгоритмах, в этом классе нет ничего особенного. Рассматриваемый классификатор должен обрабатывать данные о кредитоспособности, поэтому его снабдили классом `UserInstanceBuilder`, предназначенным для преобразования ваших данных в форму, соответствующую требованиям универсального интерфейса `Classifier`. Кроме того, мы предоставляем класс `BootstrapTrainingSetBuilder`, позволяющий создавать наборы данных для обучения с помощью самонастройки, как описано выше. В сущности, `BaggingCreditClassifier` — это удобный класс-оболочка. Реальную ценность имеют класс `BootstrapTrainingSetBuilder` и абстрактный класс `ClassifierEnsemble`. В листинге 6.19 приведен исходный программный код класса `BootstrapTrainingSetBuilder`, без комментариев Javadoc и предложений `import`.

Листинг 6.19. Вспомогательный класс для самонастройки исходного набора обучающих данных

```
public class BootstrapTrainingSetBuilder {

  private TrainingSet originalTrainingSet;

  public BootstrapTrainingSetBuilder(TrainingSet originalTrainingSet) {

    this.originalTrainingSet = originalTrainingSet;
  }

 public TrainingSet buildBootstrapSet() {

   int N = originalTrainingSet.getSize();  ❶

   Map<Integer, Instance> instances = originalTrainingSet.getInstances();

   Instance[] selectedInstances = new Instance[N];

   Random rnd = new Random();

   int center = rnd.nextInt(N);  ❷
```

```
    int countN =0;

    while (countN < N) {

      if (countN % (N/5) == 0) {  ❸
          center = rnd.nextInt(N);
      }

      int selectedInstanceId = pickInstanceId(N,center);   ❹

      Instance selectedInstance = instances.get(selectedInstanceId);
      selectedInstances[countN] = selectedInstance;
      countN++;
    }

    TrainingSet tS = new TrainingSet(selectedInstances);

    return tS;
  }

  private int pickInstanceId(int N) {   ❹
    Random rnd = new Random();
    boolean loop = true;
    int selectedInstanceId=-1;

    double scale = (double) (N/2) / 4.0d;

    while (loop) {

      selectedInstanceId = new Double(center +
➥       rnd.nextGaussian()*scale).intValue();

      if (selectedInstanceId >=0 && selectedInstanceId < N) {
         loop=false;
      }
    }
    return selectedInstanceId;
  }
}
```

❶ Каждая выборка набора данных для обучения будет включать столько же образцов, сколько есть в исходном наборе.

❷ Для большего разброса выбираем несколько точек (центров), из окрестностей которых будем отбирать образцы для каждого набора данных для обучения.

❸ Это условие определяет число образцов, которые будут отобраны в окрестности каждого центра.

❹ Самыми важными являются шаг 3 и метод `pickInstanceId`, поскольку вместе они определяют стратегию выбора обучающего набора данных, полученного путем самонастройки. В частности, эти два шага включают определение центра в области возможных образцов с последующим выбором с помощью гауссова распределения, образца

среди соседей этого центра, которое выполняется $N/5$ раз. Правдоподобие (вероятность) выбора образца, находящегося на определенном расстоянии от центра, как функция от N контролируется с помощью переменной `scale`.

Вы можете переопределить эти методы класса или написать собственные, помните только, что выбор обучающих наборов данных крайне важен для успешной работы bagging-алгоритма. На самом деле один из лучших способов добиться улучшения результатов применения bagging-алгоритма – применение изощренных методик выбора обучающих наборов данных. Как по-вашему, какого рода изощренность здесь требуется? Смелее экспериментируйте с различными механизмами выборки и анализируйте полученные результаты. Весь основной программный код, готовый к употреблению, у вас есть; это занятие должно быть увлекательным!

Основная идея метода bagging – создание ансамбля классификаторов. Разумеется, bagging – лишь один из методов, позволяющих объединить классификаторы; вы не ошибетесь, полагая, что можете существенно выиграть в смысле показателя правильности, применив другой способ объединения классификаторов. Поэтому вместо того чтобы предоставить конкретную реализацию bagging-алгоритма, мы создали абстрактный класс, иллюстрирующий общие идеи, которые можно развить и подвергнуть дальнейшему анализу. Фактически, мы используем одну и ту же абстракцию для реализации как bagging-, так и boosting-алгоритма.

6.4.3. Ансамбли классификаторов

Абстрактный класс `ClassifierEnsemble`, программный код которого приведен в листинге 6.20, инкапсулирует базовые элементы и методы, которые вам потребуются для объединения нескольких разных классификаторов, почти так же, как это делается в случае метода bagging. На самом деле, совершенно не важно, являются ли классификаторы, объединенные в ансамбль, разными версиями одного и того же классификатора или это разные версии классификаторов разных типов; необходимо лишь строгое соответствие интерфейсу `Classifier`.

Листинг 6.20. Абстрактный класс для создания классификаторов на основе ансамбля

```
public abstract class ClassifierEnsemble implements Classifier {

    private String name;

    private List<Classifier> baseClassifiers =
➥       new ArrayList<Classifier>();   ← Классификаторы-участники ансамбля

    public ClassifierEnsemble(String name) {
        this.name = name;
    }
```

```
    public Concept classify(Instance instance) {

        ConceptMajorityVoter voter =
➥       new ConceptMajorityVoter(instance);   ← Классификация, определяемая большинством голосов

        for( Classifier baseClassifier : baseClassifiers ) {

            Concept c = baseClassifier.classify(instance);

            voter.addVote(c);   ← Каждый случай классификации предоставляет один голос
        }

        return voter.getWinner();   ← Победитель, выбранный из списка кандидатов
    }

    public boolean train() {
        for( Classifier c : baseClassifiers ) {
            c.train();   ← Ансамбль обучен, когда обучены все его участники
        }

        return true;
    }

    public void addMember(Classifier baseClassifier) {
        baseClassifiers.add(baseClassifier);
    }

    public void removeMember(Classifier c) {
        baseClassifiers.remove(c);
    }
}
```

Логика, реализованная в классе ConceptMajorityVoter, довольно проста: этот класс инкапсулирует коллекцию голосов, которые каждый классификатор «отдал» данному образцу. В этой реализации победитель объявляется по правилу простого большинства. Но у вас могут быть более сложные стратегии отбора «победителя» при проведении классификации с помощью ансамбля классификаторов. Другая стратегия могла бы учитывать тип конкретного классификатора, назначая каждому голосу определенный вес. Еще один вариант стратегии мог бы предусматривать назначение веса, являющегося функцией (вероятностной) показателя правдоподобия каждой классификации. Для полноты картины в листинге 6.21 мы приводим исходный программный код класса ConceptMajorityVoter.

Листинг 6.21. Простая стратегия выбора «победителя» для ансамбля классификаторов

```
public class ConceptMajorityVoter {

    private Map<Concept, Integer> votes =
➥       new HashMap<Concept, Integer>();   ← Сохранение голосов в HashMap

    private Instance i;
```

```
    public ConceptMajorityVoter(Instance i) {
        this.i = i;
    }

    public void addVote(Concept c) {   ← Добавление голоса увеличивает число голосов

        Integer conceptVoteCount = votes.get(c);

        if( conceptVoteCount == null ) {
            conceptVoteCount =  new Integer(1);
        } else {
            conceptVoteCount = conceptVoteCount + 1;
        }

        votes.put( c, conceptVoteCount );
    }

    public Concept getWinner() {   ← Реализация правила простого большинства

        int winnerVoteCount = 0;
        Concept winnerConcept = null;

        for(Map.Entry<Concept, Integer> e : votes.entrySet()) {
            if( e.getValue() > winnerVoteCount ) {
                winnerConcept = e.getKey();
                winnerVoteCount = e.getValue();
            }
        }

        return winnerConcept;
    }

    public int getWinnerVoteCount() {   ← Минимум, необходимый для победы
        Concept winner = getWinner();
        return votes.get(winner);
    }
}
```

В заключение хотим упомянуть, что с точки зрения размера обучающего набора данных есть наиболее чувствительная зона, где наблюдается наибольший разброс, вызванный вариативностью входных данных. Эта наиболее чувствительная зона будет зависеть от разновидности классификаторов-участников и методологии обучения. Если набор данных мал, выгоды, которые обеспечивает ансамбль, обученный по методу bagging, не могут компенсировать снижение показателя правильности отдельных моделей, каждая из которых теперь видит еще меньший набор данных для обучения. На другом конце спектра, если набор данных исключительно велик, а время вычислений не является проблемой, вполне может хватить даже одного универсального классификатора.

Таким образом, метод bagging пожинает свои плоды, с выгодой используя разброс данных в обучающем наборе. Если мы получаем обучающие наборы данных, репрезентативные с точки зрения этого разброса,

наш классификатор способен будет «понять» большую часть аспектов (если не все) таких «разбросанных» данных. Если вы также прочтете и проработаете соответствующий пункт раздела «Сделать», психологически вы готовы к применению метода улучшения boosting!

6.5. Boosting – итеративный подход к улучшению

Главная идея метода boosting (улучшение) заключается в инкрементном наращивании ансамбля классификаторов таким образом, что новые классификаторы улучшают результаты классификации на тех же самых данных, на которых их предшественники потерпели неудачу. В bagging-процессе мы создавали бы входящие в ансамбль классификаторы независимо друг от друга, в надежде на то, что новые участники улучшат показатель правильности классификаций. В результате, успешное применение метода bagging зависит от тщательности выбора обучающего набора данных, но узнать заранее, будет ли этот метод работать, нельзя.

В процессе boosting-улучшения мы итеративно наращиваем ансамбль классификаторов, так что результаты классификации улучшаются. Вместо того чтобы полагаться на случайный выбор данных, которые *могли бы* быть подходящими наборами данных для обучения, мы задаемся целью улучшить результаты классификации путем тщательного отбора обучающих наборов с акцентом на те образцы, которые в предыдущих случаях были неверно классифицированы ансамблем.

Можно провести параллель между методом улучшения boosting и реальной практикой консультирования. Допустим, есть комитет, предназначенный для консультирования по какой-то корпоративной финансовой проблеме. Эта проблема может быть сложной, и ее решение могло бы зависеть от нескольких факторов. В отношении какого-то из этих факторов, скажем, фактора X, члены комитета не уверены в правильности предлагаемого ими решения. В таком случае имело бы смысл ввести эксперта, специализирующегося по вопросу X. Как только это сделано, комитет обладает всеми знаниями, необходимыми для принятия компетентных решений и выработки ценных рекомендаций по исследуемой финансовой проблеме. В этом и состоит суть boosting-улучшения, которую можно кратко сформулировать так: надо выяснить, что именно вам неизвестно, и пригласить того, кто восполнит недостающие знания!

Мы опишем boosting-алгоритм *arc-x4*, представленный Лео Брейманом [Breiman, L.]. Брейман ввел в употребление термин *arcing* для обозначения типа классификаторов, которые выполняют *адаптивную повторную выборку и объединение* (*a*daptive *r*esampling and *c*ombin*ing*). Алгоритм arc-x4 похож на другой алгоритм boosting-улучшения – AdaBoost, название которого получено путем конкатенации терминов *adaptive* (адаптивный) и *boosting* (улучшение). AdaBoost – известный и широко используемый алгоритм boosting-улучшения, впервые опи-

санный в статье [Freund, Y., and R. E. Schapire]. Вы легко можете внести соответствующие изменения в алгоритм arc-x4, получив заодно реализацию алгоритма AdaBoost. Алгоритм AdaBoost имеет прочный теоретический фундамент и является результатом строгого вывода, тогда как алгоритм arc-x4 – это специально подобранный алгоритм. С алгоритмической точки зрения, между ними есть два ключевых отличия, которые мы рассмотрим позже.

6.5.1. Boosting-классификатор в действии

Воспользуемся алгоритмом arc-x4, чтобы улучшить работу классификатора на основе дерева решений. Предназначенный для этого сценарий приведен в листинге 6.22. Структурой он похож на программный код для реализации bagging-алгоритма (см. листинг 6.17).

Листинг 6.22. Оценка классификатора Boosting*CreditClassifier*

```
UserDataset ds = UserLoader.loadTrainingDataset();

BoostingCreditClassifier arcx4 = new BoostingCreditClassifier(ds);  ❶

arcx4.setClassifierType("decision tree");  ❷

arcx4.setClassifierPopulation(1);  ❸

arcx4.setVerbose(false);

arcx4.train();

UserDataset testDS = UserLoader.loadTestDataset();

CreditErrorEstimator arcx4ee  =
➥  new CreditErrorEstimator(testDS, arcx4);  ❹

arcx4ee.run();

arcx4.setClassifierPopulation(3);
arcx4.train();
CreditErrorEstimator arcx4ee = new CreditErrorEstimator(testDS, arcx4);  ❺

arcx4ee.run();
```

❶ Создаем экземпляр класса классификатора, который будет использовать boosting-улучшение, и передаем ему ссылку на обучающий набор данных

❷ Выбираем алгоритм дерева решений в качестве базового классификатора ансамбля.

❸ Определяем состав ансамбля: в данном случае в ансамбле только один участник.

❹ Создаем экземпляр класса `CreditErrorEstimator`, чтобы оценить результаты работы классификатора.

❺ Добавляем в ансамбль новых участников и снова оцениваем результаты классификации.

Вы можете использовать любой классификатор из тех, что мы представили в разделе 6.2. В листинге 6.22 мы использовали класс `DTCreditClassifier`, вызвав его метод `setClassifierType` с аргументом `decision tree`. Чтобы сравнить результаты, полученные с применением boosting-улучшения, с результатами метода bagging, имеет смысл использовать тот же базовый классификатор, что и в листинге 6.17. Обоснование нашего выбора остается прежним – изменчивость деревьев решений. Кроме того, деревья решений демонстрируют самое короткое время выполнения, что удобно, когда вы работаете в интерактивной командной оболочке. Но вы можете с тем же успехом использовать класс `NNCreditClassifier`, передав его конструктору строку `neural network` (регистр значения не имеет), или класс `NBCreditClassifier` со строкой `naive bayes` (опять-таки, регистр не имеет значения) в качестве аргумента.

Отвлечемся и обратим внимание на то, что вы можете создать «гибридный» класс `BoostCreditClassifier`, содержащий смесь всех этих базовых классификаторов, вместо того чтобы использовать только один из них. Такой подход не является стандартной разновидностью алгоритма AdaBoost или arc-x4, но в нем есть смысл. Возможно, для определенного набора образцов один тип классификаторов будет работать лучше других, и если вы можете задействовать этот факт, не стоит упускать возможность осуществить инкрементное улучшение. Boosting-алгоритм является настраиваемым; в принципе, вы должны иметь возможность настроить выбор не только обучающего набора данных, но и типа классификатора. В одном из пунктов раздела «Сделать» этой главы мы предлагаем вам реализовать такое обобщение.

Проанализируем выходные данные, полученные в результате выполнения программного кода из листинга 6.22. Мы свели эти результаты в табл. 6.9, где демонстрируем значение показателя правильности и время выполнения (в миллисекундах) сценария из листинга 6.22; эта таблица похожа на табл. 6.8.

Таблица 6.9. Правильность и время выполнения как функция числа классификаторов для классификатора BoostCreditClassifier из листинга 6.22

Участники классификатора	Правильность	Время выполнения (мс)
1	0,57314	1108
3	0,63862	1638
5	0,64203	2153
7	0,65317	2792

Таблица 6.9 (продолжение)

Участники классификатора	Правильность	Время выполнения (мс)
11	0,64668	3947
31	0,65572	9828
41	0,65676	12870
61	0,66044	19032

Как видите, метод boosting обеспечивает несколько лучшие результаты классификации, нежели метод bagging (см. табл. 6.8). С точки зрения производительности вычислений, следует отметить, что если в качестве базового используется наивный байесовский классификатор, показатели времени, затрачиваемого на обучение, по-прежнему малы и не очень заметно отличаются от показателей времени обучения, полученных для дерева решений. Показатели правильности, которые обеспечил ансамбль наивных байесовских классификаторов, были намного лучше полученных для ансамбля классификаторов на основе деревьев решений, когда ансамбль состоял лишь из 11 участников. Но показатели времени выполнения существенно отличаются в другую сторону. Для случая, когда в ансамбле классификаторов имеется только 11 участников, время выполнения boosting-алгоритма для дерева решений примерно в 63 раза меньше времени выполнения этого же алгоритма для ансамбля наивных байесовских классификаторов.

Тут вы, возможно, поинтересуетесь, являются ли отличия в показателях правильности статистически значимыми. Вы можете воспользоваться тестом Макнемара (см. раздел 6.3.1) и провести прямое сравнение методов bagging и boosting. Какой бы классификатор вы ни выбрали в качестве базового и какими бы ни были другие выбранные вами параметры, этот тест всегда позволяет сравнить два классификатора. С инженерной точки зрения, надо знать, как соотносятся между собой сложность программного кода, правильность классификации, время обучения и время выполнения. Стоимость вашей конкретной задачи следует считать функцией этих параметров. В общем случае, чтобы принять техническое решение относительно выбора алгоритма, одного только показателя правильности недостаточно.

6.5.2. Заглянем внутрь boosting-классификатора

Наш главный класс `BoostingCreditClassifier` является расширением класса `BoostingARCX4Classifier`. Последний, в свою очередь, является расширением класса `ClassifierEnsemble`, очень похожим на то расширение, каким был класс `BaggingCreditClassifier`. Это не должно вас удивлять, поскольку boosting-алгоритм, как и bagging-алгоритм, позволяет объ-

единить различные классификаторы в надежде на выработку лучших результатов. Программный код класса `BoostingCreditClassifier` приведен в листинге 6.23.

Листинг 6.23. Реализация, которая полагается на алгоритм arc-x4

```
public class BoostingCreditClassifier extends BoostingARCX4Classifier {

  private UserInstanceBuilder instanceBuilder;
  private ClassifierMemberType classifierType;

  public BoostingCreditClassifier(UserDataset ds) {

      this(BoostingCreditClassifier.class.getSimpleName(), ds,
➥      new UserInstanceBuilder(false));
  }

  public BoostingCreditClassifier(String name, UserDataset ds,
➥    UserInstanceBuilder instanceBuilder) {

      this(name, instanceBuilder,
➥     instanceBuilder.createTrainingSet(ds));
  }

  public BoostingCreditClassifier(String name,
➥    UserInstanceBuilder instanceBuilder, TrainingSet tSet) {

      super(name, tSet);

      this.instanceBuilder = instanceBuilder;
  }

  public UserInstanceBuilder getInstanceBuilder() {
      return instanceBuilder;
  }

  public Concept classify(User user) {
      return classify(instanceBuilder.createInstance(user));
  }

  @Override
  public Classifier
➥    getClassifierForTraining(TrainingSet set) {

      Classifier baseClassifier = null;

      switch(classifierType) {

          case NEURAL_NETWORK:
              NNCreditClassifier nnClassifier = new NNCreditClassifier(set);
              nnClassifier.setLearningRate(0.01);
              nnClassifier.useDefaultAttributes();
              baseClassifier = nnClassifier;
              break;
          case DECISION_TREE:
```

Генерация образцов на основе данных о пользователях

Генерация выборки наборов данных для обучения

Создание нового классификатора в соответствии с classifierType

```
                DTCreditClassifier dtClassifier = new DTCreditClassifier(set);
                dtClassifier.useDefaultAttributes();
                dtClassifier.setPruneAfterTraining(true);
                baseClassifier = dtClassifier;
                break;
            case NAIVE_BAYES:
                NBCreditClassifier nbClassifier = new NBCreditClassifier(set);
                nbClassifier.useDefaultAttributes();
                baseClassifier = nbClassifier;
                break;
            default:
                throw new RuntimeException("Invalid type!");
        }

        return baseClassifier;
    }

    public ClassifierMemberType getClassifierType() {
      return classifierType;
    }

    public void setClassifierType(String type) {

      if (type.equalsIgnoreCase("decision tree")) {
        this.classifierType = ClassifierMemberType.DECISION_TREE;

      } else if (type.equalsIgnoreCase("neural network")) {
        this.classifierType = ClassifierMemberType.NEURAL_NETWORK;

      } else if (type.equalsIgnoreCase("naive bayes")) {
        this.classifierType = ClassifierMemberType.NAIVE_BAYES;
      }
    }
  }
```

Опять-таки, роль этого класса заключается в инкапсуляции всех составляющих задачи. Обратите внимание: мы обеспечиваем средства для задания типа базового классификатора и позволяем программному коду создавать новые экземпляры автоматически. Сам алгоритм arc-x4 можно найти в абстрактном классе `BoostingARCX4Classifier`, приведенном в листинге 6.24 без некоторых вспомогательных методов, комментариев Javadoc и предложений `import`.

Листинг 6.24. Реализация алгоритма arc-x4 *Бреймана*

```
public abstract class BoostingARCX4Classifier
➥   extends ClassifierEnsemble {                    ← Ансамбль классификаторов

    private TrainingSet originalTSet;

    private int classifierPopulation = 2;

    public BoostingARCX4Classifier(String name, TrainingSet tSet) {
        super(name);
```

```
        this.originalTSet = tSet;
    }

    public Concept classify(Instance instance) {

        ConceptMajorityVoter voter =
        new ConceptMajorityVoter(instance);

        for( Classifier baseClassifier : baseClassifiers ) {

            Concept c = baseClassifier.classify(instance);

            voter.addVote(c);
        }

        return voter.getWinner();
    }

    public abstract Classifier
      getClassifierForTraining(TrainingSet set);

    public boolean train() {

        baseClassifiers = new ArrayList<Classifier>();

        int size = originalTSet.getSize();

        double[] w = new double[size];

        int[] m = new int[size];

        double w0 = 1.0 / size;

        Arrays.fill(w, w0);
        Arrays.fill(m, 0);

        for(int i = 0; i < classifierPopulation; i++) {

            TrainingSet tSet = buildTSet(originalTSet, w);

            Classifier baseClassifier = getClassifierForTraining(tSet);

            baseClassifier.train();

            updateWeights(originalTSet, w, m, baseClassifier);

            baseClassifiers.add(baseClassifier);
        }

        return true;
    }

    public TrainingSet buildTSet(TrainingSet tSet, double[] w) {

        WeightBasedRandom wRnd = new WeightBasedRandom(w);

        int n = w.length;

        Instance[] sample = new Instance[n];
```

Классификация, определенная большинством голосов

Создание классификатора любого типа

Веса определяют выбор образцов

Сколько раз образец был классифицирован неверно

Указание, как надо формировать новый набор данных для обучения

```
        Map<Integer, Instance> instances = tSet.getInstances();

        for(int i = 0; i < n; i++) {

            int instanceIndex = wRnd.nextInt();

            sample[i] = instances.get(instanceIndex);
        }

        return new TrainingSet(sample);
    }

    public void updateWeights(TrainingSet tSet, double[] w,
      int[] m, Classifier baseClassifier) {   ← Указание, как надо обновлять веса

        int n = w.length;

        for(int i = 0; i < n; i++) {

            Instance instance = tSet.getInstance(i);

            Concept actualConcept =
            baseClassifier.classify(instance);

            Concept expectedConcept = instance.getConcept();

            if( actualConcept == null ||
               !(actualConcept.getName()
               .equals(expectedConcept.getName()) ) ) {
                m[i]++;
            }
        }

        double sum = 0.0;

        for( int i = 0; i < n; i++) {
            sum += ( 1.0 + Math.pow(m[i], 4) );
        }

        for( int i = 0; i < n; i++) {
            w[i] = ( 1.0 + Math.pow(m[i], 4) ) / sum;
        }
    }
}
```

Четвертая степень выбрана произвольно

Наша реализация – это еще один экземпляр класса `ClassifierEnsemble`, где классификация основана на большинстве голосов участников ансамбля классификаторов. Ядро алгоритма arc-x4 реализовано в методе `train()` этого класса, где мы вводим вес для каждого образца из обучающего набора данных. Эти веса используются при выборке новых наборов данных для обучения. Изначально все веса равны $1/N$, где N – число образцов в обучающем наборе данных. Но как только в ансамбль добавлен и обучен первый классификатор, мы продолжаем работу с новыми весами, обновленными посредством метода `updateWeights`.

Обратите внимание: каждый раз, когда мы хотим добавить в ансамбль очередной классификатор, выборка нового обучающего набора данных осуществляется не среди унифицированных образцов, но детерминируется только что полученными новыми весами. Такую возможность обеспечивает класс `WeightBasedRandom`, задача которого – отдавать предпочтение образцам, имеющим большие веса, избегая при этом выбора образцов с небольшими весами.

Каждый классификатор добавляет собственную ошибочную классификацию в массив `m`, что, в свою очередь (особым образом, который определяется методом), влияет на веса `w`. Число 4 в названии этого алгоритма объясняется возведением в четвертую степень числа ошибочных классификаций для нормализации и получения значений весов в методе `updateWeights`. Вы можете поэкспериментировать с другим значением для этой экспоненты – взять, скажем, число 5 и назвать это алгоритмом arc-x5. Результаты этого нового алгоритма могут быть лучше или хуже: итог, скорее всего, будет зависеть от ваших данных. Именно из-за того, что экспонента выбирается произвольно, мы считаем этот алгоритм специально подобранным.

Как по-вашему, что произойдет в случае классификатора кредитоспособности? Можно ли получить лучшие результаты путем настройки этого значения? Какие еще вы можете придумать функции, которые связали бы число ошибочных классификаций с весами, так чтобы это улучшило выборку данных для нового обучающего набора?

6.6. Заключение

В этой главе рассмотрена тема объединения классификаторов в контексте оценки кредитоспособности лиц, желающих получить ипотечный кредит. В нашем конкретном случае рассматриваются образцы данных, для описания которых используются 11 атрибутов, откуда вопрос: а сколько атрибутов нам, вообще говоря, необходимо? В процессе представления этих атрибутов мы получили возможность обратить особое внимание на то, как важно понимать природу данных и знать о влиянии, которое ваши решения оказывают на полноту и точность отображения в этих данных реальной информации. В одних случаях объединение классификаторов позволяет значительно улучшить результаты, в других более выигрышным может оказаться применение единственного классификатора. Имея в своем распоряжении мощный генератор данных, вы можете провести дальнейший анализ относящихся к данным условий, определяющих предпочтительность применения одного или нескольких классификаторов.

Разумеется, для того чтобы определить, что один или несколько классификаторов лучше других, необходима возможность оценить статистическую значимость отличий в показателях правильности этих одно-

го или нескольких классификаторов. Мы представили четыре теста, с помощью которых можно сравнивать классификаторы. Способность сравнивать классификаторы важна, поскольку число классификаторов в ансамбле следует по возможности минимизировать, а по максимуму используя доступную информацию. Крайне важно оставить в ансамбле только лучшие классификаторы. Поэтому мы представили тест Макнемара и тест на разность пропорций, которые можно использовать для сравнения двух классификаторов. Мы также познакомили вас с Q-тестом Кохрана и F-тестом для сравнения трех и более классификаторов.

Мы видели, что наши сравнения опираются на статистические тесты, исходная предпосылка (нуль-гипотеза) которых подтверждается или опровергается на основе численного сравнения легкодоступных значений *статистики* и *порога*. Все представленные нами тесты следуют одному шаблону, состоящему из двух шагов. Во-первых, мы вычисляем значение подходящей статистики. Во-вторых, сравниваем полученное значение статистики с пороговым значением.

Наш первый ансамбль классификаторов был основан на методе bagging, цель которого – улучшить показатели правильности классификации за счет создания классификаторов, обученных на разных поднаборах исходного обучающего набора данных. Применяя этот подход, мы надеемся, что получение каждый раз новой выборки (с замещением) из исходных данных обеспечивает выделение различных аспектов данных и, следовательно, мы получаем возможность наращивать суммарный объем знаний ансамбля. Метод bagging, или самонастраиваемое объединение, – это лишь верхушка айсберга возможностей, существующих в данной области. Поэтому мы предоставили абстрактный класс с именем `ClassifierEnsemble`, который вы можете использовать для экспериментов и, надеемся, для встраивания в ваши собственные проекты!

Несколько иной подход к созданию ансамбля классификаторов предлагает метод boosting (улучшение). Вместо того чтобы случайным образом выбирать новые обучающие наборы данных или даже ввести априорные знания о данных в процесс выборки новых обучающих наборов, метод boosting пытается итеративно приблизиться к лучшим результатам за счет преимущественного отбора ошибочно классифицированных образцов, рассматриваемых как более ценные, нежели образцы, классифицированные правильно. Мы представили реализацию boosting-алгоритма arc-x4 Бреймана и проанализировали его показатели правильности и время выполнения на нашем тестовом наборе данных. Главные составляющие этого алгоритма распределены по трем классам – `BoostingCreditClassifier`, `BoostingARCX4Classifier` и `WeightBasedRandom`. Теперь вы можете идентифицировать те области, в которых можно добиться улучшения результатов, получаемых для наших искусственных данных и, что важнее, для ваших собственных реальнх данных!

Этой главой мы завершаем перечисление рассматриваемых в книге средств для интеллектуальных приложений. На практике такие инструменты – просто элементы более общей картины. В следующей главе мы обсудим внедрение этих интеллектуальных элементов в приложение, обслуживающее новостной портал.

6.7. Сделать

1. *Выбор атрибутов*. Мы выбрали 11 атрибутов, чтобы охарактеризовать кредитоспособность отдельного лица. Почему не 10 или 17? Возможно, интуитивно вам кажется, что чем больше используется атрибутов, тем лучше будут результаты, которые мы собираемся получить. Это не обязательно так. С увеличением числа доступных атрибутов особую важность приобретает их правильный выбор. Некоторые алгоритмы, например, деревья решений, способны распознавать информационно насыщенные атрибуты благодаря своей структуре. Другие алгоритмы не так прозрачны, как деревья решений, и вам, возможно, придется непосредственно экспериментировать с различными наборами атрибутов.

   Как бы вы подошли к решению задачи выбора атрибутов? Если изменить атрибуты, определенные в классах семейства `UserType`, одни из них могут оказаться неприменимыми, а другие, возможно, станут доминировать. На какие классификаторы окажут наибольшее влияние такие изменения? Можете ли вы объяснить, почему? Вы можете определить свои атрибуты, еще больше расширив их пространство. На практике важно понимать природу данных и способ функционирования алгоритмов в различных областях конфигурационного пространства в смысле правильности, но также в смысле эффективности использования оперативной памяти и времени вычислений.

2. *Деревья решений* используются в задачах классификации из-за простоты интерпретации результатов выполняемой ими классификации. Представьте древовидную структуру, где каждому узлу назначен какой-то атрибут и каждая выходящая из данного узла связь соответствует конкретному значению назначенного узлу атрибута: строго говоря, каждая связь ассоциируется с предикатом атрибута. Каждый листовой узел в таком дереве соответствует конкретному классу рассматриваемой задачи классификации. Это и есть дерево решений.

   Чтобы понять, как этот механизм работает, представьте, что корневой узел дерева находится сверху, а все остальные узлы располагаются под ним. Для каждого образца, который тредуется классифицировать, мы совершаем обход дерева решения: начинаем с корневого узла, анализируя значение назначенного корневому узлу атрибута. Значение этого атрибута направит нас к следующему узлу; пройдя

по соответствующему ребру, мы анализируем значение атрибута, связанного с узлом первого уровня, и так далее, до тех пор пока не доберемся до листовых узлов. Поскольку каждый листовой узел соответствует некоторой категории, классификация завершена. Логика достаточно проста, правда?

Разумеется, чтобы использовать дерево решений для классификации, прежде всего, необходимо сформировать само дерево! А это уже не так легко, как использовать дерево. Приглашаем вас изучить нашу реализацию и – чтобы лучше вникнуть в тему – прочесть о деревьях решений в книгах [Witten, I. and Frank, E.] и [Dunham, M. H.] (см. раздел 5.9.2).

3. *Сравнение двух и более нейронных сетей.* Проектировать архитектуру нейронных сетей довольно сложно, здесь требуется хорошее понимание нейронных сетей и той задачи, которую вы пытаетесь разрешить. В разделе 6.2.3 мы представили класс `UserCreditNN` – реализацию пользовательской нейронной сети, созданной специально для рассматриваемого нами случая оценки кредитоспособности.

   Полезно поэкспериментировать с нейронными сетями другой архитектуры и сравнить результаты соответствующих классификаторов с помощью представленных в этой главе методик. Например, можно создать сеть, имеющую 11 узлов на уровне ввода, 5 узлов на скрытом уровне и только 1 узел на уровне вывода. В качестве эксперимента можно добавить еще один скрытый уровень в любую из уже имеющихся у вас сетей. Даже в контексте нейронной сети с фиксированной структурой узлов различные сплетения синапсов и разные смещения узлов обеспечат получение разных результатов. Смело исследуйте все доступные возможности.

   Стоит ли трудов усложнение сетевой архитектуры? Имея в распоряжении много параметров, с чего бы вы начали оптимизировать свою структуру и по каким критериям судили бы о ее оптимальности или об отсутствии таковой?

   В связи с этим мы должны сказать, что классификатор на базе нейронной сети был выбран нами как «плохой» классификатор. Поскольку наша цель состояла в сравнении классификаторов, нам нужен был классификатор, результаты которого в достаточной степени отличались бы (в худшую сторону) от результатов двух других классификаторов. Проще всего добиться этого, используя нейронную сеть: при наличии огромного количества вариантов сетевой структуры случайный выбор одного из них окажется субоптимальным с вероятностью, близкой к единице!

4. *Тест χ^2 (хи-квадрат) для двух распределений.* В разделе 6.3.1 для теста Макнемара мы ввели статистику χ^2. Это же вероятностное распределение можно использовать иначе. Для этого необходимо представить задачу сравнения не так, как это делает тест Макнемара. Кон-

текст сравнения остается прежним: есть два классификатора и обычные наборы данных для обучения и тестирования. Если мы обучим и протестируем классификаторы, можно ли будет утверждать, что разность результатов окажется статистически значимой?

Давайте рассмотрим эту оценку следующим образом. Вспомните матрицу неточностей , которую мы представили в главе 5, – эта матрица была напечатана для каждого классификатора в выходных данных в сеансе командной оболочки в результате выполнения метода `CreditErrorEstimator`. Пусть эта матрица хранится в двумерном массиве, например в массиве `int[][] confusion`. Если два классификатора статистически эквивалентны, разумно предположить, что будут «похожи» и значения в их матрицах неточностей. Итак, давайте вычислим, следующую статистику:

```
double chi2=0;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
        int diff = confusion1[i][j]-confusion2[i][j];
        int  sum = confusion1[i][j]+confusion2[i][j];
        chi2 = chi2 + (diff*diff)/sum;
        }
    }
```

где `n` – число классов (в нашем случае `n = 5`), `confusion1` – матрица неточностей первого классификатора, а `confusion2` – матрица неточностей второго классификатора. Напишите новый класс для тестирования, который является расширением класса `Test`, подобным классу `McNemarTest`, но вычисляет вышеупомянутую статистику χ^2. Как бы вы вычислили пороговое значение для этой статистики? Сколько степеней свободы нам нужно? Если статистика включает только диагональные элементы матрицы, будет ли она более показательной? Почему?

(Подсказка: консультацию можно получить в соответствующей литературе, где рассматривается оценка значений статистики хи-квадрат с помощью неполных гамма-функций; необходимые ссылки даны в *приложении C*. Ответ на вопрос о степенях свободы: *n – 1*.)

5. *«Смесь экспертов» – методика отбора классификаторов.* Как мы сказали в начале этой главы, тема объединения классификаторов в целом распадается на два общих направления. Одно направление – это подход с отбором классификаторов, а другое – слияние классификаторов, например, методы bagging (см. раздел 6.4) и boosting (см. раздел 6.5). Как специалисты по программному обеспечению мы можем оценить красоту и эффективность принципа «разделяй и властвуй», на котором основан этот подход к выбору классификаторов.

   Интересный с практической точки зрения и хорошо изученный вариант подхода к выбору классификаторов – *смесь (локальных) экс-*

пертов [Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton], возникший как модульный вариант многоуровневой (управляемой) нейронной сети. Этот подход предполагает, что мы можем улучшить коэффициент обучения и способность к обобщению (правильная классификация не встречавшихся ранее образцов), если будем использовать набор локальных экспертов и фильтрующую сеть (вообще говоря, еще один классификатор). Фильтрующая сеть обеспечивает выбор эксперта, которого следует использовать в каждом случае обучения; в результате, к концу обучения каждый классификатор является экспертом в конкретной области входных данных (образцов, включенных в обучающий набор данных). Подробно этот подход описан в [Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton] и [Jordan, M. I., and R. A. Jacobs].

Создайте три классификатора на основе нейронных сетей, подходящих для случая оценки кредитоспособности, и объедините их с четвертой нейронной сетью (служащей фильтром), чтобы обеспечить смесь экспертов. Здесь по ходу дела возникает ряд вопросов. Какой метод следует применять для обучения? Обеспечивает ли смесь экспертов результаты лучше тех, что мы получили, применив единственную нейронную сеть? Стоит ли это решение затраченных на его реализацию трудов? Какие здесь есть преимущества и недостатки в сравнении с методами bagging и boosting?

6. *«Маленькие хитрости» и советы по применению метода bagging.* В разделе 6.4 вы узнали о bagging, методе улучшения результатов работы классификаторов. Несмотря на принципиальную простоту, оптимизация улучшений, которую может обеспечить метод bagging, требует определенных трудозатрат. Важнейший вопрос bagging-алгоритма – выбор поднаборов данных для обучения классификаторов-участников ансамбля.

   Другим важнейшим решением является определение числа классификаторов в ансамбле. Иначе говоря, сколько классификаторов мы должны использовать – 10, 20, 50, 100? Начните с простого и напишите класс для анализа улучшений, полученных в результате добавления в ансамбль нового классификатора. Первый тест, который вам надо выполнить, применив одну из методик, описанных нами в разделе 6.3, должен выяснить, является ли отличие в результатах классификации двух и более классификаторов статистически значимым. Если да, то можно выполнить второй тест, чтобы проверить, превышает ли улучшение показателя правильности установленный порог, значение которого вы можете настраивать. Если оба теста пройдены, оставляем классификатор в ансамбле; в противном случае отбрасываем этот классификатор и берем новый.

   Испытайте разные классификаторы и проанализируйте полученные результаты. Какие различия и сходства наблюдаются у выбранных вами вариантов? Что происходит, если смешать классификаторы

разных типов? Работает ли в этом случае bagging-алгоритм? Лучше или хуже полученные вами результаты? Проанализируйте эти вопросы в контексте класса, который вы только что написали для оптимизации показателя правильности.

Напишите собственную реализацию метода `pickInstanceId` класса `BootstrapTrainingSetBuilder`. Один из наиболее эффективных способов улучшить результаты работы bagging-алгоритма – использовать более изощренные методики выбора обучающего набора. Как по-вашему, какие усовершенствования необходимы для этого? Не бойтесь экспериментировать с различными механизмами выбора обучающего набора данных и анализируйте результаты. Весь основной, готовый к употреблению программный код у вас есть – развлекайтесь!

7. *Обобщение метода boosting путем адаптации типов классификаторов.* В разделе 6.5 мы представили метод boosting и объяснили работу алгоритма arc-x4. Брейман утверждает, что разработал алгоритм arc-x4 для того, чтобы «показать, что дело не в конкретной форме алгоритма arc-fs, а в адаптивной повторной выборке». (Под arc-fs Брейман имеет в виду алгоритм AdaBoost.)

   Если Брейман прав, мы могли бы создать класс `BoostCreditClassifier`, который содержит смесь базовых классификаторов, вместо того чтобы использовать единственный базовый классификатор. Этот подход не является стандартной разновидностью алгоритма AdaBoost или arc-x4, но он имеет смысл; кроме того, как мы сказали, алгоритм arc-x4 является специально подобранным. Возможно, для определенного набора образцов классификатор определенного типа будет работать лучше, чем классификаторы других типов, и если вы можете воспользоваться этим фактом, не стоит отказываться от попытки осуществить пошаговое улучшение. Boosting является адаптивным методом, и, в принципе, вы можете адаптировать не только выбор обучающего набора данных, но и выбор типа классификатора.

   Предлагаем вам реализовать такое обобщение. Вы можете внести два изменения. Можно было бы пойти по следующему пути: каждый раз, когда требуется добавить в ансамбль нового участника, включать в него представителя каждого типа. Возможно, как вариант, лучше создать ансамбль на основе заранее определенного соотношения в ансамбле классификаторов разных типов: например, 50% – деревья решений, 30% – наивный байесовский алгоритм и 20% – нейронные сети.

   Проанализируйте, как соотносятся выгоды и потери, и придумайте стратегию для получения максимального выигрыша в правильности, не приводящего к нереальному возрастанию времени обучения или вычислений на этапе выполнения. Достаточно ли повысились показатели правильности, чтобы оправдать возросшие сложность и затраты на вычисления? Как еще можно было бы изменить этот

алгоритм? В разделе 6.3 мы узнали, как сравнить два и больше классификаторов. В данном случае мы заинтересованы в распознавании классификаторов, результаты которых отличаются, следовательно, эти методы сравнения могут пригодиться в самих алгоритмах классификации!

6.8. Ссылки

- Bell, R., Y. Koren, and C. Volinsky «Chasing $1,000,000: How we won the Netflix progress prize», *ASA Statistical and Computing Graphics Newsletter*, Vol 18 (2), pp. 4–12, 2007. *http://stat-computing.org/newsletter/v182.pdf*.
- Breiman, L. «Bagging predictors», *Machine Learning*. Vol 24 (2), pp. 123–140, 1996.
- Breiman, L. «Arcing classifiers», *The Annals of Statistics*. Vol 26 (3), pp. 801–849, 1998.
- Dietterich, T. G. «Ensemble methods in machine learning», Multiple Classifier Systems, (Editors: J. Kittler and F. Roli) volume 1857 of Lecture Notes in Computer Science, pp. 1–15. Cagliari, Italy. Springer, 2000. *http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=743935*.
- Freund, Y., and R. E. Schapire «A decision-theoretic generalization of on-line learning and an application to boosting», *Journal of Computer and System Sciences*. Vol 55 (1), pp. 119–139, 1997.
- Friedman, J., T. Hastie, and R. Tibshirani «Additive logistic regression: A statistical view of boosting», *Annals of Statistics*, Vol 28 (2), pp. 337–407, 2000.
- Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton «Adaptive mixtures of local experts», *Neural Computation*. Vol 3, pp. 79–87, 1991.
- Jordan, M. I., and R. A. Jacobs «Hierarchical mixtures of experts and the EM algorithm», *Neural Computation*. Vol 6, pp. 181–214, 1994.
- Kuncheva, L. I. «Combining Pattern Classifiers: Methods and Algorithms», Wiley-Interscience, 2004.
- Looney, S. W. (1988) «A statistical technique for comparing the accuracies of several classifiers», *Pattern Recognition Letters*. Vol 8, pp. 5–9, 1988.

7

Все вместе: интеллектуальный новостной портал

- Получение и обработка интернет-контента
- Поиск новостных сообщений
- Категоризация новостей
- Объединение новостей в группы

В каждой из предыдущих глав была рассмотрена отдельная группа алгоритмов. Мы изучили алгоритмы поиска, выработки рекомендаций, кластеризации и классификации. Для анализа каждого алгоритма был использован свой контекст, для каждого алгоритма рассмотрен конкретный случай его применения. Эти алгоритмы находят применение во множестве других случаев, и все вместе их можно использовать для того, чтобы усовершенствовать готовое приложение.

Цель данной главы – продемонстрировать, как применить каждый из этих алгоритмов в рамках одного приложения. Где можно использовать алгоритм кластеризации? Как повлияет кластеризация на результаты классификации? Обеспечивают ли алгоритмы PageRank и DocRank кроме поиска и другие возможности? Как написать систему выработки рекомендаций для моего приложения, воспользовавшись тем, что я узнал из этой книги? На все вопросы мы ответим в этой главе.

Мы не будем писать веб-приложение в полном объеме – просто представим контекст такого приложения и рассмотрим применение в нем интеллектуальных алгоритмов. Другими словами, мы будем говорить о применении наших алгоритмов в контексте воображаемого веб-приложения.

В частности, в рассматриваемом примере подразумевается существование некоего новостного портала, идея которого навеяна веб-сайтом Google News («Новости Google»).

В процессе встраивания в наше приложение интеллектуальных алгоритмов мы столкнемся с рядом проблем практического характера, о которых вы должны знать. Для многих интеллектуальных алгоритмов может быть важен порядок обработки данных. Поэтому ключевым фактором для успешного объединения всех методик может стать потребность в *метаалгоритме*, задача которого – управлять другими алгоритмами или объединять их.[1]

В этой главе нам также представится возможность повторить весь материал, рассмотренный в предыдущих главах. Начнем с описания конкретного случая применения методики краулинга (обхода веб-контента с помощью поискового робота) и объединения собранных в Интернете новостных сообщений в группы. Эти вопросы рассматривались в главе 2, но теперь мы можем предложить ряд способов, позволяющих усовершенствовать указанные процессы за счет классификации. Чтобы обеспечить повторяемость результатов и организовать базовую структуру данных для обсуждения, мы расширим набор новостных сообщений, который использовался в главе 2, и будем работать с этим расширенным набором.

Продолжим, естественно, разделом, посвященным поиску новостных сообщений для разных запросов. В связи с этим возникает множество интересных вопросов. В рассматриваемом случае (веб-приложение новостного портала) нам не нужны результаты поиска по всему Интернету. Мы хотим получить результаты поиска в массиве документов, обладающих особыми свойствами (имеются в виду новостные сообщения). Из знания этого факта можно извлечь пользу и повысить качество получаемых результатов. Это урок универсальной применимости. Никто не знает ваши данные лучше вас.

В контексте выяснения, какие новости следует объединить в одну группу, и к каким новостным категориям их следует отнести (см. раздел 7.4), мы рассмотрим ряд тем, связанных с тем, что для решения нашей задачи используются несколько интеллектуальных алгоритмов. Если выполнить кластеризацию новостных сообщений, а затем классифицировать созданные кластеры, то результаты будут не такими, как если сначала классифицировать новости, а затем выполнить кластеризацию в рамках каждой новостной категории. В связи с этими отличиями возникает много вопросов, которые мы обстоятельно рассматриваем в разделах 7.4 и 7.5.

В последнем разделе этой главы мы затрагиваем тему выработки рекомендаций. В частности, анализируется случай, когда пользователи

[1] Объединение классификаторов, которое мы рассмотрели в *главе 6*, – частный случай создания и применения метаалгоритмов на практике.

могут оценивать новостные сообщения. В этом случае применение находят все механизмы алгоритмов выработки рекомендаций, разработанные в главе 3. Мы используем и представим подход, основанный на сходстве предметов, и предложим для него несколько дополнений.

Мы написали много практических пунктов для раздела «Сделать», которые должны помочь вам исследовать немало других аспектов интеллектуальных алгоритмов в контексте новостного портала; кроме того, эти указания должны быть полезны для многих других приложений. Пользуйтесь!

7.1. Обзор функциональности

Если вы не видели веб-сайт Google News, введите в своем броузере адрес *http://news.google.com/* и познакомьтесь с действующей иллюстрацией того, что мы собираемся обсуждать. Первое впечатление от этого сайта – просто еще один портал, однако внешность может быть обманчивой. Как мы увидим, между порталом Google News и другими новостными порталами есть некоторые важные отличия. С самого начала мы должны обратить внимание на то, что никогда не были связаны с фирмой Google и не знаем, как и где именно на этом сайте применяются интеллектуальные методики. Мы располагаем сведениями только о той функциональности, которая открыта и доступна обычному пользователю. Наша цель – представить более общий взгляд на материал, а не помочь вам продублировать конкретный веб-сайт.

Что представляют собой упомянутые отличия? В чем особенность сайта Google News? Начать с того что этот сайт генерируется компьютером на основе контента, собранного из тысяч новостных источников по всему миру. Собранные сообщения группируются и распределяются по категориям, таким как «World» («В мире»), «Business» («Бизнес»), «Sci/Tech» («Наука и техника») и так далее. Суть в том, что создание групп и назначение категорий выполняются автоматически, без вмешательства человека. После чтения глав 4 и 5 должно быть понятно, что объединить новостные статьи в группы можно путем кластеризации, назначив категории с помощью классификации, – отсюда важность данного веб-сайта с точки зрения рассматриваемой нами темы. Есть у него и другие интересные для нас функциональные особенности.

На сайте Google News есть раздел «Top stories» («Главные новости»), который создается, по-видимому, на основе оценок релевантности (ранжирования), подобных тем, которые мы описали в главе 2. Если исходить из того, о чем мы уже говорили в этой книге, такую группу новостных сообщений можно было бы определить с помощью рейтинговых оценок, получение которых обеспечивает алгоритм PageRank или метод ранжирования на основе сходства контента, вроде представленного нами алгоритма DocRank. Мы можем использовать оценки, выставленные пользователями конкретным статьям, чтобы определять или хотя бы

корректировать рейтинговые оценки, полученные алгоритмическим путем, и предлагать каждому пользователю улучшенный персональный интерфейс.

Кроме того, персонализация может быть направленной и явной. Ссылка Personalize this page (Персонализировать эту страницу) позволяет реорганизовать набор тематических разделов, представленный на исходной странице. Здесь вы также можете добавить пользовательские разделы, отражающие ваш интерес к конкретной категории новостей. Такого рода персонализация предлагает прекрасную возможность для «добычи данных» на веб-сайтах с большим числом пользователей. За счет кластеризации адаптированного к требованиям конкретного пользователя новостного контента мы можем идентифицировать группы пользователей, имеющих общие интересы. Также можно задействовать рекомендации, применив понятие сходства между статьями, которые читают пользователи, и учитывая, насколько им понравилась та или иная статья. То есть можно создать систему выработки рекомендаций, основанную как на выставленных пользователями рейтинговых оценках, так и на сходстве контента новостных статей, подобную той, что мы представили в главе 3.

Еще одна интересная возможность сайта Google News – раздел «In the News» («Помеченные»). Это список отдельных лиц, некоммерческих организаций, фирм, правительственных органов и так далее, которым отводится «важное» место в сегодняшних новостях. Разумеется, мы можем создать такую функциональную возможность, опираясь исключительно на элементарные статистические данные, но если подумать, значимость конкретного имени зависит от нескольких факторов, а не только от частоты его появления в новостных статьях. Это особенно справедливо для контента, собранного из многочисленных разнородных источников, как на новостном сайте вроде Google News. В таких случаях значительный выигрыш могут обеспечить методы получения метрик релевантности, такие как PageRank и DocRank, с которыми мы встречались в главе 2.

Возможность News Alerts («Уведомления о новостях») позволяет получать контент по электронной почте с учетом заданных вами поисковых термов и типа медиа (например, новостные сообщения, видеозаписи, блоги, списки групп). Уведомления следует высылать только в том случае, если пользователь мог бы отнестись к этому контенту как к «ценной» информации – критерий, который, разумеется, меняется от пользователя к пользователю. Опять-таки, на выручку могут прийти интеллектуальные алгоритмы.

Разработчики сайтов проявили бы небрежность, проигнорировав возможности локализации, поэтому, естественно, на указанном сайте полно настроек, которые зависят от используемого в новостях языка и географического местонахождения пользователя. В нашем контексте такие возможности важны только с точки зрения применения средств

обработки естественного языка (Natural Language Processing, NLP). Превосходный обзор NLP-методик сделан в книге [Jurafsky, D., and J. H. Martin].

Сначала опишем процесс краулинга новостных сайтов и накопление новостных сообщений в нашей системе для их дальнейшей обработки.

7.2. Получение и обработка контента

Отправной точкой новостного портала, такого как Google News, является, конечно, сбор новостей. Сбор статей из разных источников в Интернете осуществляется с помощью поискового робота (crawler). Есть несколько бесплатных поисковых роботов, но для удобства и полноты рассмотрения мы используем робота, которого сами написали для этой книги и который является вариантом робота из главы 2.

Мы также предоставим обзор предварительных условий поиска, поскольку сейчас – пройдя через шесть глав, описывающих интеллектуальные алгоритмы, – можно взглянуть на каждый этап с иной точки зрения. Хотя вы можете с помощью краулинга получить собственные новостные сообщения (или любой другой контент), для наших примеров нужен фиксированный набор данных, обеспечивающий повторяемость результатов. В конце этого раздела мы опишем используемый по умолчанию набор новостных сообщений, с которым будем работать в оставшейся части книги.

7.2.1. На старт, внимание, краулинг!

Как можно было догадаться, класс нашего робота, предназначенного для поиска новостей, называется `NewsCrawler`; его конструктор получает три аргумента:

- Базовый каталог для хранения извлеченных данных
- Глубину, на которую совершается обход ссылочной структуры
- Максимальное число документов, которые требуется извлечь

Давайте воспользуемся этим классом в среде BeanShell и выполним краулинг веб-сайта издательства Manning.[1] В листинге 7.1 показаны несколько шагов, с помощью которых задается конфигурация поискового робота.

Листинг 7.1. Краулинг веб-сайта издательства Manning

```
String rootDir = Ch7Constants.CRAWL_DATA_ROOT_DIR;

NewsCrawler crawler = new NewsCrawler(rootDir, 2, 100);
```

Определение корневого каталога

[1] Просим вас: выполняя краулинг, будьте сознательным гражданином Интернета. Все наши примеры краулинга настроены на небольшую глубину и (относительно) малое число извлекаемых документов.

```
crawler.addSeedUrl("http://www.manning.com");

crawler.run();
```

Первым делом определяем место хранения извлеченных документов. По умолчанию используется папка *C:/iWeb2/data/ch07/news-crawls*, но вы можете хранить контент в любом удобном для вас месте, изменив для этого значение свойства `CRAWL DATA ROOT DIR`. Когда робот приступит к извлечению контента посещенных им URL-адресов, он будет запоминать его в подкаталоге корневого каталога. Имя нового подкаталога включает префикс *crawl-*, за которым следует числовое значение временной метки робота в миллисекундах, например, *crawl-1200697910111*. Класс `NewsCrawler` делегирует свою работу классу `BasicWebCrawler` почти так же, как это делал класс `FetchAndProcessCrawler` в главе 2.

Вспомним, с чего начинается работа поискового робота. Задача класса `BasicWebCrawler` – извлечь данные и выполнить их синтаксический разбор. Результат запоминается в подкаталоге *processed*. Для хранения каждой группы обработанных документов используются четыре подкаталога: *fetched*, *knownurls*, *pagelinks* и *processed*. В каталоге *fetched* хранятся оригинальные HTML-страницы. В каталоге *knownurls* находится единственный файл, который содержит все URL-адреса, обнаруженные в процессе краулинга. В каталоге *pagelinks* также хранится единственный файл со всеми ссылками, связывающими посещенные URL-адреса. Структура каталога *processed* внешне аналогична структуре каталога *fetched*, но в нем мы храним контент веб-страниц, который получен в результате синтаксического разбора. Некоторые свойства веб-страниц, например, заголовок веб-страницы и ее URL-адрес, хранятся отдельно от основного контента. Внешние ссылки, имеющиеся на каждой странице, также должны быть извлечены и храниться отдельно. Если сравнить файлы из подкаталога *fetched* с теми, которые хранятся в подкаталоге *processed*, вы увидите, что отличие между ними заключается в удалении большей части элементов HTML-структуры, таких как заголовки, таблицы, разделители и так далее. На ноутбуке или настольном компьютере со средним быстродействием и средней скоростью подключения к Интернету поисковый робот завершит свою работу за пару минут.

Мы не будем подробно обсуждать поискового робота, поскольку – на данном этапе – он не делает ничего интеллектуального! Элементарный поисковый робот с исходным кодом предоставляется в этой книге затем, чтобы сподвигнуть вас на применение интеллектуальных алгоритмов в процессе краулинга (см. п. 1 раздела «Сделать»).

7.2.2. Обзор предварительных условий поиска

В главе 2 мы проанализировали основные этапы поиска:

- Краулинг
- Синтаксический разбор

- Анализ
- Индексирование
- Поиск

Для полноты картины рассмотрим, что включает в себя каждый этап. Синтаксический разбор и анализ извлеченных документов – это то, что мы в совокупности называем *чисткой* (cleansing). Парсеры документов, которые мы описали в главе 2, необходимы для преобразования различных документов (таких как XML, HTML, Word, PDF) в стандартный чисто текстовый вид. Для разбора HTML-документов используется программный код из проекта NekoHTML (*http://nekohtml.sourceforge.net/*). Проект NekoHTML включает простой HTML-парсер, способный сканировать HTML-файлы, исправляя многие часто встречающиеся в HTML-документах ошибки. Возможности этого парсера включают добавление пропущенных скобочных элементов, автоматическое закрытие элементов необязательными концевыми тегами и обработку несовпадающих тегов элементов внутри строки. Парсер NekoHTML довольно надежен и быстр в работе, но если вы совершаете краулинг каких-то особенных сайтов, возможно, придется написать собственное средство синтаксического разбора.

Если вы планируете краулинг PDF-документов, можно использовать программный код из проекта PDFBox (*http://www.pdfbox.org/*). Релизы этого проекта сопровождаются программной лицензией университета Беркли (Berkley Software Distribution, BSD) и объемной документацией. Проект PDFBox включает класс `LucenePDFDocument`, позволяющий сразу получить Lucene-объект `Document` с помощью единственной строки кода, например:

```
Document doc = LucenePDFDocument.convertDocument(File file)
```

Дополнительную информацию ищите в комментариях Javadoc. Как и для PDF-документов, есть парсеры для документов текстового процессора Word. Apache-проект POI (*http://poi.apache.org/*) предоставляет API для манипулирования форматами файлов на основе формата Microsoft OLE 2 Compound Document исключительно средствами языка программирования Java. Кроме того, программный код TextMining, доступный по адресу *http://www.textmining.org/*[1], обеспечивает Java-библиотеку для извлечения текста из документов Microsoft Word 97/2000/XP/2003.

Этап анализа документов также важен. В этой книге мы используем Lucene-класс `StandardAnalyzer`, который помогает извлечь «смысл» из текста соответствующих документов. Безусловно, для успешного анализа документов крайне важны средства обработки данных на естественном языке (NLP). Краткое введение в методики NLP и многочисленные

[1] На момент подготовки книги данный сайт был недоступен, но вы можете ознакомиться с материалом здесь: *http://kickjava.com/src/org.textmining.text.extraction.index.htm*. – *Прим. науч. ред.*

ссылки на соответствующую литературу можно найти в *приложении D*. Задача NLP – помочь нам зафиксировать слова, наиболее релевантные и важные для описания контента документа, игнорируя все остальное. Если на этапе анализа проигнорировать что-то представляющее для вас интерес, вы уже не сможете использовать этот фрагмент потока информации при обработке своих данных. И наоборот, если запомнить массу нерелевантных или малозначащих слов, есть риск снизить тем самым качество обработки из-за высокого уровня зашумленности данных. При кластеризации или классификации новостных сообщений вы не сможете создать представительные группы новостей, а то и отнесете статьи о бизнесе к категории «Спорт»!

Алгоритмы кластеризации или классификации в принципе не могут удовлетворительно работать с «грязными» данными. Как автомобиль не может ехать на загрязненном горючем: даже если сама по себе машина хороша, она не поедет, если не заправить ее чистым топливом и не обеспечить необходимую для сгорания правильную пропорцию бензина и воздуха. Удалить «загрязнения» и добиться получения правильной «топливной смеси» для ваших алгоритмов поможет чистка (cleansing)!

Объем связанной с чисткой обработки данных, необходимой для конкретного веб-сайта, зависит от сложности структуры сайта. Современные веб-сайты насыщены графическими и аудиовизуальными элементами, контент которых гораздо труднее извлечь и обработать, чем содержимое «беспримесной» HTML-страницы или PDF-документа. Наш программный код и те инструменты, на которые мы ссылаемся, в основном демонстрируют достойную работу в применении к общей базе данных, собранных в Интернете.

Поисковый робот профессионального уровня должен уметь выяснить язык и даже точное местонахождение веб-сайта и выполнить чистку с применением наиболее подходящих инструментов. Запустив краулинг, можно оказаться где угодно. Начав со списка URL-адресов, указывающих на местонахождение в Соединенных Штатах, можно закончить в другом полушарии – в индийском Бангалоре или китайском Пекине. Поисковый робот должен суметь определить, как правильно «прочесть» такой контент.

Хотя есть стандартные методики идентификации географического местонахождения сервера, не исключено получение смешанного контента или сообщений на английском языке в контенте из Индии и Китая на веб-сайтах, которые находятся в Соединенных Штатах. Но наши классификаторы блеснут и здесь. Например, наивный байесовский алгоритм не зависит от языка, следовательно, его можно применять для идентификации языка конкретного документа или части документа. Это будет другой наивный байесовский классификатор, отличный от того, что применяется для классификации новостного сообщения: алгоритм один и тот же, а классификатор другой (например, `NBLanguageDetector`). Обучающий набор данных для этого другого классификатора

может включать простой разбор текста без анализа, а концептами классификации могут быть географические точки или языки. Мы не рассматриваем здесь этот пример, но вы должны учитывать, нет ли у вас текстов на нескольких языках. Подумайте о возможных проблемах с различными существующими в мире кодировками и о том, как это повлияло бы на правильность определения языка.

7.2.3. Используемый по умолчанию набор извлеченных и обработанных новостных сообщений

В оставшейся части главы мы задействуем набор веб-страниц, включающий страницы, с которыми мы работали в главе 2. Использование именно этого набора необходимо для того, чтобы конкретизировать примеры и обеспечить повторяемость упражнений. Указанные страницы можно найти в каталоге *data/ch07/*. Все они хранятся в подкаталоге *all* и имеют следующий состав (выбор контента случаен):

- 20 документов связаны с новостями бизнеса
- 24 документа – спортивные новости
- 23 документа имеют отношение к проблемам здоровья и медицине
- 26 документов содержат новости техники
- 16 документов относятся к политике США
- 20 документов содержат международные новости

Этот список говорит о том, что по умолчанию на нашем новостном портале основное внимание будет уделяться новостям из шести названных категорий; позже мы рассмотрим, как можно добавить к ним пользовательские категории. Так что давайте загрузим используемый по умолчанию набор данных и посмотрим, что собой представляют эти новостные сообщения. В листинге 7.2 показано, как загрузить набор документов, похожих на те, что были использованы в главе 2. Мы предоставляем две реализации интерфейса `NewsDataset`. Первая реализация – класс `CrawlResultsNewsDataset`, который можно использовать для любых (ну хорошо, почти любых) документов, извлеченных из Интернета с помощью класса `NewsCrawler` из листинга 7.1. Вторая реализация – это класс `FileListNewsDataset`, приведенный в листинге 7.2. Главное отличие этих двух реализаций заключается в том, что в случае класса `FileListNewsDataset` новостная категория определяется по имени документа, а в случае класса `CrawlResultsNewsDataset` новостная категория, к которой принадлежит каждый документ, неизвестна.

Листинг 7.2. Загрузка используемых по умолчанию новостных сообщений и запуск специализированного броузера новостей

```
NewsDataset dataset = new FileListNewsDataset("DefaultDS");

dataset.setDocumentDir("C:/iWeb2/data/ch07/all");
```

Задание каталога с документами

```
dataset.setTopTerms(15);   ← Число сохраняемых термов

dataset.loadTopics();

dataset.loadStories();

NewsUI ui = new NewsUI(dataset);

NewsUI.createAndShowUI(ui);
```

Элементарный пользовательский интерфейс, показанный на рис. 7.1, обеспечивает быстрый доступ к содержимому новостных сообщений и помогает нам проанализировать результаты, получаемые по мере того, как мы постепенно наращиваем интеллектуальные способности. В левой панели видны шесть новостных категорий и соответствующие им новостные сообщения (в круглых скобках указано число сообщений по каждой теме). Если щелкнуть мышью на конкретном новостном сообщении, его содержимое отобразится в правой панели.

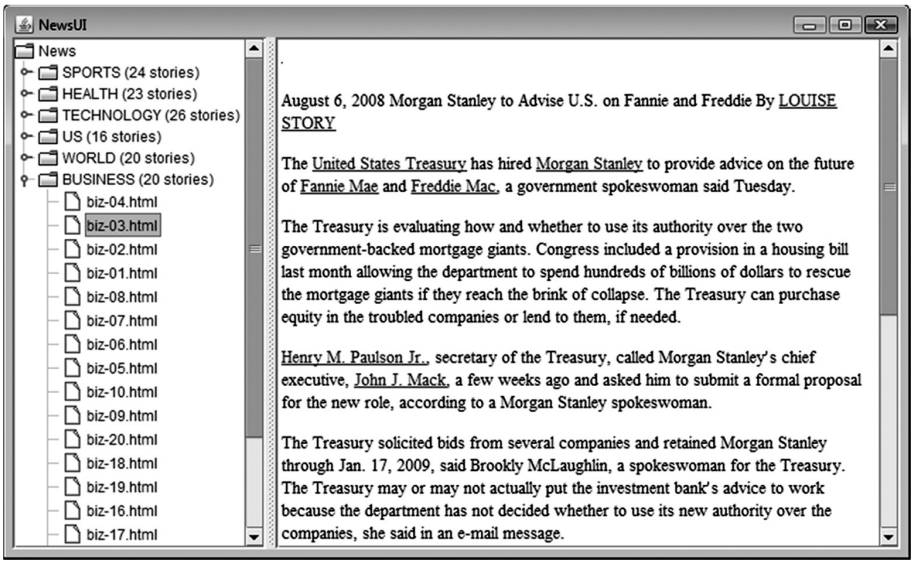

Рис. 7.1. *Специальный пользовательский интерфейс для просмотра используемых по умолчанию новостных сообщений*

Таким образом, внешне процесс сбора и очистки новостных сообщений может показаться прозаическим занятием, но это по-настоящему трудная работа. При широкомасштабном краулинге и очистке документов, полученных с разнообразнейших веб-сайтов, трудно добиться высоких показателей оперативности и эффективности. В таких случаях краулинг и чистка могут стать благодатной почвой для применения интеллектуальных алгоритмов. А теперь перейдем к индексированию и поиску новостных сообщений.

7.3. Поиск новостных сообщений

Как обычно, для индексирования и поиска документов мы используем библиотеку Lucene. Но всю функциональность, обеспечивающую обработку данных для нашего новостного портала, мы объединим в классе NewsProcessor. Программный код из листинга 7.3 обеспечивает создание экземпляра класса NewsProcessor, создание индекса для используемого по умолчанию набора данных, а также выполнение пары запросов. Предполагается, чтобы вы уже выполнили программный код из листинга 7.2.

Листинг 7.3. Индексация и поиск используемых по умолчанию новостных сообщений

```
NewsProcessor newsProcessor = new NewsProcessor(dataset);

newsProcessor.buildIndexDir();

newsProcessor.runIndexing();

newsProcessor.search("cell",5);

newsProcessor.search("football",5);
```

Индексирование и поиск были рассмотрены в главе 2, поэтому сейчас сосредоточимся в первую очередь на результатах и анализе различных элементов, которые могут превратить наш новостной портал в наилучшее место для его пользователей! Поиск слова «cell» обеспечивает результаты, показанные на рис. 7.2.

Заголовок документа – это метка для новостного сообщения, как показано в левой панели броузера новостей. Термы документа – это 15 наиболее часто встречающихся термов каждого документа, выявленные в результате нашего анализа. Оценка релевантности является Lucene-оценкой, полученной для каждого документа. В данный момент у вас должно быть открыто окно броузера новостей. Поэтому щелкнем на заголовке каждого документа в левой панели, чтобы изучить полные тексты статей. Первые два результата – это сообщения о влиянии сотовых телефонов на наше здоровье. Третий результат поиска относится к новостям техники: речь идет об использовании сотовых телефонов для обнаружения потерянных объектов. Четвертый и пятый результаты поиска никак не связаны с сотовыми телефонами. Это статьи о возможности подавления роста раковой опухоли за счет приема большой дозы витамина С.

Представленные результаты являются типичными: вы можете выполнить поиск на сайте Google News и получить набор новостей, относящихся к разным темам. В связи с этими результатами возникает ряд важных вопросов. Во-первых, когда пользователь ищет на новостном портале слово «cell», что именно он имеет в виду? Ищет ли он сведения об исследованиях в области стволовых клеток? Подразумевается ли не-

```
bsh % newsProcessor.search("cell",5);

Search results using Lucene index scores:
Query: cell

Document Title: health-21.html
Document Terms: warn, limit, phone, cell, children, us, risk,
herberman, ronald, becaus, especi, institut, brain, cancer, dr
Document URL: file:/C:/iWeb2/data/ch07/all/health-21.html -->
Relevance Score: 0.610166192054749
_____________________________________________________________
Document Title: health-23.html
Document Terms: phone,concern,cell,us,tumor,research,risk
herberman,pittsburgh,dont,publish,studi,brain,institut,cancer
Document URL: file:/C:/iWeb2/data/ch07/all/health-23.html -->
Relevance Score: 0.537154197692871
_____________________________________________________________
Document Title: tech-15.html
Document Terms: control, home, phone, can, your, cell, starner,
you, us, technolog, what, could, kientz, gestur, we
Document URL: file:/C:/iWeb2/data/ch07/all/tech-15.html   -->
Relevance Score: 0.341093242168427
_____________________________________________________________
Document Title: health-02.html
Document Terms: work, c, tumour, treatment, cell, inject, suggest
research, dose, bbc, human, mice, becaus, vitamin, cancer,
Document URL: file:/C:/iWeb2/data/ch07/all/health-02.html  -->
Relevance Score: 0.337665110826492
_____________________________________________________________
Document Title: health-09.html
Document Terms: telomeras, have, chemic, structur, cell, block
most, activ, bbc, help, new,  could,  immort,  cancer,  drug,
Document URL: file:/C:/iWeb2/data/ch07/all/health-09.html     -->
Relevance Score: 0.334201782941818
```

Рис. 7.2. *Результаты поиска слова «cell», выполненного только на основе индексирования*

кий технический гаджет? Может быть, нужны данные о влиянии электромагнитного излучения сотовых телефонов на людей? Понятно, что пытаться ответить на эти вопросы вне контекста бесполезно.

Можно ли воспользоваться алгоритмами PageRank или DocRank, которые мы изучили в главе 2? Да, безусловно. Эти алгоритмы ранжирования способны помочь нам улучшить результаты поиска, но только в минимальной степени. Главное преимущество алгоритмов ранжирования – это определение значимости сайта относительно прочих сайтов, которые являются частью нашего графа обхода (гиперссылок). Но

подавляющая часть внешних ссылок в новостных выпусках являются ссылками «на самого себя» или никуда не ведут (например, новостные отчеты в формате PDF, краткие анонсы новостей и так далее), в главе 2 они называются висячими узлами.

Не падайте духом! Мы еще многое можем сделать, чтобы улучшить взаимодействие пользователя с нашим порталом. Другая методика, которую мы изучали в главе 2, обращалась к данным о переходах пользователей по ссылкам и не исключено, что эти данные больше связаны с нашим запросом о возможностях улучшения результатов поиска в случае новостного портала. Очевидно, что в данном случае речь не идет о получении дивидендов за привлечение новых пользователей, но в долгосрочной перспективе использование данных о переходах пользователей по ссылкам может обернуться крупным вознаграждением. Как мы уже говорили, релевантность – вещь субъективная, и это одна из основных причин, по которым ее трудно оценить. Если мы с вами ищем результаты для запроса «cell», то, возможно, вас интересуют исследования стволовых клеток, тогда как меня может интересовать платформа Android или новейший гаджет фирмы Nokia. Таким образом, результаты, наиболее релевантные для одного пользователя, могут отличаться и довольно часто отличаются от результатов, наиболее релевантных для другого, даже при идентичных термах запроса!

Обратите внимание: в случае нашего новостного портала мы можем применить данные о переходах пользователей по ссылкам на двух уровнях. Первый уровень – отдельные новостные сообщения, к которым переходит пользователь, что аналогично варианту применения этой методики, рассмотренному в главе 2. Второй уровень информации – тема, которой принадлежит каждая новостная статья. К тому же мы, скорее всего, улучшим результаты поиска, сохраняя как термы запроса, так и термы документа в списке атрибутов для обучения (см. п. 2 раздела «Сделать» этой главы).

Приведенные примеры напоминают также о важности методов NLP для поиска. В частности, нам хотелось бы упомянуть три высокоуровневых элемента естественных языков, не доступных непосредственно из грамматических или синтаксических структур низкого уровня. Речь идет о *семантике*, *прагматике* и *дискурсе*. Семантика (знание о смысле) служит для применения наших знаний о мире. Прагматика связана с нашим знанием о взаимосвязи между смыслом слов и целями или намерениями пользователя. Дискурс относится к знанию, охватывающему более крупные лексические структуры, чем отдельные выражения или предложения. Последнее понятие особенно важно для результатов поиска, и его универсальность выходит за рамки обработки языков. Это то, что мы называем «общей картиной» или «общим контекстом» (см. также ссылки на литературу, приведенные в *приложении D*).

Значение интеллектуального поиска трудно переоценить. Но извлечение релевантных новостных сообщений в результате поиска – не един-

ственный способ, позволяющий предоставить нашим пользователям свежие новости. В сущности, стандартная разметка новостного портала отражает базовую схему классификации – организацию страницы с точки зрения составляющих ее содержимое разнообразных новостных категорий. Что ж, перейдем к изучению распределения новостных сообщений по новостным категориям.

7.4. Распределение по новостным категориям

К категории новостей можно отнести отдельное новостное сообщение или группу (кластер) таких сообщений. В этом разделе будут рассмотрены оба случая. Мы рассмотрели классификацию предметов в главе 5 и, классифицируя новости по категориям, естественно, полагаемся на представленные там алгоритмы. Но если проанализировать задачу классификации в более широком контексте приложения, включающего дополнительные возможности, такие как поиск, кластеризация и выработка рекомендаций, возникает много вопросов, которые не являются очевидными или релевантными, если рассматривать классификацию как обособленный процесс.

В качестве первой темы в этом разделе обсудим влияние очередности применения алгоритмов в приложении. После того как поисковый робот завершил свою работу, новостные сообщения извлечены, но не упорядочены и не структурированы. Одно из возможных направлений дальнейших действий – взять весь список только что собранных новостей и объединить их в группы. Другой возможный вариант: сначала классифицировать каждое новостное сообщение и отнести его к одной из 6 общих тем, а затем внутри каждой темы объединить похожие сообщения в группы. Как вы считаете, эквивалентны ли эти две возможности? Для общей массы новостных сообщений они, скорее всего, неэквивалентны. Говоря формальным языком, мы сказали бы, что эти две операции не обладают свойством коммутативности.

В разделах 7.4.2 и 7.4.3 подробно представлен специализированный классификатор новостных сообщений. Вы снова убедитесь в универсальности и ценности наивного байесовского алгоритма как классификатора общего назначения. Пожалуй, самая интересная часть нашего специализированного классификатора – это введение понятия *стратегии классификации*. Соответственно, в разделе 7.4.4 мы объясняем, что собой представляет стратегия классификации, зачем она нужна и как можно эффективно ее использовать.

Итак, приступим к распределению новостных сообщений по новостным категориям. Мы изучим две комбинации: классификацию результатов кластеризации и применение кластеризации к классифицированным новостным сообщениям.

7.4.1. Порядок имеет значение!

Мы уже намекали, что если изменить порядок операций, то будут получены разные результаты. Но насколько разные и какой порядок выполнения операций обеспечит лучшие результаты? Ответ на эти два вопроса будет зависеть как от природы данных, так и от применяемых алгоритмов. Тем не менее, выбор второго подхода (сначала общая классификация, а потом кластеризация) обычно аргументируют так: после классификации существенно снижается уровень зашумленности данных, следовательно, облегчается работа алгоритма кластеризации.

Посмотрите на рис. 7.3, где показаны результаты выполнения комбинации этих операций (классификации и кластеризации) применительно к набору новостных данных, используемому по умолчанию, когда кластеризация осуществляется перед проведением классификации.

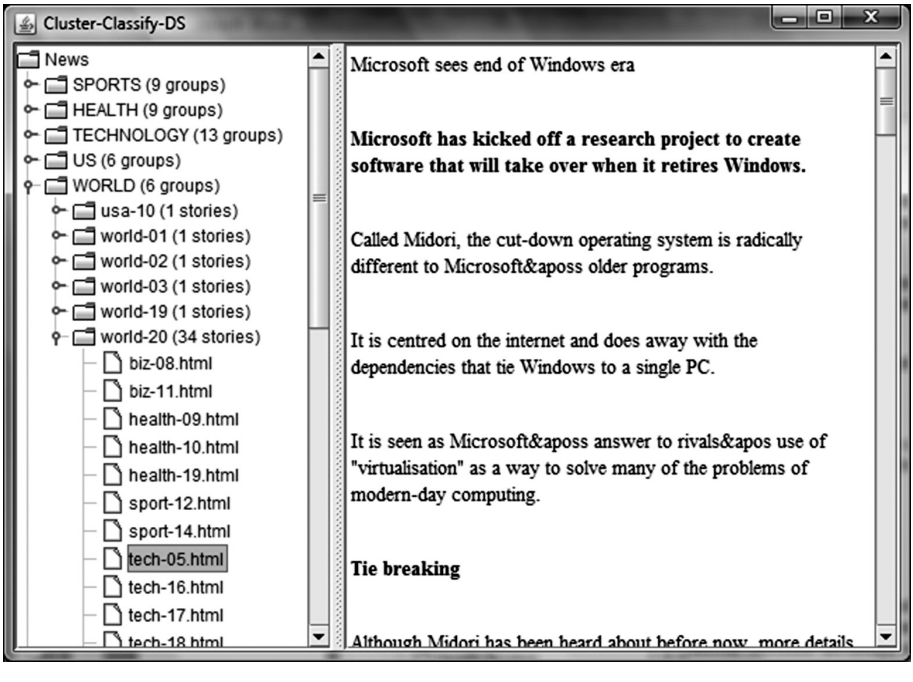

Рис. 7.3. *Сначала выполнена кластеризация новостных сообщений, затем их классификация по темам*

На рис. 7.3 выбранная статья (tech-05) – это новостное сообщение об исследовательском проекте, который инициировала фирма Microsoft, чтобы заменить действующее семейство своих операционных систем Windows. Понятно, что это новостное сообщение относится к новостям

техники, но оно попало в общую категорию международных новостей. В частности, данное сообщение попало в одну группу с сообщением из категории международных новостей о восстановлении Россией своих позиций как политической и экономической державы, то есть эти два сообщения по содержанию так далеки друг от друга, что дальше уже некуда – ну, разве что у вас очень живое воображение!

В этой же группе под заголовком статьи о возрождении России мы обнаруживаем изрядное количество статей, не имеющих ни малейшего отношения к России и даже не принадлежащих категории международных новостей, как статья tech-05. Всего в эту группу попало 34 новости, при 129 сообщениях в наборе данных, используемом по умолчанию. Эти сообщения разбиты по следующим категориям: 20 сообщений – новости бизнеса, 23 – новости здравохранения, 24 – спортивные новости, 26 – новости техники, 16 – политические новости США, 20 – международные новости. Таким образом, группа, включающая 34 сообщения, явно нарушает общий баланс, и если не настроить наш алгоритм кластеризации точнее, появление таких групп вполне вероятно.

В листинге 7.4 показано, как создать эти новостные группы и вывести их на экран в специализированном броузере. Можете прямо сейчас выполнить этот сценарий и внимательно рассмотреть структуру созданных групп и распределение новостных сообщений по соответствующим темам.

Листинг 7.4. Создание групп новостей путем их кластеризации перед проведением классификации

```
NewsDataset trainingDS =
➥   new FileListNewsDataset("TrainingDS");
trainingDS.setDocumentDir("C:/iWeb2/data/ch07/training");
trainingDS.init();

NewsDataset ds1 = new FileListNewsDataset("Cluster-Classify-DS");
ds1.setDocumentDir("C:/iWeb2/data/ch07/all");
ds1.init();

NewsProcessor newsProcessor = new NewsProcessor(trainingDS);

newsProcessor.trainClassifier();

newsProcessor.createClusters(ds1);

newsProcessor.classifyClusters(ds1);

NewsUI ui1 = new NewsUI(ds1);
NewsUI.createAndShowUI(ui1);
```

Создание наборов данных для обучения и тестирования

Кластеризация новостных сообщений

Классификация новостных сообщений, объединенных в группы

Мы еще вернемся к рис. 7.3 и более подробно рассмотрим структуру, созданную с применением программного кода из листинга 7.4. А сейчас, взглянем на структуру, полученную при обратном порядке выполнения операций: кластеризация выполняется *после* классификации.

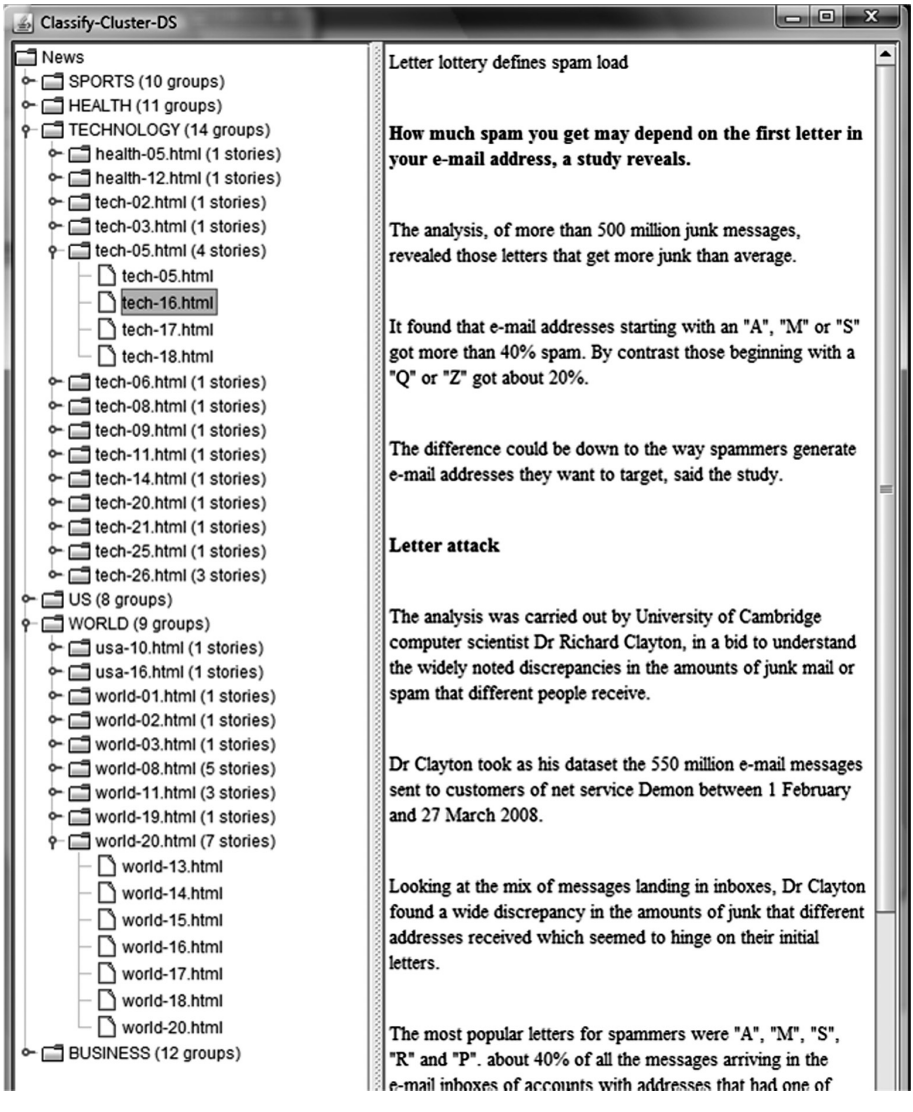

Рис. 7.4. *Сначала выполнена классификация новостных сообщений по темам, затем их кластеризация*

На рис. 7.4 показаны новостные группы, образованные из того же набора данных, что и на рис. 7.3, но кластеризации предшествовала классификация.

Если сравнить рис. 7.3 и 7.4, выясняется несколько вещей. На рис. 7.4 сообщения из категории новостей техники теперь попали в одну группу и (правильно) отнесены к соответствующей категории. Все новостные

сообщения, попавшие в одну группу с сообщением world-20, связаны между собой и относятся к возрождению России. Более того, в этой группе теперь отсутствуют новости, относящиеся к бизнесу и здравоохранению. Такая классификация новостных сообщений, разумеется, лучше предыдущей, но несколько случайностей все-таки присутствуют. Технические статьи tech-16, tech-17 и tech-18 (все они относятся к теме спама в электронной почте) правильно объединены в одну группу. Но новостное сообщение tech-05, относящееся к фирме Microsoft, не имеет к ним прямого отношения и не должно находиться в одной группе с этими новостями.

В листинге 7.5 показано, как создать эти новостные группы и отобразить их на экране в нашем специализированном броузере. Программный код этого листинга в основном идентичен коду из листинга 7.4, но есть одно важное отличие. Как и обещали, мы добавили сюда программный код, который изменяет порядок выполнения операций кластеризации и классификации на обратный. Естественно, изменились также имена инициируемых нами методов, поскольку теперь проводится классификация каждого отдельного сообщения (а не целых кластеров), а кластеризация выполняется только в пределах каждой темы. Мы используем новый экземпляр класса `FileListNewsDataset` с именем `ds2` и таким образом получаем возможность оперировать идентичной копией данных, но на экран выводим данные по отдельности. Если использовать тот же самый экземпляр (`ds1`), нельзя будет сравнить два рассматриваемых подхода! Если вы не вышли из сеанса командной оболочки, в котором выполняли код листинга 7.4, можете начать вводить команды со строки, которая позволяет создать экземпляр `ds2` класса `NewsDataset`.

Листинг 7.5. Создание групп новостей путем кластеризации, которая выполняется после классификации

```
NewsDataset trainingDS = new FileListNewsDataset("TrainingDS");
trainingDS.setDocumentDir("C:/iWeb2/data/ch07/training");
trainingDS.init();

NewsDataset ds1 = new FileListNewsDataset("Cluster-Classify-DS");
ds1.setDocumentDir("C:/iWeb2/data/ch07/all");
ds1.init();

NewsProcessor newsProcessor = new NewsProcessor(trainingDS);

newsProcessor.trainClassifier();

newsProcessor.createClusters(ds1);

newsProcessor.classifyClusters(ds1);

NewsUI ui1 = new NewsUI(ds1);
NewsUI.createAndShowUI(ui1);

NewsDataset ds2 = new FileListNewsDataset("Classify-Cluster-DS");
```

```
ds2.setDocumentDir("C:/iWeb2/data/ch07/all");
ds2.init();

newsProcessor.classifyStories(ds2);          ← Классификация всех новостных сообщений

newsProcessor.createClustersWithinTopics(ds2);   ← Кластеризация новостных
                                                   сообщений внутри каждой темы

NewsUI ui2 = new NewsUI(ds2);
NewsUI.createAndShowUI(ui2);
```

Итак, по-прежнему, должны быть видны оба Swing-клиента, – `ui1` и `ui2`. Измените размеры окон, показанных на рис. 7.3 (Cluster-Classify-DS) и 7.4 (Classify-Cluster-DS), и разместите их рядом друг с другом, чтобы проанализировать структуру групп и способ распределения новостных сообщений в случае применения каждого метода. На рис. 7.5 виден только первый уровень иерархии узлов, поскольку окна уменьшены; у себя на экране вы можете развернуть окна броузеров, так чтобы одно окно занимало всю левую часть экрана, а другое – всю правую.

В обоих случаях операции кластеризации и классификации выполнялись применительно к одним и тем же данным, но два разных варианта очередности выполнения этих операций привели к созданию двух разных структур, показанных на рис. 7.5. Сопоставив эти структуры, видим, что порядок выполнения операций «кластеризация-классификация» дает в результате меньшее число крупных кластеров, а «классификация-кластеризация» – большее число довольно небольших кластеров, корректных с точки зрения включенных в них новостей.

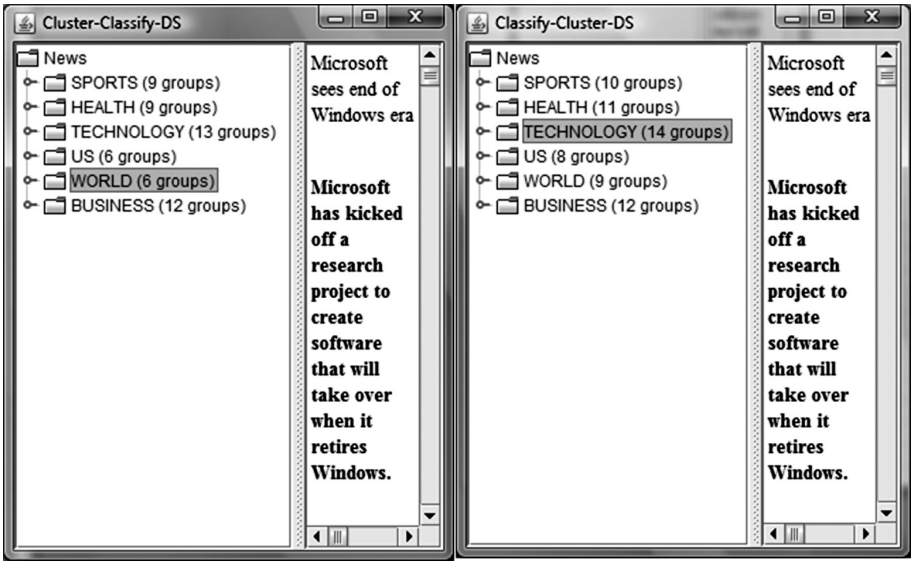

Рис. 7.5. *Сопоставление двух разных древовидных структур, показанных на рис. 7.3 и 7.4*

Заглянув в категорию международных новостей, мы обнаруживаем наибольшее расхождение в количестве созданных групп; при подходе «кластеризация-классификация» образованы 6 групп, а при подходе «классификация-кластеризация» – 9 групп. Новостное сообщение usa-10 было неправильно классифицировано в обоих случаях и попало в «чистую» группу, не включающую другие новостные сообщения.

Заметим, что оба метода мы применяли к расширенному набору данных для обучения. Из этого следует, что текст упомянутого новостного сообщения не соотносится ни с одной другой новостью из нашего набора. Это также свидетельствует о том, что его контент намного ближе к контенту обучающих документов из категории международных новостей, чем к контенту любого другого документа из оставшихся новостных категорий. Значит, следует включить этот образец (новостное сообщение usa-10) в обучающий набор данных! Именно такого рода анализ, или гарантия качества, если хотите, приводит к увеличению объема знаний в нашей системе.

Группа новостей usa-16 также была классифицирована неверно в случае подхода «классификация-кластеризация», но при порядке выполнения «кластеризация-классификация» эта группа не была даже сформирована. Если вы заглянете в группу с меткой world-20, то обнаружите там – в одной группе с 33 другими новостями – сообщение usa-16. Что произошло? Используя подход «кластеризация-классификация», мы начали кластеризацию со списка новостных сообщений, содержание которых было очень разнообразным. В результате большие группы, такие как world-20, оказались устойчивыми, и алгоритму кластеризации не удалось разбить их на меньшие фрагменты; разумеется, путем тонкой настройки параметров вы можете влиять на уровень детализации, но главная проблема не решается. При подходе «классификация-кластеризация» первичная классификация приводит в результате к разделению документов, что намного облегчает работу алгоритма кластеризации. Таким образом, при значительном снижении уровня шума среди документов, попавших в категорию международных новостей, алгоритм кластеризации смог определить, что новостное сообщение usa-16 не принадлежит ни к одному другому кластеру и, как и новость usa-10, его следует поместить в свою, отдельную группу. В связи с этим мы должны напомнить, что кластеры, состоящие из единственного члена, часто называют *синглтонами* (singletons), что вам, профессионалам в области программного обеспечения, должно быть знакомо по известному паттерну с таким же названием.

7.4.2. Классификация с помощью класса NewsProcessor

Давайте подробнее рассмотрим внутренние механизмы классификации с помощью класса `NewsProcessor`. Для описания каждого важного этапа будем использовать особый, только для этого этапа, набор листингов. Начнем с этапа обучения класса `NewsProcessor`, приведенного в листин-

ге 7.6. Этот этап состоит в обучении классификатора, поскольку операция кластеризации не подразумевает обучение, но заметим, что этот этап не сводится только к определению классификатора и вызову метода `train`.

Листинг 7.6. Обучение наивного байесовского классификатора и выбор стратегии классификации

```
public void trainClassifier() {

    if( topicSelector == null ) {

      NBStoryClassifier storyClassifier =
  new NBStoryClassifier("NewsStoryClassifier",trainingDataset);

      storyClassifier.train();

      ClassificationStrategyImpl defaultTopicSelector =
  new ClassificationStrategyImpl();   ← Выбор стратегии классификации

      defaultTopicSelector
➥    .setStoryClassifier(storyClassifier);   ← Назначение классификатора
                                                для выбранной стратегии
      topicSelector = defaultTopicSelector;
    }
 }
```

Имя класса `NBStoryClassifier` намекает на то, чего вы, скорее всего, и ожидали. Мы собираемся задействовать мощь наивного байесовского классификатора, с которым познакомились в главе 2 и вновь встретились в главе 5. В классе `NBStoryClassifier` реализовано ядро интерфейса `Classifier`, как должно быть у всех классификаторов, и содержится механизм, который будет «переводить» данные конкретного новостного сообщения на «язык», используемый классификатором `NaiveBayes`, — в классы `Concept` (концепт) и `Instance` (образец).

Рассмотрим класс `NBStoryClassifier`, но сначала обсудим выбор стратегии классификации, который присутствует в коде из листинга 7.6. Что делает этот класс `ClassificationStrategyImpl`? Ранее он нам не встречался. Почему нам понадобился этот класс? Потому что все примеры классификации, которые мы уже проработали в этой книге, относились к классификации списка образцов по соответствующим им концептам. Здесь, в примере с новостным порталом, нам приходится иметь дело с классификацией *групп образцов* (групп новостных сообщений) по соответствующим им концептам (категориям новостей). Это можно сделать несколькими способами, поэтому нам нужна стратегия, позволяющая решить, как именно мы будем это делать.

Одна стратегия могла бы состоять в том, что выбирается и классифицируется самая «репрезентативная» новость в группе, а затем каждый раз, когда в группу попадает эта репрезентативная новость, ее категория назначается всей группе; аналогично тому, как работает коллегия

выборщиков в США. Другой стратегией была бы классификация всех новостей в сочетании с принципом *принятия решения большинством голосов* (majority vote). Прежде чем взглянуть на программный код, попробуйте придумать другие возможные стратегии, взвесив отличия, которые каждая из них может внести в окончательную структуру новостных сообщений.

Итак, процесс классификации для нашего класса `NewsProcessor` не так прост как то, что мы видели до сих пор. Его обеспечивают два класса – `NBStoryClassifier` и `ClassificationStrategyImpl`. Первый класс позволяет использовать наивный байесовский классификатор, а второй отвечает за инкапсуляцию стратегии принятия решения, которой мы хотим следовать, классифицируя группу новостей.

7.4.3. Знакомьтесь: классификатор

Чтобы погрузиться в программный код еще на один уровень, подробнее рассмотрим каждый из указанных выше двух классов. Код класса `NBStoryClassifier` приведен в листинге 7.7. Как обычно, мы удалили из него некоторые методы, комментарии Javadoc и так далее. Помимо важнейших методов `train` и `classify`, которые должен реализовать каждый классификатор `Classifier`, здесь есть ряд вспомогательных методов, отвечающих за создание объекта `Instance` (образец) для новостного сообщения и объекта `Concept` (концепт) для новостной категории `Category`.

Листинг 7.7. Классификатор новостных сообщений, использующий наивный байесовский алгоритм

```
public class NBStoryClassifier implements Classifier {

    private List<ClassificationResult> conceptScores;
    private NaiveBayes nbClassifier;
    private boolean verbose = true;

    public NBStoryClassifier(String name, NewsDataset ds) {

        TrainingSet tSet = createTrainingSet(ds);

        nbClassifier = new NaiveBayes(name, tSet);
    }

    public TrainingSet createTrainingSet(NewsDataset ds) {

        int nStories = ds.getSize();
        List<Instance> allTrainingInstances =
          new ArrayList<Instance>(nStories);

        Iterator<NewsStory> iter = ds.getIteratorOverStories();

        while( iter.hasNext() ) {
            NewsStory newsStory = iter.next();

            Instance instance = toInstance(newsStory);
            allTrainingInstances.add(instance);
```

❶

❷

```
        }

        Instance[] instances =
            allTrainingInstances.toArray(new Instance[nStories]);

        return new TrainingSet(instances);
    }

    public boolean train() {
        TrainingSet tSet = nbClassifier.getTset();
        for(String attributeName : tSet.getAttributeNameSet() ) {
            nbClassifier.trainOnAttribute(attributeName);
        }
        return nbClassifier.train();
    }

    public Instance toInstance(NewsStory newsStory) {  ❷

        Concept concept              = toConcept(newsStory.getTopic());
        String[]               terms = newsStory.getTopNTerms();
        StringAttribute[] attributes = new StringAttribute[terms.length];

        for(int i = 0; i < terms.length; i++) {
            String name = terms[i];
            String value = "Y";
            attributes[i]= new StringAttribute(name, value);
        }
        return new BaseInstance(concept, attributes);                    ❸
    }

    private Map<String, NewsCategory> allTopics =
    new HashMap<String, NewsCategory>();

    private Map<String, Concept> allConcepts =
    new HashMap<String, Concept>();

    public Concept toConcept(NewsCategory t) {

        if( t == null ) return null;

        String topicName = t.getName();
        Concept c = allConcepts.get(topicName);
        if( c == null ) {
            c = new BaseConcept(t.getName());
            allConcepts.put(topicName, c);
        }
        return c;
    }

    public Concept classify(Instance instance) {  ❹

        conceptScores       = new ArrayList<ClassificationResult>();
        Concept bestConcept = null;
        double        bestP = 0.0;
        TrainingSet    tSet = nbClassifier.getTset();
```

```
        for (Concept c : tSet.getConceptSet()) {
            double p = nbClassifier.getProbability(c, instance);

            ClassificationResult cR = new ClassificationResult(c, p);
            conceptScores.add(cR);

            if( p >= bestP ) {
                bestConcept = c;
                bestP = p;
            }
        }
        return bestConcept;
    }
}
```

❶ Метод `createTrainingSet` — это главная обязанность класса `NBStoryClassifier`. Мы намерены делегировать задачу классификации нашей стандартной реализации `NaiveBayes`, но для этого нам нужен объект `TrainingSet`.

❷ Метод `toInstance` получает объект `NewsStory` в качестве входного параметра и создает объект `BaseInstance`. Чтобы создать объект `BaseInstance`, необходимы две вещи: объект `Concept`, которому принадлежит образец `Instance`, а также список атрибутов и их значений. Число атрибутов зависит от числа наиболее часто встречающихся в новостном сообщении термов. По умолчанию это значение (задается равным 25) определяется в классе `FileListNewsDataset`. Мы неявно использовали это значение в листинге 7.5, когда создали набор данных для обучения и два набора данных для тестирования (`ds1` и `ds2`). Изменение числа наиболее часто встречающихся термов приведет к изменению результатов классификации. Непосредственно перед тем, как обратиться к методу `init` набора данных, воспользуйтесь методом `setTopTerms`, чтобы изменить это значение, и посмотрите, как изменятся результаты в сравнении с полученными ранее.

Как вариант, можно было бы изменить реализацию метода `toInstance`, так чтобы он не обучал классификатор на всех часто встречающихся термах. Это упражнение описано также в п. 3 раздела «Сделать» этой главы.

❸ Метод `toConcept` — это второй важный вспомогательный метод для формирования обучающего набора данных, необходимый для метода `toInstance`. Логика реализации этого метода проста, здесь мало возможностей для изменений. Мы создаем объект `NewsCategory` класса `BaseConcept`, запоминая все концепты, встретившиеся нам до сих пор.

❹ Сравнив реализацию этого метода `classify` с реализацией метода `classify` в классе `NaiveBayes`, вы увидите, что они почти идентичны. Отличие заключается в том, что здесь мы собираем классификационные рейтинговые оценки для всех концептов и запоминаем их в пере-

менной conceptScores, доступной до тех пор, пока в очередной раз не будет инициирован метод classify. Это даст нам дополнительные возможности позже, когда мы будем использовать класс ClassificationStrategyImpl для назначения новостной категории каждому сообщению или группе новостей.

Класс ClassificationStrategyImpl служит для фиксации определений двух возможных подходов к классификации новостных сообщений. Первый подход – это категоризация новостной группы. Второй подход – категоризация новостного сообщения. Давайте проанализируем работу этого класса.

7.4.4. Стратегия классификации: выход за пределы низкоуровневой категоризации

Класс ClassificationStrategyImpl реализует интерфейс ClassificationStrategy, который отражает два возможных подхода к классификации новостных сообщений. В частности, методы assignTopicToCluster и assignTopicToStory, соответственно, отвечают за инкапсуляцию этих подходов – как эквивалент новостной категории здесь используется понятие *темы* (topic). Без лишних слов давайте взглянем на программный код этого класса, представленный в листинге 7.8, чтобы понять, что именно он делает (в листинге приведен не весь исходный код класса).

Листинг 7.8. Класс ClassificationStrategyImpl: определение стратегии классификации

```
public class ClassificationStrategyImpl implements ClassificationStrategy {

    private NBStoryClassifier storyClassifier;

    public void assignTopicToCluster(NewsStoryGroup cluster) {  ❶

        List<NewsStory> newsStories = cluster.getStories();
        NewsStory      rpStory = selectRepresentativeStory(newsStories);  ❷

        NewsCategory bestTopic = selectBestMatchingTopic(rpStory);  ❸

        cluster.setTopic(bestTopic);  ❹
        cluster.setRepresentativeStory(rpStory);
    }

    public void assignTopicToStory(NewsStory newsStory) {  ❺

        Instance instance = storyClassifier.toInstance(newsStory);
        Concept concept = storyClassifier.classify(instance);

        NewsCategory bestTopic = storyClassifier.toTopic(concept);
        newsStory.setTopic(bestTopic);
    }

    private NewsStory selectRepresentativeStory(List<NewsStory> newsStories)
{
```

```
        return selectLongestStory(newsStories);
    }                                                                  ❻

    private NewsStory selectLongestStory(List<NewsStory> newsStories) {

        NewsStory representativeStory = null;
        int maxContentLength = 0;

        for(NewsStory newsStory : newsStories) {

            int storyContentLength = newsStory.getContent().getText().
length();

            if( storyContentLength > maxContentLength ) {

                maxContentLength = storyContentLength;

                representativeStory = newsStory;
            }
        }

        return representativeStory;
    }

    private NewsCategory selectBestMatchingTopic(NewsStory newsStory) {  ❸

        Instance instance = storyClassifier.toInstance(newsStory);
        Concept   concept = storyClassifier.classify(instance);

        return storyClassifier.toTopic(concept);
    }
}
```

❶ Метод `assignTopicToCluster` инкапсулирует шаги, необходимые для решения о том, к какой новостной категории следует отнести данную новостную группу. Эта конкретная реализация проста. Здесь одно новостное сообщение из группы идентифицируется как представитель группы, выполняется классификация этой репрезентативной новости и запоминаются оба значения – как новостная категория (тема), так и представитель кластера. В п. 4 раздела «Сделать» этой главы вам предлагается исследовать более сложные стратегии категоризации новостных групп.

❷ Выбор репрезентативной новости полагается исключительно на метод `selectLongestStory`, но этот выбор может быть и более изощренным; также см. п. 4 раздела «Сделать».

❸ Возможно, вас интересует, не следует ли нам использовать здесь метод `assignTopicToStory` вместо метода `selectBestMatchingTopic`. В конце концов, последний просто инициирует классификатор и не отличается от первого. Но эти два метода не должны быть одинаковыми, и вот почему мы явно вызываем другой метод.

❹ На этом шаге выполняется запись выбранной репрезентативной новости и назначенной ей соответствующей новостной категории.

❺ Как уже говорилось, данная реализация эквивалентна реализации метода selectBestMatchingTopic. Это работа класса NBStoryClassifier – классифицировать новостное сообщение по новостным категориям. Этот метод не слишком полезен, если он просто делегирует классификацию классу NBStoryClassifier; мы и сами могли бы использовать этот классификатор. Значение этого метода выясняется, когда результат классификации подвергается последующей обработке или когда оригинальное новостное сообщение обрабатывается перед тем, как мы передадим его классификатору.

❻ Метод selectLongestStory обеспечивает простую эвристику выявления репрезентативной новости для списка новостных сообщений. Такая новость выбирается по принципу: чем больше размер новостного сообщения, тем больше оно пересекается с каждым из остальных новостных сообщений. Не особо изобретательный подход, и мы могли бы его существенно улучшить с помощью материала предыдущих глав, особенно главы 4. Некоторые идеи, которые вы можете проанализировать, изложены в п. 4. раздела «Сделать».

Наш вывод: класс ClassificationStrategyImpl играет роль метаалгоритма. То есть это та конструкция, в которую можно внедрить бизнес-логику или сложную комбинацию методик, чтобы помочь сделать наилучший выбор исходя из того, что нам известно (данные) и что мы можем сделать (низкоуровневые или элементарные алгоритмы). В главе 6 мы изучили ряд методик, помогающих объединить классификаторы. Эти методики, которые также являются метаалгоритмами, можно было бы применить здесь наряду с другими эвристиками или бизнес-правилами, чтобы оптимизировать распределение новостных сообщений по категориям.

Вернемся к самому классу NewsProcessor. В листинге 7.9 показано, как в двух его методах классификации – classifyClusters и classifyStories – используется класс ClassificationStrategyImpl. Класс NBStoryClassifier в данном случае непосредственно не задействован; «фасадом» для всех работ, связанных с классификацией, является класс ClassificationStrategyImpl. В самом классе NewsProcessor классификация – и кластеров новостей, и новостных сообщений – сведена к единственной строке кода.

Листинг 7.9. Категоризация групп новостей и новостных сообщений

```
public void classifyClusters(NewsDataset ds) {

    List<NewsStoryGroup> clusters = ds.getStoryGroups();

    for(NewsStoryGroup cluster : clusters) {

        topicSelector.assignTopicToCluster(cluster);
    }
}
```

Классификация всех групп в указанном наборе данных

Передача классу ClassificationStrategyImpl

```
public void classifyStories(NewsDataset ds) {          ← Классификация всех сообщений
                                                         в указанном наборе данных
   Iterator<NewsStory> iter = ds.getIteratorOverStories();

   while(iter.hasNext()) {

      NewsStory newsStory = iter.next();

      topicSelector.assignTopicToStory(newsStory)      ← Передача классу
   }                                                     ClassificationStrategyImpl
}
```

Этот раздел включает подробный обзор классификации кластеров новостей и новостных сообщений. Мы показали, что для результатов классификации важна очередность ее выполнения. Мы также описали, каким образом более общий контекст классификации – классификация отдельных новостных сообщений или групп таких сообщений – позволяет расширить возможности, доступные нам для проектирования эффективного решения. В частности, естественным образом вводится понятие стратегии классификации, позволяющее применять высокоуровневые классификационные решения, когда мы классифицируем новостные группы. Теперь нужно более подробно описать, как мы формируем эти группы новостей, с самого начала.

7.5. Формирование групп новостей с помощью класса NewsProcessor

Мы описали кластеризацию в главе 4, так что вы не удивитесь, обнаружив, что мы полагаемся на эти алгоритмы при создании групп новостей. В листинге 7.5 нам встретились два основных метода для создания групп новостных сообщений. Первым был метод `createClusters`, а вторым – метод `createClustersWithinTopics`. На первый взгляд, судя по программному коду листинга 7.10, эти два метода полагаются на разные классы для выполнения операций кластеризации. Но оба эти метода связаны с алгоритмом ROCK, о котором мы говорили в разделе 4.5.1.

Листинг 7.10. Класс NewsProcessor: два метода кластеризации

```
public void createClusters(NewsDataset ds) {

    NewsClusterBuilder clusterBuilder =
➥     new NewsClusterBuilder();                  ← Применение класса NewsClusterBuilder

    clusterBuilder.setNewsDataset(ds);

    clusterBuilder.cluster();
}

 public void createClustersWithinTopics(NewsDataset ds) {

    TopicalNewsClusterBuilder clusterBuilder =
➥      new TopicalNewsClusterBuilder();          ← Применение класса TopicalNewsCluster
```

```
        clusterBuilder.setNewsDataset(ds);

        clusterBuilder.cluster();
    }
```

Если оба метода, в конечном счете, полагаются на алгоритм ROCK, почему мы создали два отдельных класса? Главное отличие между данными реализациями этих двух классов заключается в том, что первый подвергает кластеризации все новости из набора данных `NewsDataset`, а другой – все новости внутри каждой категории новостей `NewsCategory` из набора данных `NewsDataset`. По аналогии с рассмотренной в предыдущем разделе классификацией результаты кластеризации можно улучшить, организовав реализацию этих двух случаев по-разному. Отличие между этими двумя классами выяснится по мере дальнейших рассуждений. Начнем с рассмотрения создания кластеров без учета новостных категорий.

7.5.1. Кластеризация обычных новостных сообщений

Метод `createClusters`, примененный к используемому по умолчанию набору данных новостных сообщений, вырабатывает кластеры, показанные на рис. 7.6. Всего имеется 55 кластеров: на рисунке видна только часть кластеров, но все их можно видеть в окне Swing-клиента. В част-

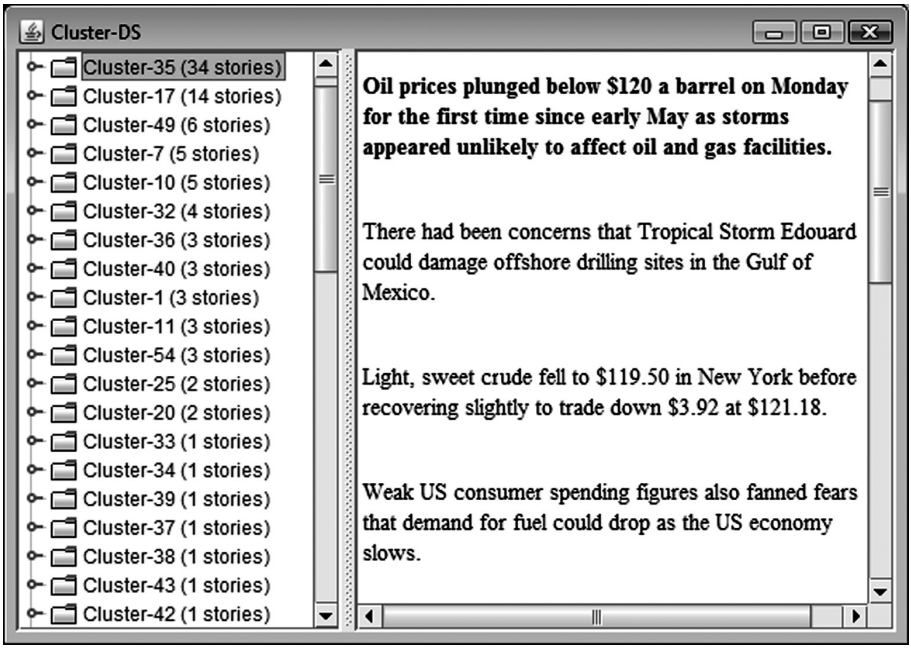

Рис. 7.6. *Результаты кластеризации используемого по умолчанию новостного набора данных, выполненной до классификации*

ности, есть 42 кластера-синглтона; 2 кластера содержат по 2 новости; 5 кластеров включают по 3 новости каждый; в 1 кластере – 4 новости; в 2 кластерах – по 5 новостей; 1 кластер содержит 6 новостей; есть 1 кластер с 14 новостями и 1 кластер с 34 новостями!

Это относительно важные числа. На обычном наборе новостных сообщений вы ожидали бы, что кластеры должны иметь приблизительно равный размер и что большинство сообщений скорее попали бы в кластеры, нежели остались в одиночестве (количество синглтонов составляло бы небольшой процент от общего числа кластеров). Если кластерное распределение известно, то мы можем вычислить производительность алгоритма кластеризации. Определение «идеального» кластера отчасти субъективно и зависит от того, каким, по-нашему, должен быть этот кластер. Например, взгляните на наш новостной набор данных, используемый по умолчанию: как бы вы объединили в кластеры эти сообщения? Чтобы еще больше конкретизировать и сократить набор новостей, рассматривайте все те сообщения, имена файлов которых начинаются с префикса *biz-*. Какие из этих новостей вы объединили бы в одну группу? Почему? Обсуждение этого важного вопроса вы найдете в п. 5 раздела «Сделать».

Итак, изучим результаты, показанные на рис. 7.6. Вы можете создать Swing-клиент, который содержит показанные на этом рисунке результаты, выполнив сценарий из листинга 7.11. Первая проблема кластеризации новостных сообщений – большой кластер, содержащий 34 сообщения. Наличие этого кластера проблематично, потому что нам известно, что 129 новостных сообщений более или менее равномерно распределены по 6 общим темам. Следовательно, этот кластер, занимающий 26% всего новостного набора данных, по размеру превышает самую большую из возможных групп при данном распределении сообщений по темам. Есть две серьезные причины полагать, что этот кластер слишком велик. Эту количественную, упрощенную оценку может подтвердить качественная оценка, которую можно провести, внимательно рассмотрев эти 34 новостных сообщения в Swing-клиенте. Аналогично, можно утверждать, что Cluster-17 с 14 новостями также слишком велик. Взгляните на эти новостные сообщения и еще раз прочтите раздел 4.5.1, чтобы освежить в памяти внутренние механизмы работы алгоритма ROCK.

Листинг 7.11. Кластеризация новостных сообщений без ссылки на категории новостей

```
NewsDataset ds = new FileListNewsDataset("Cluster-DS");
ds.setDocumentDir("C:/iWeb2/data/ch07/all");
ds.init();

NewsProcessor newsProcessor = new NewsProcessor();
newsProcessor.createClusters(ds);

NewsUI ui = new NewsUI(ds);
```

Загрузка используемого по умолчанию набора новостных данных

Кластеризация новостных сообщений

```
ui.showClustersOnly(true);   ← Вывод только кластеров
NewsUI.createAndShowUI(ui);
```

Похоже, принятые по умолчанию значения для числа кластеров и порога создания связи между двумя точками данных (новостными сообщениями) допускают создание *суперкластера*. Естественно, первым делом надо рассмотреть настройку этих двух параметров, которая выполняется в методе `cluster` класса `NewsClusterBuilder`, как показано в листинге 7.12.

Листинг 7.12. Метод кластеризации класса NewsClusterBuilder

```
public void cluster() {

   DataPoint[] dataPoints = createDataPoints(ds);

   int k = dataPoints.length / 3;   ← Необходимое число кластеров

   double linkThreshold = 0.15;   ← Порог для формирования связи

   ROCKAlgorithm rock = new ROCKAlgorithm(dataPoints, k, linkThreshold);

   Dendrogram dnd = rock.cluster();

   List<NewsStoryGroup> storyGroups = createStoryGroups(dnd);

   for(NewsStoryGroup cluster : storyGroups) {
       ds.addStoryGroup(cluster);
   }
}
```

Как видите, в этом методе число необходимых кластеров определяется как одна треть размера набора данных. Мы просим алгоритм создать приблизительно 43 кластера. В выборе этого числа нет ничего особенного. Поэкспериментируйте со значением `k` и понаблюдайте его влияние на самый большой из формируемых кластеров. Для этого можно заменить метод `cluster()` переопределяющим его методом `cluster(int k, double linkThreshold)`, который также можно найти в классе `NewsClusterBuilder`. Изменение значения переменной `linkThreshold` также повлияет на получаемые результаты, поэтому не следует менять эти значения одновременно. Изменяйте один параметр, сохраняя все остальное в прежнем виде.

Вторая проблема представленного на рис. 7.6 распределения кластеров – большое число синглтонов (кластеров, содержащих единственный элемент). Мы знаем, что это число не может быть таким большим, потому что эти новостные сообщения подбирались вручную! Как вы думаете, какая часть алгоритма ROCK оказывает наибольшее влияние на число синглтонов? Обращайте внимание не только на число нужных кластеров и параметр, определяющий порог связывания. Если вы еще не перечитали раздел 4.5.1, пожалуйста, сделайте это сейчас. Что если мы напишем другую реализацию класса `ROCKAlgorithm`? Что бы вы в ней

изменили? Дополнительные подсказки и анализ этого направления вы найдете в п. 6 раздела «Сделать».

Чтобы глубже вникнуть в результаты, сосредоточимся на новостных сообщениях, имена файлов которых начинаются с префикса *biz-*. На рис. 7.7 показаны три кластера, содержащих новости бизнеса. На этом рисунке видно, что сообщение из категории политических новостей США (новостные сообщения, имена файлов которых начинаются с префикса *usa-*) включено в третий кластер, где у нас есть две новости бизнеса. Оказывается, это сообщение включено в кластер правильно; щелкните мышью на каждой статье, чтобы убедиться в том, что эти новости связаны друг с другом. Создать кластер такого типа было бы непросто, если бы эти новостные сообщения сначала были распределены по категориям новостей. В этом случае новостное сообщение usa-01 могло быть отделено от сообщений biz-01 и biz-05 на этапе классификации.

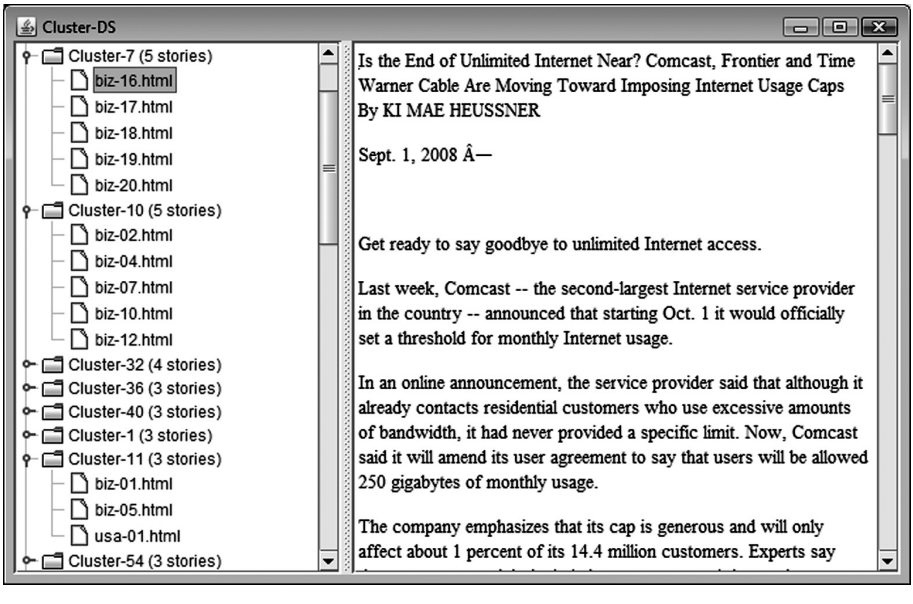

Рис. 7.7. *Просмотр трех кластеров с новостями бизнеса (в левой панели они развернуты)*

Это наблюдение приводит к интересной мысли по поводу интеллектуальных алгоритмов и возможных эффектов их объединения. Нет необходимости рассматривать эти два подхода к кластеризации как взаимоисключающие. Можно применить и тот и другой, получив выигрыш от каждого.

Если оказывается, что кластеризация новостных сообщений, выполненная *после* классификации, генерирует кластеры более высокого качества, из этого не следует, что применять эти операции в обрат-

ном порядке бесполезно. Выполнение кластеризации до классификации могло бы иметь смысл, например, в случае *перекрестных ссылок* (crossreferencing). Под перекрестными ссылками мы имеем в виду, что на веб-сайте может быть отдельная панель, где указаны связанные новости из других новостных категорий. Это не то же самое, что рекомендации, хотя их и можно предоставить конечному пользователю аналогичным способом, например: «Если вас заинтересовала эта новость, возможно, вам будут интересны следующие сообщения». Перекрестные ссылки имеют отношение к формированию базовой семантической сети для вашего набора данных. Надеемся, что это обеспечит вам быстрое знакомство с возможностями, которые открываются при использовании комбинации интеллектуальных алгоритмов. Одно только изучение очередности выполнения операций позволило нам естественным образом подойти к новой интересной функциональности!

Проверка кластеров Cluster-7 и Cluster-10 ведет к еще одной интересной мысли. Наши кластеры имеют одноуровневую структуру – внутри них нет других кластеров. Отчасти это объясняется замыслом, поскольку новостные сообщения на портале Google News представлены именно в таком формате. Тем не менее, можно усилить эффект путем создания иерархических взаимосвязей и групп новостных групп.

Сначала определим ценность такой возможности с практической точки зрения. В кластере Cluster-7 есть 5 сообщений, которые действительно связаны друг с другом. Но степень их релевантности неодинакова для всех новостных сообщений. Новостные сообщения biz-17 и biz-20 являются двумя разными отчетами об одном и том же событии – ограничении пропускной способности сети для некоторых пользователей провайдера Comcast. Аналогично, новостные сообщения biz-18 и biz-19 являются двумя разными отчетами о проводимом фирмой Time Warner Cable эксперименте по использованию Интернета. Последняя новость в кластере Cluster-7 – biz-16 – более общая по характеру, но все-таки непосредственно связана с обоими событиями, о которых говорится в остальных сообщениях.

Человек мог бы распределить эти новостные сообщения иначе. Имеет смысл объединить эти новости «под одной крышей», но сгруппировать вместе новости biz-17 и biz-20, а также новости biz-18 и biz-19, оставив как единственную запись новостное сообщение biz-16 из-за его более общего по характеру описания. Такая структура была бы более убедительной и удобной для пользователя. Один из возможных способов, позволяющих этого добиться, состоит в том, чтобы применить алгоритм кластеризации многократно, последовательно.

Фактически, это даже не обязательно должен быть один и тот же алгоритм, и как правило, применяются разные алгоритмы. Обычная практика – применить сначала алгоритм k-средних, а затем алгоритм ROCK в отдельных k-кластерах. Такой подход обычно применяют, чтобы повысить качество кластеров и улучшить показатели производительно-

сти самой операции кластеризации. В п. 7 раздела «Сделать» этой главы мы предлагаем вам поэкспериментировать с этим подходом применительно к настоящему примеру.

7.5.2. Кластеризация новостных сообщений в категории новостей

Метод `createClustersWithinTopics`, примененный к используемому по умолчанию набору новостных сообщений, вырабатывает кластеры, показанные на рис. 7.8. Заметим, что теперь получены 64 кластера, а именно: 45 кластеров являются синглтонами; 4 кластера содержат по 2 новости; в 6 кластеров попали по 3 новости; в 2 кластерах по 4 новости; 3 кластера содержат по 5 сообщений; в 1 кластере – 6 новостей; в 1 кластере – 7 новостей; 1 кластер с 8 сообщениями и 1 кластер содержит 14 новостей, как в предыдущем разделе.

Имена кластеров теперь отражают категории новостей (темы), внутри которых сформирован каждый кластер. Такое распределение кластеров выглядит лучше прежнего; сравните его с результатами, показанными на рис. 7.6. Мы избавились от суперкластера, при этом число кластеров, содержащих 2, 3 и 4 сообщения, возросло. К сожалению, синглтонов также стало больше. В листинге 7.13 приведен сценарий на языке BeanShell, который позволяет получить результаты, показанные на рис. 7.8. Выполните этот сценарий сейчас, чтобы иметь под рукой структуру для справки, пока мы обсуждаем эти результаты. Логика

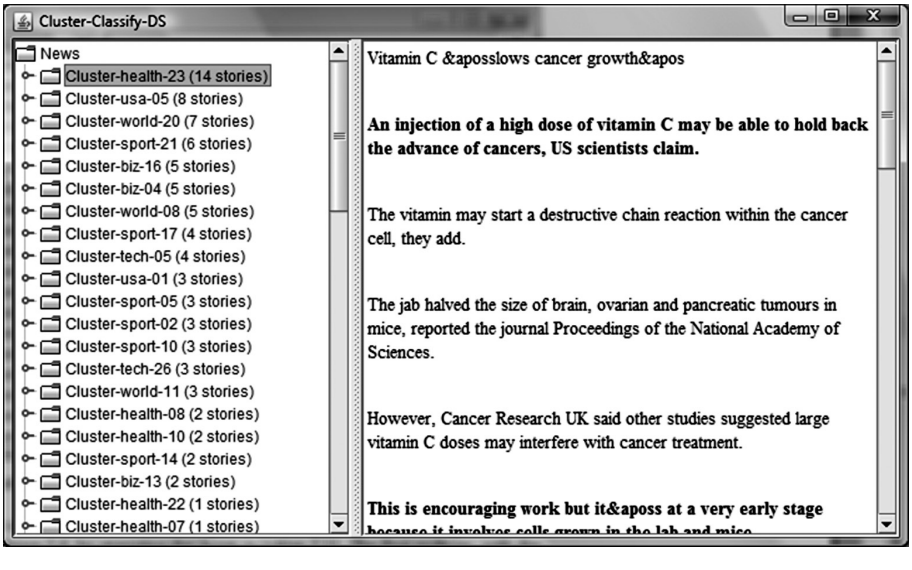

Рис. 7.8. *Результаты кластеризации используемого по умолчанию набора новостей, проведенной после классификации новостных сообщений*

представленных в этом листинге шагов проста и похожа на ту, что реализована в листинге 7.5.

Листинг 7.13. Кластеризация новостных сообщений, предварительно распределенных по новостным категориям

```
NewsDataset trainingDS = new FileListNewsDataset("TrainingDS");
trainingDS.setDocumentDir("C:/iWeb2/data/ch07/training");
trainingDS.init();

NewsDataset ds = new FileListNewsDataset("Classify-Cluster-DS");
ds.setDocumentDir("C:/iWeb2/data/ch07/all");
ds.init();

NewsProcessor newsProcessor = new NewsProcessor(trainingDS);

newsProcessor.trainClassifier();

newsProcessor.classifyStories(ds);

newsProcessor.createClustersWithinTopics(ds);

NewsUI ui = new NewsUI(ds);
ui.showClustersOnly(true);
NewsUI.createAndShowUI(ui);
```

Раскрыв самый большой кластер, Cluster-health-23, вы поймете, что его контент идентичен контенту кластера Cluster-17 (см. рис. 7.6). Совпадает не только число новостных сообщений, совпадают сами новостные сообщения, принадлежащие этим двум кластерам, как показано на рис. 7.9. На этом кластере очередность выполнения операций никак не отразилась.

Неизменность этого кластера важна и может иметь следующее объяснение. Ни сообщение tech-04, ни сообщение tech-15 не принадлежат к обучающему набору данных классификатора, вот классификатор и затруднился назначить им соответствующую новостную категорию. Новостное сообщение health-16 включено в обучающий набор данных, и его содержание связано с применением мобильных телефонов, что частично пересекается с контентом двух новостей техники. По-видимому, это пересечение более важно, чем пересечение новостных сообщений из обучающего набора данных, отнесенных к категории новостей техники. После того как эти две новости техники попали в неправильный кластер, самое большее, на что мы могли рассчитывать, – что они останутся синглтонами или попадут в один кластер (в неправильной новостной категории, разумеется).

Во всяком случае, содержание 14 новостных сообщений одинаково воспринимается как алгоритмом кластеризации, так и алгоритмом классификации. С практической точки зрения, в случае кластеризации, выполняемой после классификации, эту ситуацию можно исправить за счет расширения набора данных для обучения. Получив неправиль-

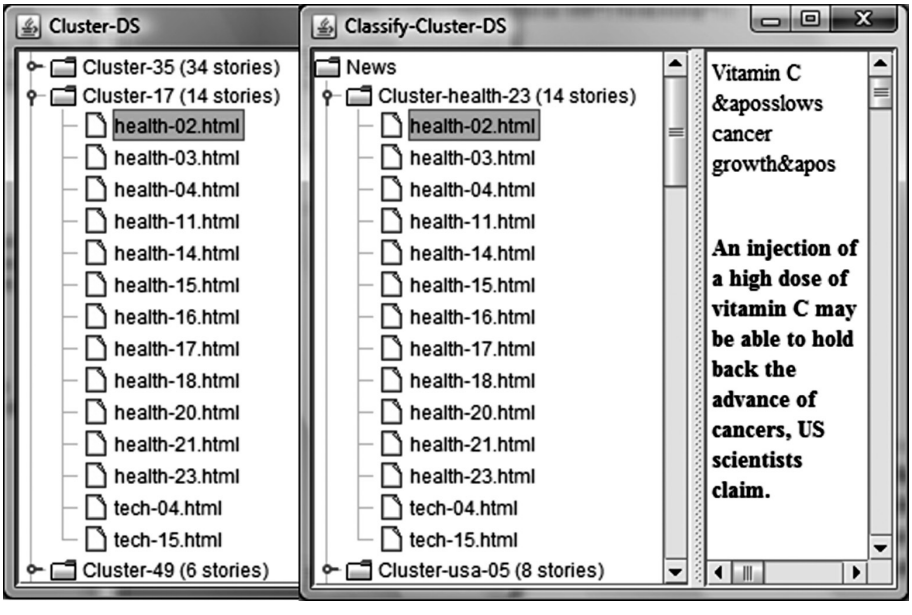

Рис. 7.9. *После изменения очередности выполнения кластеризации и классификации на обратную большой кластер не изменился. В крайней слева панели показаны результаты кластеризации, полученные в том случае, когда она выполнялась первой. В средней панели показаны результаты кластеризации, полученные в том случае, когда сначала выполнялась классификация*

но классифицированные элементы, нужно провести кластеризацию с большей детализацией в пределах темы.

Если рассмотреть результаты внимательнее, выясняется, что многие кластеры, сформированные внутри каждой новостной категории, идентичны кластерам, полученным, когда кластеризация выполнялась до классификации. Это значит, что самой большой выгодой оказалось разбиение суперкластера. Разумеется, нам следует проявить осмотрительность, не обобщая это открытие на произвольный набор новостных данных, но мы предполагаем, что оно подтвердится, если обучающий набор обеспечивает достаточное покрытие категорий новостей.

Рассмотрим настройку используемых по умолчанию значений для числа кластеров и порога создания связи между двумя точками данных (новостными сообщениями), выполняемую в методе `cluster` класса `TopicalNewsClusterBuilder`, см. листинг 7.14. Эти параметры могут изменить количество синглтонов в окончательных результатах. Их влияние сохраняется независимо от очередности выполнения операций кластеризации и классификации. Но изменение этих параметров должно ока-

зывать большее влияние, если кластеризация выполняется после классификации, потому что наборы точек данных, переданные в качестве входных параметров в алгоритм кластеризации, имеют пониженный уровень шума.

Листинг 7.14. Метод кластеризации класса TopicalNewsClusterBuilder

```
public void cluster() {

  for(NewsCategory topic : ds.getTopics() ) {

    List<NewsStory> stories = ds.getStories(topic);
    DataPoint[]  dataPoints = createDataPoints(stories);

    int k = dataPoints.length / 3;   ← Необходимое число кластеров

    double linkThreshold = 0.15;   ← Порог для формирования связи

    ROCKAlgorithm rock = new ROCKAlgorithm(dataPoints, k, linkThreshold);

    Dendrogram dnd = rock.cluster();

    List<NewsStoryGroup> storyGroups = createStoryGroups(topic, dnd);

    for(NewsStoryGroup cluster : storyGroups) {
       ds.addStoryGroup(cluster);
    }
  }
}
```

Листинг 7.14 напоминает листинг 7.12. Главное отличие – внешний цикл, определяющий новую область видимости для каждой новостной категории. Оправдывается ли этим различием в реализации существование двух отдельных классов для обработки кластеризации новостных сообщений? Как мы видели, текущая реализация предотвращает появление суперкластеров, но что еще важнее, она прокладывает путь для реализации, в которой число кластеров и пороговое значение для создания связи между двумя точками данных могут быть функцией новостной категории. Изучение открываемых этим подходом возможностей вы можете начать с чтения п. 8 раздела «Сделать».

Мы только что рассмотрели два самых важных способа кластеризации новостных сообщений – кластеризацию отдельных новостных сообщений и кластеризацию новостных сообщений внутри новостной категории. Мы завершили представление всех интеллектуальных алгоритмов, описанных в этой книге, кроме алгоритма выработки рекомендаций. Тема следующего раздела – применение на нашем новостном портале алгоритма выработки рекомендаций, основанных на сходстве предметов. Исследуется случай, когда у наших пользователей есть возможность оценивать новостные сообщения, а наша цель – с помощью этих оценок динамически упорядочивать контент портала для каждого пользователя.

7.6. Динамический контент на базе пользовательских оценок

Имея оценки новостных сообщений, мы можем рекомендовать новости, которые пользователь еще не видел. В главе 3 мы рассмотрели две главные категории методик, позволяющих вырабатывать рекомендации, – коллаборативную фильтрацию и подход, основанный на сходстве контента. В процессе построения нашей модели музыкального интернет-магазина была создана инфраструктура, позволяющая вам написать универсальную систему выработки рекомендаций для своего приложения. Пора проверить это утверждение. Создадим систему выработки рекомендаций новостных сообщений, взяв за основу программный код из главы 3. Если вам нужно перечитать главу 3 или вы ее еще не читали, сейчас самое время это сделать.

Сначала создадим файл, который содержит оценки, выставленные пользователями. Как вы, возможно, помните, элементарными концептами системы выработки рекомендаций являются предметы `Items`, пользователи `Users` и оценки `Ratings`. В этой главе новостные сообщения соответствуют предметам. Файл *ratings.txt* из каталога *C:\iWeb2\data\ch07\ratings* содержит данные, приведенные в табл. 7.1.

Таблица 7.1. Пятнадцать оценок, выставленных тремя пользователями новостного портала

Пользователь	Новостное сообщение	Оценка
A	Usa-04	4
	Usa-05	3
	Biz-05	5
	Biz-01	4
	World-16	5
B	Usa-01	5
	Sport-04	3
	Sport-03	4
	Biz-05	5
	Tech-06	2
C	World-15	3
	World-16	5
	Usa-04	4
	Biz-01	4
	Usa-01	5

Табл. 7.1 содержит 15 оценок, выставленных тремя пользователями (A, B и C). В разделе 7.5 нам постоянно встречался кластер, содержащий статьи biz-01, biz-05 и usa-01. Мы разработали оценки так, что новостное сообщения biz-01 читали пользователи A и C, а сообщение biz-05 – пользователи A и B. Судя по таблице, среди новостей, рекомендованных пользователю A, должно быть новостное сообщение usa-01. Выполним сценарий из листинга 7.15 и посмотрим, что получится!

Листинг 7.15. Выработка рекомендаций по новостным сообщениям на основе оценок

```
NewsDataset ds = new FileListNewsDataset("NewsDataset");
ds.setDocumentDir("C:/iWeb2/data/ch07/all");
ds.setTopTerms(25);
ds.setUserAndRatingsFilename("c:/iWeb2/data/ch07/ratings/ratings.txt");
ds.init();

StoryRecommender delphi = new StoryRecommender(ds);
delphi.calculateRecommendations();

delphi.recommendStories("1");
```

Создание системы выработки рекомендаций

Результаты показаны на рис. 7.10, и новостное сообщение usa-01 в самом деле является одной из рекомендуемых новостей. Возможно, это не производит особого впечатления в случае небольшого числа новостных сообщений, пользователей и оценок, используемых нами в рассматриваемом примере, но этот же программный код с удовлетворительными результатами можно применять для более крупных наборов данных. Обратите внимание: наш рекомендатель пропускает новостные сообщения, которые пользователь A уже оценил.

```
bsh % delphi.recommendStories("1");
Skipping item:biz-01.html
Skipping item:biz-05.html
Skipping item:usa-05.html
Skipping item:usa-04.html
Skipping item:world-16.html

Recommendations for user UserA:
Item: usa-01.html                  , predicted rating: 5.000000
Item: sport-03.html                , predicted rating: 4.000000
Item: world-15.html                , predicted rating: 3.000000
Item: sport-04.html                , predicted rating: 3.000000
Item: health-16.html               , predicted rating: 2.500000
bsh %
```

Рис. 7.10. *Рекомендации для пользователя A, основанные на оценках из табл. 7.1*

Заглянем в класс StoryRecommender, который обеспечивает получение этих рекомендаций. В листинге 7.16 приведен весь его исходный код. С помощью системы выработки рекомендаций Delphi мы создали специализированную систему выработки рекомендаций по новостным сообщениям, написав лишь несколько строк программного кода. Это тот же класс Delphi, с которым мы встречались в главе 3.

Листинг 7.16. Класс StoryRecommender: выработка рекомендаций новостных сообщений для пользователя

```
public class StoryRecommender {

  private DatasetAdapter rDs;
  private Recommender delphi;

  public StoryRecommender(NewsDataset ds) {
      this.rDs = new DatasetAdapter(ds);
  }

  public void calculateRecommendations() {

    Delphi d = new Delphi(rDs,
➥     RecommendationType.ITEM_PENALTY_BASED, false);   ← Создание системы выработки рекомендаций
    d.setVerbose(true);

    this.delphi = d;
  }

  public List<PredictedNewsStoryRating>
➥  recommendStories(String newsUserId) {

    if( delphi == null ) {
      String msg = "Recommender not initialized.";
      throw new RuntimeException(msg);
    }

    User user = rDs.getUserForNewsUserId(newsUserId);

    List<PredictedItemRating> predictedRatings = delphi.recommend(user);

    List<PredictedNewsStoryRating> ratings =
➥     new ArrayList<PredictedNewsStoryRating>();

    for(PredictedItemRating iR : predictedRatings) {

            PredictedNewsStoryRating r = new PredictedNewsStoryRating();
            r.setUserId(newsUserId);
            r.setRating(iR.getRating());
            NewsStory newsStory =
➥           rDs.getNewsStoryForItemId(iR.getItemId());
            r.setStoryId(newsStory.getId());

            ratings.add(r);
        }
```

```
            return ratings;
        }
    }
```

Мы не использовали ничего, кроме необходимого «перевода» терминологии, специфичной для новостных сообщений, в понятия универсального интерфейса наборов данных из главы 3, о которых заботится класс DatasetAdapter. Система выработки рекомендаций Delphi основана на определении предметного сходства со штрафами (тип ее рекомендаций – RecommendationType.ITEM_PENALTY_BASED). Вы можете экспериментировать с другими типами, доступными для нашей системы выработки рекомендаций.

Класс StoryRecommender и его вспомогательные классы, такие как DatasetAdapter и PredictedNewsStoryRating, могут служить шаблоном для вашего собственного приложения. Общая идея проста: преобразовать данные так, чтобы их можно было отобразить в предметы Items, пользователей Users и оценки Ratings. Затем нужно делегировать выработку рекомендаций системе Delphi; вы можете повозиться с доступными типами рекомендаций, выбрав тип, наиболее соответствующий вашей задаче. Ваши данные могут храниться и быть структурированы как-то иначе, но их преобразование в наши универсальные классы не должно вызвать затруднений.

7.7. Заключение

Эта глава демонстрирует применение интеллектуальных алгоритмов, изученных нами к этому моменту, в рамках веб-приложения, а именно в контексте новостного портала, такого как Google News. Мы убедились в важности применения интеллектуальных алгоритмов с самого начала – с возможности интеллектуального краулинга. Разумеется, материал главы 2 можно было также применить в рассмотренном примере новостного портала, обеспечив поиск на основе индексов и не только!

Возможно, в каждом конкретном приложении трудно сразу распознать место и значение интеллектуальных алгоритмов. Тем не менее, когда мы начинаем задавать вопросы, становится очевидным, что эти алгоритмы могут повысить уровень обслуживания пользователей приложения довольно неожиданными способами. Мы видели, что с помощью алгоритмов кластеризации можно не только объединять в группы похожие новостные сообщения, но и расширить видимость релевантных новостей путем предоставления *перекрестных ссылок*.

Кроме того, мы убедились в возможности многократного применения одного и того же алгоритма или последовательного – разных алгоритмов. Но, как оказалось, при последовательном применении имеет значение порядок. Мы продемонстрировали эту особенность, явно изменив порядок выполнения операций кластеризации и классификации но-

востных сообщений на обратный. Результаты кластеризации новостей, выполненной после их классификации, оказались лучше, чем результаты кластеризации, выполненной до классификации. Это наблюдение достаточно общего характера, скорее всего, проявится и на ваших данных, будь это новостные сообщения или что-либо еще.

В результате последовательного применения алгоритмов кластеризации и классификации открылся еще один интересный факт. Мы показали, как следует поступать с классификацией самих групп. До сих пор мы видели только классификацию простых объектов, таких как сообщения электронной почты или транзакции по кредитным картам. Здесь мы описали, как классифицировать новостное сообщение или группу таких сообщений. Это можно сделать несколькими способами, поэтому в таких случаях на практике может оказаться целесообразным построение метаалгоритма.

На любой вопрос по применению интеллектуальных алгоритмов практически невозможно дать ответ, подходящий для всех приложений. Тем не менее, мы надеемся, что обрисовали в общих чертах разумный подход к внедрению интеллектуальных алгоритмов. Мы также надеемся, что в этой главе дали понять следующее: для достижения конкретной цели можно создавать компоненты, объединяющие несколько интеллектуальных алгоритмов. Внимательно рассмотрите поднятые в тексте вопросы и несколько идей, которые мы предлагаем осуществить на практике в соответствующих пунктах раздела «Сделать». Не сомневаемся, что вы найдете их полезными для своей работы.

7.8. Сделать

1. *Добавление интеллектуальных возможностей в класс NewsCrawler.* Наш класс `NewsCrawler` хорош как отправная точка, позволяющая начать с небольшой по объему базы программного кода. Но если вы хотите просто извлечь контент, следуя по списку URL-адресов, то для этой цели найдется много других поисковых роботов, скорее всего, гораздо лучше нашего. Мы написали этот небольшой робот, желая особо выделить несколько областей, в которых интеллектуальные алгоритмы могут существенно повысить оперативность и эффективность краулинга. В этом упражнении мы хотим рассмотреть эти потенциальные улучшения, надеясь пробудить в вас достаточный интерес для их реализации!

   Рассмотрим случай веб-сайтов со спамом или неподходящим материалом – неподходящим вы можете объявить любой материал, какой хотите, вернее, не хотите получать! Речь идет о таких веб-сайтах, краулинг которых вас, вероятно, не интересует. Но даже если вы, допустим, намеренно не указали такие сайты как целевые, поисковый робот может случайно попасть на один из них. Следовательно, было бы неплохо, если бы робот обладал способностью определять подобные сайты и избегать их посещения. Как бы вы добились этого?

В случае электронной почты со спамом, над которым мы работали в главе 5, для фильтрации содержащих спам почтовых сообщений были использованы наивный байесовский классификатор и классификатор на основе правил. Так же можно поступить и здесь. Наш поисковый робот мог бы сделать выборку страниц и классифицировать их как подходящие или не подходящие для извлечения. Для наивного байесовского классификатора потребовалось бы получить образцы веб-страниц как из одной, так и из другой категории; эти образцы составили бы обучающий набор данных. Чтобы можно было применить классификатор на основе правил, потребуются какие-то правила. Например, если страница содержит слово *xyz*, ее надо пропустить.

Эти же алгоритмы можно применить в несколько иной ситуации. Если вы хотите указать определенные веб-страницы в качестве целевых (а не тех, что следует отбросить), можно создать робота, внимание которого будет сфокусировано на статьях, имеющих отношение к языку программирования Java или, более конкретно, к Java-проектам с открытым исходным кодом. Тут вы можете начать со списка известных URL-адресов или ссылок, полученных в результате поиска с помощью системы Google, но скачивать контент веб-сайтов только в том случае, если классификатор сообщит о том, что этот контент релевантен вашей тематике.

Вы можете надстроить интеллектуальную обработку поверх полученных результатов. Если ссылка с сайта A ведет на сайт B, а сайт B считается неподходящим, скорее всего, целесообразно подсчитать число ссылок на сайте A, ведущих к таким сайтам, как B. Само это соотношение можно было бы использовать как пороговое значение для построения графа ссылок, который можно проанализировать с помощью метода, аналогичного применяемому в алгоритме PageRank. Это лишь несколько идей, над которыми вы можете поработать. Если вы их реализовали и остались довольны полученными результатами и предоставленными возможностями, попробуйте предложить аналогичные идеи для встраивания интеллектуальных алгоритмов в процесс краулинга. Наградой за это должны стать приобретенные знания, а кроме того это может оказаться полезным в вашей работе.

2. *Уточнение результатов поиска на новостном портале за счет применения данных о переходах пользователей по ссылкам*. В главе 2 мы видели, что взаимодействие пользователя с поисковой системой определяется областями его или ее интересов и личной субъективностью. Мы ввели наивный байесовский классификатор, чтобы использовать данные о переходах пользователей по ссылкам и уточнить результаты поиска. Как вы, возможно, помните, мы исходили из предположения о том, что собрали данные о переходах пользователей по ссылкам и они хранятся в файле *user-clicks.csv* каталога *data/ch02*. Этот файл, в котором значения разделяются запятыми,

имеет три поля: идентификатор пользователя, термы поискового запроса и строку URL-адреса.

Аналогичным образом вы можете создать файл, содержащий данные о переходах пользователей по ссылкам, чтобы уточнить результаты поиска на нашем новостном портале. Какими теперь должны быть обучающие атрибуты? Попробуйте следующий рецепт. Выберите в качестве обучающих атрибутов:

- Идентификатор пользователя
- Термы поискового запроса
- Термы выбранного документа
- Тему выбранного документа

Можете ли вы обосновать этот выбор? Почему мы заменяем URL-адрес темой? Почему мы вводим термы документа? Как бы вы приступили к уточнению рейтинговых оценок? Запишите свои идеи и реализуйте их.

Файл с данными о переходах пользователей по ссылкам должен содержать несколько записей для одного и того же пользователя и одних и тех же термов запроса. В примере из главы 2 число встретившихся в этом файле переходов по конкретной ссылке определяло URL-адрес этой ссылки как лучшего кандидата на попадание в результаты нашего поиска. То же самое истинно для различных тем в случае новостного портала. Как правило, один и тот же пользователь, выполняя один и тот же запрос, прочтет несколько сообщений разной тематики, потому что со временем его интересы могли измениться или потому что он, возможно, ищет дополнительную информацию по конкретной теме.

В случае новостного портала интересной с точки зрения ее решения является следующая задача. Если у меня есть несколько новостей в одной категории и мой классификатор обучен идентифицировать эту тему, как это поможет мне уточнить список наиболее интересных сообщений для данного пользователя? Возможно ли это? Если нет, что можно сделать для решения этой задачи?

3. *Оценка влияния выбора атрибутов на результаты новостного портала.* Выбор атрибутов для новостного сообщения был основан на наиболее часто встречающихся термах, извлекаемых из данного сообщения. Из листинга 7.7 понятно, что каждый часто встречающийся терм становится обучающим атрибутом для классификатора. Разумеется, вариантов гораздо больше!

   Очевидное изменение метода `toInstance` из листинга 7.7 – создать единственный строковый атрибут, включающий весь набор часто встречающихся термов, разделенных, скажем, пробелами при их конкатенации, чтобы нам удобнее было читать значение этого атрибута. Как вы думаете, улучшит или ухудшит результаты классификации такое изменение? Каково бы ни было ваше мнение, попробуй-

те обосновать его на концептуальном уровне, а затем реализовать это изменение, чтобы посмотреть, что получится!

Обратите внимание на способ создания экземпляров класса `StringAttribute` в методе `toInstance`. Фактически, строковые атрибуты становятся булевскими переменными! Еще одно изменение этого программного кода может быть направлено на получение фиксированного набора наиболее часто встречающихся термов с последующим присваиванием атрибутам собственно значений строк, содержащих наиболее часто встречающиеся термы. Вы можете осуществить это, выполнив следующие шаги:

- Создайте параметр, определяющий число обучающих атрибутов.
- Создайте имя для каждого из этих атрибутов, применив простое соглашение; например, используйте постоянный терм с номером: attr-1, attr-2 и так далее.
- Присвойте соответствующее значение каждому атрибуту исходя из порядка перечисления наиболее часто встречающихся термов. То есть атрибут attr-1 получает свое значение по первому из наиболее часто встречающихся термов, атрибут attr-2 получает свое значение по второму терму из этого перечня и так далее.

Создав такую структуру, снова выполните наши примеры и приглядитесь к полученным результатам. Как вы думаете, что произойдет? Считаете ли вы, что такой подход улучшит наши результаты? Если да, то почему вы так считаете?

Какие проблемы могут возникнуть – если вы считаете, что они возникнут? Допустим, вы хотите использовать 32 атрибута, но почему-то в некоторых новостях не набирается 32 часто встречающихся терма, то есть для последних атрибутов (в перечислении от 1 до 32) не найдутся соответствующие часто встречающиеся термы. Каким должно быть значение таких атрибутов? Важно ли это?

4. *Усложненная реализация класса ClassificationStrategyImpl.* В листинге 7.8 мы описали элементарную реализацию класса `ClassificationStrategy` нашего новостного процессора. В представленной реализации метод `assignTopicToStory` является рудиментарным. Поэтому проанализируем, возможны ли другие реализации и будут ли они полезны.

   Рассмотрим случай, где мы получаем результаты классификации с помощью двух классификаторов. В отличие от методов объединения классификаторов, обсуждаемых в главе 6, здесь мы просто хотим использовать классификаторы последовательно, один за другим. Вам, может быть, это кажется странным, но представьте следующий сценарий. Мы обучаем наивный байесовский классификатор на основе контента каждой веб-страницы, но такое обучение не учитывает никаких зависящих от контекста знаний или метаинформации о контенте.

Если ваша веб-страница была загружена с сайта ESPN[1], довольно высока вероятность, что она ссылается на спортивные новости. Мы можем сформулировать правила на основе метатегов веб-страниц и обеспечить выполнение этих правил после того, как наивный байесовский алгоритм завершит работу. Иначе говоря, мы позволяем вероятностному классификатору сначала «очистить» классифицируемые новостные сообщения, а затем выполнить ряд правил, которые, как мы допускаем, будут применимы независимо от того, что представляет собой контент. Беритесь за дело, реализуйте такую стратегию классификации в методе `assignTopicToStory` и сравните полученные результаты с результатами исходной реализации.

Этот подход должен обеспечить вам существенное улучшение показателей правильности автоматизированной классификации! Разумеется, вы могли бы изобрести другие стратегии, возможно, гораздо лучшие. Мы призываем вас к тому, чтобы вы это сделали и сравнили полученные результаты. Заметим, что логика последовательного применения классификаторов не является такой же прямолинейной, как в случае, когда мы хотим классифицировать кластер, потому что сообщения в данной новостной группе почти наверняка взяты из разных источников. Применение этого подхода к новостной группе могло бы оказаться более трудным делом. Следовательно, давайте обратим внимание на второй важный метод класса `ClassificationStrategyImpl`.

Данная конкретная реализация метода `assignTopicToCluster` довольно проста. Определяем одно новостное сообщение из группы как представителя этой группы, классифицируем выбранное репрезентативное сообщение и запоминаем как назначенную новостную категорию (тему), так и представителя кластера. Выбор репрезентативного новостного сообщения основан исключительно на методе `selectLongestStory`, обеспечивающем простую эвристику для выбора из списка новостных сообщений репрезентативной новости. Идея этого выбора состоит в том, что чем больше размер новостного сообщения, тем больше оно пересекается с каждым из остальных сообщений.

Не слишком изобретательный подход, и мы можем его существенно улучшить. Описывая алгоритм k-средних в главе 4, мы ввели понятие центроида. Реализуйте метод `selectRepresentativeStory`, определяющий ближайшее к центроиду новостное сообщение как репрезентативное. Как бы вы добились этого? Учтите тот факт, что вы сравниваете документы, и поэкспериментируйте с разными метриками расстояния.

В заключение, реализуйте метод `assignTopicToCluster`, который не полагается на единственное репрезентативное сообщение, а реализует

[1] Американский кабельный спортивный телевизионный канал (Entertainment and Sports Programming Network, ESPN). – *Прим. перев.*

принцип принятия решения большинством голосов для членов новостной группы. Надо ли учитывать всех членов? Если нет, то как решить, кто должен голосовать?

5. *Существует ли наилучшее распределение кластеров?* Определение идеального кластера носит отчасти субъективный характер и, безусловно, зависит от того, какой кластер мы хотим получить. Взгляните на наш новостной набор данных, используемый по умолчанию: как бы вы объединили в кластеры эти новостные сообщения? Чтобы конкретизировать и сократить число сообщений, возьмите все новостные сообщения, имена файлов которых содержат префикс *biz-*. Какие новости вы объединили бы в одну группу? Почему?

   Чтобы методично отвечать на такие вопросы, вы должны определить меру, позволяющую определить «уровень качества» кластерного образования. Отсюда сам собой возникает следующий вопрос: с помощью чего следует оценивать кластер? С чем надо сравнивать кластер, чтобы можно было сказать, лучше он или хуже? В случае классификации есть структура (как правило, иерархическая) и набор данных для обучения, которые задают ту систему отсчета, с помощью которой мы измеряем качество проведенной классификации. В случае кластеризации мы должны изобрести другие средства измерения качества кластера, потому что система отсчета здесь отсутствует. Следовательно, это измерение должно зависеть от внутренней информации. Какие атрибуты кластерной структуры вы выбрали бы для своей метрики?

6. *Настройка алгоритма ROCK*. Вторая проблема распределения кластеризации, представленного на рис. 7.6, – большое число синглтонов (кластеров, содержащих единственный элемент). Мы знаем, что это число не может быть таким большим, поскольку подбирали новостные сообщения сами.

   Какая, по-вашему, часть алгоритма ROCK в наибольшей степени повлияет на число синглтонов? Как должна сказаться на количестве синглтонов настройка числа нужных кластеров? Как насчет порогового значения для создания связи?

   Что если мы напишем другую реализацию алгоритма `ROCKAlgorithm`? Предполагается, что у этого алгоритма несколько параметров. Пожалуй, главное – выбрать меру сходства для создания матрицы связей. Какая еще мера сходства была бы уместна? Можно ли разработать полностью настраиваемый «вход» для нашего новостного портала? Рассмотрите возможность применения наивного байесовского классификатора для назначения *вероятности существования связи* (probability of linkage) вместо меры сходства.

7. *Иерархическая кластеризация новостных сообщений*. Представление в виде иерархической структуры – наиболее эффективный способ организации информации, предназначенной для восприятия человеком. Иерархические структуры применяются повсеместно.

Папки и файлы на вашем компьютере представлены в виде иерархии (дерева). Каталоги в интернет-магазинах тоже иерархически организованы. Эти структуры нагляднее и удобнее для пользователя, чем список или простое объединение в группы одного уровня.

Мы можем создавать иерархические кластерные структуры путем многократного применения алгоритма кластеризации. По сути, это даже не обязательно должен быть один и тот же алгоритм. Рассмотрите пример кластеризации новостных сообщений, когда сначала применяется алгоритм k-средних, а затем к каждому кластеру, определенному алгоритмом k-средних, применяется алгоритм ROCK. Изменение порядка применения этих двух алгоритмов на обратный и использование единственного алгоритма – две жизнеспособные альтернативы.

Какие преимущества вы видите в каждой из них? С чего бы вы начали реализацию такого решения? Рассматривайте эту задачу с точки зрения не только качества кластеров, но и производительности (стоимости вычислений).

8. *Тонкая настройка кластеризации для каждой новостной категории.* В листинге 7.14 мы описали метод `cluster` класса `TopicalNewsClusterBuilder`. Изучив его внимательнее, вы убедитесь, что в этом коде нет ничего особенного, кроме того, что внутри каждой новостной категории мы объединяем в кластеры только те новостные сообщения, которые входят в эту категорию. Разумеется, имя этого класса было бы ошибочным, если бы мы оставили все как есть. Давайте рассмотрим некоторые возможности.

   Как насчет использования массива целых чисел и массива чисел типа `double`, скажем, `int k[]` и `double linkThreshold[]`, для представления значений таких параметров, как число нужных кластеров и порог создания связи, соответственно? Да, вы можете задействовать разные значения этих двух параметров для каждой новостной категории, но какими именно должны быть их значения? Как бы вы выбрали наилучшее значение для каждого из этих параметров?

   При условии что каждая новостная категория будет иметь собственные репрезентативные темы, какие еще изменения можно внести, чтобы улучшить результаты кластеризации в каждой новостной категории? Прибегните к «мозговому штурму» и проверьте свои идеи экспериментальным путем – здесь нет единственного ответа!

7.9. Ссылки

- Jurafsky, D., and J. H. Martin «Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition», Second Edition. Prentice Hall (Series in Artificial Intelligence), 2008, pp. 1024. *http://www.cs.colorado.edu/~martin/slp.html*.

A

Введение в BeanShell

На протяжении всей этой книги мы используем сценарии на языке BeanShell, чтобы не вдаваться в мелкие подробности и сосредоточить внимание на принципах работы алгоритма. Предоставляемые нами в этой книге сценарии на языке BeanShell обеспечивают общее представление об алгоритмах. Кроме того, они способствуют быстрому проведению экспериментов и интерактивному изучению материала. Пользуясь сценариями на BeanShell, мы обнаружили, что этот язык обладает множеством достоинств, и получили большое удовольствие от работы с ним. В процессе написания этой книги мы протестировали много идей, относящихся к рассматриваемым темам, и полагаем, что, освоив язык BeanShell, вы также оцените его по достоинству. Так что же это за штука – BeanShell?

A.1. Что такое BeanShell?

BeanShell – это облегченный язык сценариев, совместимый с языком программирования Java. В сущности, интерпретатор BeanShell динамически выполняет стандартный синтаксис Java, дополняя его обычными при работе со сценариями «удобствами», такими как нестрогая типизация, команды и упрощенная обработка событий (method closures), похожая на ту, что имеется в языках написания сценариев Perl и JavaScript. Этот язык был создан Пэтом Нимейером (Pat Niemeyer), и вы можете найти его реализацию в проекте с открытым исходным кодом, который так и называется – BeanShell (*http://www.beanshell.org*).

Была сделана попытка стандартизировать этот язык сценариев и включить его в очередной комплект разработчика приложений на языке Java (Java Development Kit, JDK). Как утверждается в запросе на спецификацию Java 274 (Java Specification Request 274):

> *BeanShell – это язык виртуальной машины, поддерживающий динамическое выполнение полной Java-грамматики и семантики, а также прозрачный доступ к Java-объектам и интерфейсам API. Дополнительные возможности, обеспечивающие написание сценариев и удобство работы, входят в язык как ограниченное расширение набора синтаксических правил языка Java. Таким образом, язык BeanShell пытается минимизировать барьеры, как синтаксические, так и связанные с выполнением, между кодом Java-приложений и сценариями, облегчая разработку и способствуя переходу от сценариев к статичному Java-коду (static Java).*

Тут вы, возможно, спросите: если этот язык BeanShell более или менее совпадает с Java, в чем смысл его использования? Попробуем ответить на этот вопрос.

А.2. Зачем нужен язык BeanShell?

С точки зрения его использования в этой книге, язык BeanShell обеспечивает среду, которая помогает вам изучать интеллектуальные алгоритмы и экспериментировать с ними. Что касается изучения, вы можете начать с определенных написанных нами интерфейсов API и сценариев, а приобретя опыт, может быть, захотите подробнее раскрыть какой-то алгоритм с помощью сценариев BeanShell.

С точки зрения исследований и разработки, возможно, вы выберете язык BeanShell для быстрой проверки результатов каких-то настроек своего алгоритма. А может, захотите взаимодействовать с алгоритмами посредством командной строки интерпретатора, чтобы проанализировать отдельные ответные реакции. Допустим, вы только что обучили классификатор и хотите использовать его в качестве «оракула», чтобы выяснить категории конкретных образцов. Если вам неизвестно, что это за образцы, удобно и полезно опрашивать классификатор в интерактивном режиме. Надеемся, эти доводы обеспечают вам необходимую и достаточную мотивацию. Итак, посмотрим, как быстро получить работающую среду BeanShell.

А.3. Выполнение BeanShell

До тех пор пока вам доступен выполняемый модуль Java, а в переменной окружения `classpath` указан JAR-файл среды BeanShell (*bsh.jar*), вы можете запустить BeanShell на выполнение и выполнять в этой среде сценарии из настоящей книги. Исходный программный код в нашем дистрибутиве содержит версию 2.0b4 среды BeanShell. Мы также включили в этот код файлы сценариев для операционной системы Windows и *nix-файлы сценариев, которые могут загрузить интерпретатор BeanShell в окно консоли; эти файлы можно найти в каталоге *deploy\bin* дистрибутива.

Мы исходим из того, что вашим рабочим окружением является ОС Microsoft Windows; вы сможете аналогичным образом настроить эти сценарии для любого другого окружения, где доступен выполняемый модуль Java. Напоминаем: мы предполагаем, что исходный код для этой книги находится в каталоге *C:\iWeb2*. На этот каталог ссылается переменная окружения `%IWEB2_HOME%`, и все соответствующие ссылки даны с учетом этого корневого каталога. Если вы хотите хранить исходный код где-то в другом месте своей системы, то внесите в сценарии соответствующие изменения. Откройте командную строку, для чего перейдите в меню Start, щелкните мышью на пункте Run и введите в текстовом поле команду `cmd`. Перейдите в каталог *C:\iWeb2\deploy\bin*. Вы должны увидеть Windows-файл сценария *bsc.bat*.

Помимо местонахождения исходного кода, этот сценарий предполагает также, что переменная окружения `%JAVA_HOME%` задана и видна в окружении командной строки. Если это не так, то, по крайней мере, в вашей переменной окружения `PATH` должен быть указан выбранный вами выполняемый модуль Java. Обратите внимание: весь программный код мы тестировали с версиями Java не ниже 1.5. Рекомендуем инсталлировать самый последний релиз JDK (версия 6) с сайта фирмы Sun.

Наконец, поскольку вы выполняете (точнее, интерпретируете) программу на языке Java, то можете настроить опции JVM-машины из командной строки соответственно своим потребностям и характеристикам системы. Мы рекомендуем в качестве настроек для кучи (heap) JVM-машины использовать параметры `-Xms256M -Xmx1280M`, но эти значения могут быть слишком велики для вашей системы. Помните об этом и если возникнут какие-либо проблемы, настройте свои значения соответствующим образом.

А.4. Ссылки

- Проект BeanShell с открытым исходным кодом: *http://www.beanshell.org/home.html*.
- JSR 274: The BeanShell Scripting Language. Java Specification Requests: *http://jcp.org/en/jsr/detail?id=274*.

B

Краулинг

В этом приложении дан обзор компонентов процедуры краулинга (обхода веб-контента поисковым роботом), кратко описаны подробности реализации робота, распространяемого вместе с этой книгой, а также роботов с открытым исходным кодом, написанных на языке программирования Java.

B.1. Обзор компонентов поискового робота

Поисковые роботы (crawlers) предназначены для обнаружения, загрузки и сохранения контента Интернета. Как мы видели в главе 2, поисковый робот является частью более крупного приложения, такого как поисковая система.

Типичный поисковый робот включает следующие компоненты:

- Модуль, обслуживающий хранилище всех известных роботу url-адресов
- Модуль загрузки документов, который извлекает документы из интернета по предоставленному набору url-адресов
- Модуль синтаксического разбора документов (парсер), отвечающий за извлечение оригинального контента из документов разных форматов, таких как html, pdf, microsoft word. Парсеры также отвечают за извлечение содержащихся в документе url-адресов и других данных, которые могут быть полезны на этапе создания индексов – в частности, метаданных
- Модуль, обслуживающий хранилище метаданных и контента, извлеченных из оригинальных загруженных документов в процессе совершаемого роботом обхода
- Модуль нормализации url-адреса, который преобразует url-адреса в стандартную форму, так что их можно сравнивать, вычислять и так далее

- Модуль фильтрации url-адресов, необходимый для того, чтобы робот мог пропустить нежелательные для посещения url-адреса

Структура и реализация отдельных компонентов зависит от того, что вы планируете обходить, и от масштаба обработки, которую должен осуществить робот. В простейших случаях, когда надо собрать пару страниц с известного веб-сайта, весь код реализации робота может уместиться на одной странице. Для обхода веб-сайтов сети интранет также можно обойтись довольно простой реализацией. Но в реализации, способной обрабатывать крупномасштабные коллекции документов из Интернета, робот будет представлен как набор приложений, распределенных по сети аппаратных узлов. Эти узлы могут быть распределенными даже с точки зрения их географического местоположения, чтобы быть ближе к источнику данных. В конце этого раздела мы даем ряд ссылок на литературу, где описана реализация, относящаяся к крупномасштабному краулингу.

В.1.1. Этапы краулинга

Функционирование типичного робота включает следующие два этапа:

1. Хранилище URL-адресов робота инициализируется списком URL-адресов (их обычно называют начальными URL-адресами, seed URLs) и начинается краулинг.
2. Робот загружает еще не посещенные им начальные URL-адреса, чтобы определить область своей работы.

Для каждого подходящего URL-адреса поисковый робот должен:

1. Получить контент, который можно обнаружить по этому URL-адресу.
2. Выполнить синтаксический разбор полученных документов, чтобы извлечь исходящие URL-адреса и содержимое документа.
3. Сохранить ту информацию, которую мы хотим оставить себе.
4. Нормализовать обнаруженные новые URL-адреса.
5. Отфильтровать URL-адреса, которые робот должен игнорировать.
6. Обновить хранилище URL-адресов списком новых адресов для посещения.
7. Повторять шаг 2 до тех пор, пока не будет достигнута заданная глубина обнаружения контента.

Есть две основные категории поисковых роботов: универсальные и специализированные. Универсальные роботы собирают все документы, которые они могут заполучить. Чтобы ограничить список URL-адресов, предназначенных для обхода, они могут применять методики фильтрования URL-адресов. Специализированные роботы служат для обнаружения контента, относящегося к конкретной интересующей теме. Все роботы могут с пользой для себя использовать методики, рассмотренные в этой книге.

В.1.2. Наш простой поисковый робот

Наш простой поисковый робот может загружать страницы из Интернета или из локальной файловой системы. Кроме HTML-документов, он может обрабатывать документы текстового процессора Microsoft Word. Как мы уже говорили, это показательный вариант робота, предназначенный для того, чтобы облегчить вам экспериментирование с интеллектуальными методиками, представленными в этой книге. Программный код создавался так, чтобы он был ясным и простым, и вы без труда могли получить представление о том, что происходит на каждом шаге обработки. Каждый документ сохраняется в отдельном файле; робот использует для этого файла обычный текстовый формат, так что вы легко можете просматривать его содержимое. Весь связанный с нашим роботом исходный программный код можно найти в пакете `iweb2.ch2.webcrawler`.

Мы демонстрируем работу нашего кода в приводимых в книге листингах. Каждый раз при выполнении робота, он создает новый каталог `crawl-<метка-времени>` для хранения всех релевантных данных. В этом каталоге будет создан следующий набор подкаталогов и файлов:

- `<crawl-dir>/knownurls/knownurlsdb.dat` – этот файл содержит список известных роботу URL-адресов
- `<crawl-dir>/fetched/` – в этом каталоге хранятся оригинальные документы, загруженные роботом. Документы загружаются и обрабатываются в пакетном режиме; по умолчанию размер пакета равен 10
- `<crawl-dir>/fetched/<batch-number>/<doc-id>.fetched` – в каждом файле хранится оригинальное содержимое конкретного документа
- `<crawl-dir>/fetched/<batch-number>/<doc-id>.meta` – каждый файл содержит метаданные содержимого документа; эти метаданные определяются транспортным протоколом
- `<crawl-dir>/processed/` – в этом каталоге хранятся документы в обработанном виде. Здесь есть отдельный подкаталог для каждого обработанного пакета URL-адресов
- `<crawl-dir>/processed/<batch-number>/content/<doc-id>.content` – каждый документ содержит контент, полученный после синтаксического разбора. Если вы замените парсер, содержимое этих документов может измениться
- `<crawl-dir>/processed/<batch-number>/outlinks/<doc-id>.outlinks` – каждый файл содержит все ссылки, обнаруженные парсером документов в соответствующем родительском документе
- `<crawl-dir>/processed/<batch-number>/properties/<doc-id>.properties` – в каждом файле хранятся свойства документа, такие как тип документа, заголовок, URL-адрес и так далее

- `<crawl-dir>/processed/<batch-number>/txt/<doc-id>.txt` – каждый файл содержит только текст из загруженного документа, извлеченный парсером
- `<crawl-dir>/pagelinks/pagelinkdb.dat` – в этом файле хранятся данные о внешних ссылках для всех обработанных URL-адресов. Запоминаются внешние ссылки только тех URL-адресов, которые прошли фильтрацию

В.1.3. Поисковые роботы с открытым исходным кодом

Есть два главных проекта на базе Java – Nutch и Heritrix, – которые предлагают реализацию поискового робота. Nutch – это подпроект Apache-проекта Lucene, который вы можете найти по адресу *hhttp://lucene.apache.org/nutch/*. На сайте этого проекта есть пошаговое руководство по инсталляции и выполнению приложения, которое, разумеется, включает поисковый робот. В отличие от нашего робота, робот из проекта Nutch хранит в одном файле данных сразу несколько страниц. Это затрудняет просмотр результатов, полученных в случае небольшого краулинга, но при крупномасштабном краулинге это обеспечивает более высокую производительность операций ввода/вывода.

Heritrix – это поисковый робот с открытым исходным кодом организации Internet Archive. Он предназначен для архивирования крупных «кусков» Интернета. Веб-сайт проекта находится по адресу *http://crawler.archive.org/*. Проект Heritrix ориентирован на крупномасштабный краулинг, его релиз выпущен со свободной лицензией LGPL[1].

Еще одну библиотеку с открытым исходным кодом, которая вас может заинтересовать, предоставляет проект Apache Tika, адрес его веб-сайта: *http://lucene.apache.org/tika/*. Tika – это набор инструментов для обнаружения и извлечения метаданных и структурированного текстового содержимого различных документов с помощью готовых библиотек парсеров.

Более подробно различные аспекты структуры поискового робота описаны в [Heydon, A. and M. Najork] и [Gomes, D. and M. Silva]. Обзор вопросов, требующих рассмотрения при проектировании высокопроизводительных масштабируемых поисковых роботов дан в [Boswell, D.] и [Shkapenyuk, V. and T. Suel]. Подробное описание специализированного обхода вы найдете в статье [Chakrabarti, S., M. Berg, and B. Dom].

[1] GNU Lesser General Public License – стандартная общественная лицензия ограниченного применения GNU. – *Прим. перев.*

В.2. Ссылки

- Boswell, D. «Distributed High-performance Web Crawlers: A Survey of the State of the Art», 2003, *http://www.cs.ucsd.edu/~dboswell/PastWork/WebCrawlingSurvey.pdf*.
- Gomes, D. and M. Silva «The Viuva Negra crawler: An experience report», Software—Practice & Experience, Volume 38 (2), pp. 161–188, 2006.
- Heydon, A. and M. Najork «Mercator: A Scalable, Extensible Web Crawler», Compaq Systems Research Center, 1999.
- Chakrabarti, S., M. Berg, and B. Dom «Focused crawling: a new approach to topic-specific Web resource discovery», WWW8 International World Wide Web Conference, vol. 31, pp. 1623–1640. Toronto, 1999. *http://www8.org/w8-papers/5a-search-query/crawling/index.html*.
- Shkapenyuk, V. and T. Suel «Design and Implementation of a High-Performance Distributed Web Crawler», Polytechnic University: Brooklyn, NY, 2001. *http://cis.poly.edu/suel/papers/crawl.pdf*.

C

Памятка по математике

При изложении материала в настоящей книге мы не предполагали наличия у нашей аудитории специальной математической подготовки. В этом приложении приведены несколько математических формул, которые мы использовали в своих алгоритмах, но ни разу не приводили явно, в стандартном виде. Классический труд, объединяющий сжатое математическое описание с численными алгоритмами, – книга [Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery], непременное справочное пособие при рассмотрении множества тем, связанных с разработкой интеллектуальных алгоритмов.

C.1. Векторы и матрицы

В контексте элементарной математики и физики вектор – математическое представление стрелки. Самый распространенный и интуитивно понятный пример – это стрелка, соединяющая две точки на плоскости, например A и B. В применяемых нами формулах вектор представлен одномерным массивом. Другими словами, вектор – это упорядоченное множество чисел. Как правило, мы обозначаем вектор латинской буквой в полужирном начертании, например x. Например, вектор длиной 10 это упорядоченное множество чисел $\mathbf{x}_i$, где индекс i получает значения от 1 до 10 включительно, или от 0 до 9 включительно, если вы начинаете отсчет с 0.

Двумерную матрицу можно представить таблицей со строками и столбцами. Каждый элемент матрицы соответствует ячейке в таблице. В применяемых нами формулах для представления матрицы используется двумерный массив. Как правило, мы обозначаем матрицу заглавной буквой латинского алфавита в полужирном курсивном начертании, например A. Как и вектор, матрицу можно обозначить, указав ее отдельные упорядоченные элементы. Иными словами, матрицу A можно также записать как A_{ij}, где два индекса могут иметь разные диапазоны.

Матрица, полученная в результате изменения порядка индексов на обратный, называется *транспонированной* матрицей исходной матрицы. Следовательно, A_{ji} – это транспонированная матрица матрицы A_{ij}; понятно, что обратное также справедливо!

Матрица не обязательно должна быть двумерной; она может быть n-мерной, где n – произвольное целое. Разумеется, у n-мерной матрицы n индексов. Можно считать вектор частным случаем одномерной матрицы. У матриц много интересных свойств, и их можно использовать для символических операций, точно так же, как мы используем переменные x и y для обозначения неизвестных в элементарной алгебре. Пока речь идет о символических операциях, можно считать, что матрицы – это числа! Но в отличие от обычных чисел, две матрицы не обязательно обладают переместительным свойством при перемножении. Следовательно, в общем случае, если мы имеем дело с двумя матрицами A и B, скорее всего $A\,B \neq B\,A$.

Таким образом, для большинства практических целей можно считать, что вектор одномерным Java-массивом, а матрицу – дву- или (в более общем случае) n-мерным Java-массивом. О векторах и матрицах написано очень много. Вы найдете массу учебников, в которых эта тема раскрывается во всех подробностях и на разных уровнях сложности; некоторые классические учебники мы указали в ссылках на литературу.

С.2. Измерение расстояний

В главе 3 мы говорили от том, что есть много способов измерить расстояние между двумя точками, например A и B. Посмотрим, как все эти способы измерения расстояния переводятся на язык формул с использованием понятия вектора. Если каждую точку представить вектором (a и b для точек A и B, соответственно), то разность между этими двумя векторами – это еще один вектор, например **x**, абсолютная величина которого позволяет измерить расстояние между этими двумя точками.

Наиболее часто используют евклидово расстояние, которое называют также *L^2-расстоянием* (distance), или *L^2-нормой* (norm). Как можно представить, евклидово расстояние является особым случаем общей формулы расстояния, которое называют *L^p-нормой*, где p может быть любым действительным числом, большим или равным 1. Формула для L^p-нормы имеет следующий вид:

$$\|x\|_p = \left(\sum_{i=1}^{n} |\, x_i \,|^p\right)^{\frac{1}{p}}$$

Евклидово расстояние получается при $p = 2$. Расстояние «городских кварталов» , или «расстояние таксистов» («taxi cab» distance), которое рассматривалось в п. 1 раздела «Сделать» главы 3, получается при

$p = 1$. Разумеется, в предыдущей формуле вектор $\mathbf{x}$ – это разность между двумя векторами, расстояние между которыми мы хотим измерить: $x = b - a$.

Еще одна формула расстояния, которую мы часто применяем, связана с метрикой *косинусного сходства* (cosine similarity). Чтобы получить формулу, которую мы использовали для получения косинусного сходства, нужно определить внутреннее произведение двух векторов (его часто называют *скалярным произведением* – dot product). Внутреннее произведение двух векторов измеряет *проекцию* одного вектора на другой. На рис. С.1 показаны два (двумерных) вектора на плоскости. Если два вектора перпендикулярны друг другу, то проекция одного вектора на другой минимальна (равна нулю). Если два вектора лежат на одной прямой (параллельны), проекция максимальна.

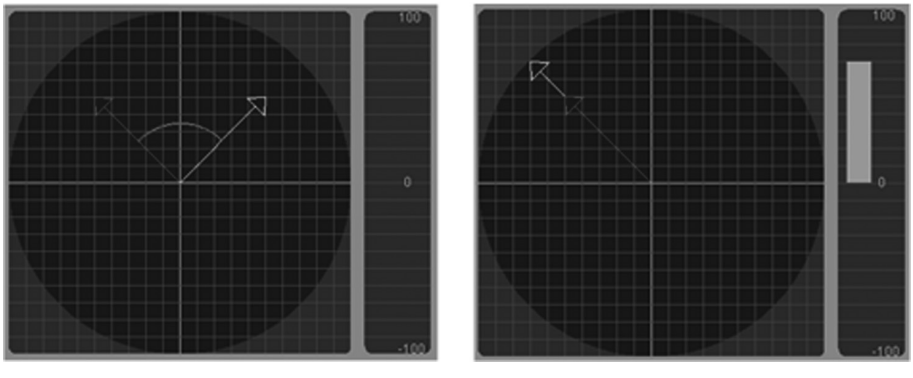

Рис. С.1. *Внутреннее произведение двух векторов, перпендикулярных (ортогональных) друг другу (слева) и лежащих на одной прямой (справа)*

Чтобы получить эти изображения, мы использовали свободно доступный апплет. Вы можете поэкспериментировать с этим и другими апплетами в онлайн-режиме по адресу *http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/linear_algebra.html*. Формула для внутреннего произведения двух векторов $\mathbf{a}$ и $\mathbf{b}$, из n элементов каждый, имеет следующий вид:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

На основании внутреннего произведения мы определили косинусное сходство по формуле:

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{(\mathbf{a} \cdot \mathbf{a})(\mathbf{b} \cdot \mathbf{b})},$$

где Θ – это угол между двумя векторами a и b. Программный код для реализации этой формулы реализован в классе `iweb2.ch3.collaborative.similarity.CosineSimilarityMeasure`. Иногда используют величину самого угла, применяя функцию арккосинуса в правой части уравнения (`Math.acos`). Но если вы оперируете показателями сходства, это вопрос удобства. Выбор того или иного варианта оказывается важным, если используемая величина должна удовлетворять требованиям математической нормы, такой как L^p-норма, например.

Если вы хотите сформулировать собственные нормы и меры расстояний, наиболее подходящие для нужд вашей задачи, следует почитать учебник по функциональному анализу. Мы рекомендуем книгу [Kolmogorov, A. N., and S. V. Fomin]. Измерение расстояний, с точки зрения математики, связано с изучением метрических пространств. Требуются некоторые познания в теории множеств, чтобы надлежащим образом разобраться в сходимости последовательностей, то есть в понятиях предельных точек, открытых и замкнутых множеств и так далее. Есть огромное количество книг по функциональному анализу; мы рекомендуем именно эту книгу, потому что она отличается лаконичным, но полным изложением материала и относительно недорога.

C.3. Развитые матричные методы

Если ваша задача уже «переведена» на язык матриц, вам доступен огромный арсенал методик и подходов. Вы можете подробно изучить свойства своей системы с той строгостью, которую может предложить только математика. Разумеется, на деле это изучение может оказаться не таким простым, как на словах. Не исключено, что в вашем случае будут применимы только некоторые из имеющихся методов, к тому же интеллектуальное сопоставление матрицы концептов с характеристиками вашей задачи – не только наука, но и искусство. Тем не менее, если у вас есть склонность к математике и интеллектуальным приключениям, мы настоятельно рекомендуем знакомство с книгами, в которых развитые матричные методы применяются для анализа данных.

Не так давно была издана одна из подобных книг [Eldén, L.], посвященная, как и наша, интеллектуальным алгоритмам. В этой книге рассматриваются фундаментальные основы векторов и матриц, систем линейных уравнений и методы наименьших квадратов, ортогональность векторов и матриц, различные разложения матриц (QR и сингулярное SVD), а также неотрицательная факторизация матриц. Особый интерес для наших читателей может представлять ее раздел 9.1, где рассмотрен алгоритм k-средних. В оставшейся части этой книги описано применение инструментов матричных вычислений в решении таких задач, как классификация рукописных цифр, анализ текста, в алгоритме PageRank (который мы рассмотрели в главе 2), для автоматического извлечения ключевых слов и предложений, а также для распознавания

лиц с помощью тензорного сингулярного разложения. Научная литература по вычислениям изобилует ссылками, поэтому мы настоятельно рекомендуем книгу [Eldйn, L.] для более глубокого погружения в некоторые из алгоритмов, представленных в нашей книге.

С.4. Ссылки

- Eldén, L. «Matrix Methods in Data Mining and Pattern Recognition» (Series: Fundamentals of Algorithms), SIAM: Society for Industrial and Applied Mathematics, 2007.
- Kolmogorov, A. N., and S. V. Fomin «Elements of the Theory of Functions and Functional Analysis»[1], Dover Publications, Inc. New York, 1961.
- Lay, D. C. «Linear Algebra and Its Applications», Third Edition, Addison Wesley, 2005.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery «Numerical Recipes: The Art of Scientific Computing», Third Edition, Cambridge University Press, 2007.
- Trefethen, L. N., and D. Bau «Numerical Linear Algebra», SIAM: Society for Industrial and Applied Mathematics, 1997.

[1] А. Н. Колмогоров, С. В. Фомин «Элементы теории функций и функционального анализа» – М., Физматлит, 2006.

D

Обработка естественных языков

На протяжении всей этой книги мы использовали обработку текстов на естественных языках (Natural Language Processing, NLP). Обработкой текстов на естественных языках называют набор методик и методов, предназначенных для обработки письменной и устной (обычно человеческой) речи. В практическом отношении эти средства помогают обрабатывать текст и аудиозаписи в целях анализа их содержимого. Как можно догадаться, это поле деятельности столь же обширно, сколь интересно.

Начало работ в области NLP приходится на первые годы развития ИИ. Фактически, знаменитый тест Тьюринга был сформулирован с точки зрения способности компьютера общаться с человеком по проводной связи, когда человек не может распознать, является существо на другом конце провода человеком или нет; хороший обзор теста Тьюринга дан в работе [Saygin, A.P., I. Cicekli, and V. Akman]. В столь давно развивающейся области вы можете обнаружить несколько ответвлений, которые рассматривают эту же проблему под другим углом. Например, такие понятия, как вычислительная лингвистика и синтез речи, относятся к областям исследований, направленным на решение задач того же рода (или близко связанных), что и у NLP.

Превосходный справочник по NLP – книга [Jurafsky, D. and J. Martin]. Авторы разбивают прикладную область естественных языков на следующие составляющие.

- *Фонетика и фонология* – изучение произношения и распознавание слов на основе человеческого голоса.
- *Морфология языка* – изучение структуры словоформ, являющихся носителями смысла в человеческом языке; например, распознавание того, что «phone» так же относится к phonology, как «hydro» к «hydrology».

- *Синтаксис языка* – изучение конструкции предложений естественных языков. Первостепенную важность синтаксиса хорошо иллюстрирует пример известного предсказания, полученного неким солдатом у Дельфийского оракула, прежде чем отправиться на войну. Оракул сказал (по-древнегречески) буквально следующее: «ηξειζ αφφηξειζ θυηξειζ». Эти слова можно интерпретировать двояко. В первой интерпретации (оптимистичной) оракул сказал: «Ты пойдешь, ты вернешься, ты не погибнешь на войне» Во второй интерпретации (пессимистичной) оракул сказал: «Ты пойдешь, ты не вернешься, ты погибнешь на войне». Знание синтаксиса для правильной интерпретации послания Аполлона было жизненно важным!
- *Семантика* – изучение смысла слов в конкретном контексте. Например, слово «cool» в американском английском можно интерпретировать как «прохладный», «уверенный и не испытывающий волнения» или «откровенно равнодушный или презрительный», в зависимости от контекста.
- *Прагматика* – изучение взаимосвязи между смыслом слов и намерениями или целями общения.
- *Дискурс* – изучение лингвистических образований, более крупных, чем одно выражение.

[Jurafsky, D. and J. Martin] блестяще раскрывают все эти составляющие. Кроме того, они предлагают исчерпывающий обзор таких тем, как синтез и распознавание речи, статистический разбор, временной анализ выражений и даже агенты-собеседники и машинный перевод.

Если подходить к проблеме с более практической точки зрения, существует несколько проектов с открытым исходным кодом, разработка которых связана с NLP, вполне заслуживающих потраченного на знакомство с ними времени, если вас интересуют вопросы работы с текстовым представлением и другими лингвистическими представлениями информации. Проект UIMA был инициирован фирмой IBM больше десяти лет тому назад и в настоящее время стандартизирован консорциумом OASIS (*http://www.oasis-open.org/*). UIMA (Unstructured Information Management Applications – приложения для обработки неструктурированной информации) ссылается на программные системы, которые анализируют большие объемы неструктурированной информации (например, неформатированный текст бизнес-отчетов, аналитики и контрактов). Этот проект сейчас поддерживается в Apache-инкубаторе по адресу *http://incubator.apache.org/uima/*. Как всякий долгожитель, этот проект стабилен и вполне активен.

Еще один стабильный, превосходно документированный проект – GATE (General Architecture for Text Engineering – универсальная архитектура для обработки текста), разработка которого ведется в Шеффилдском университете с 1995 года. Он очень подходит для старта, если вы ищете практические примеры обработки естественного языка, иллюстрирую-

щие широкий спектр вопросов из списка [Jurafsky, D. and J. Martin]. Три элемента этого проекта согласно документации:

- Архитектура, которая описывает, как надо строить системы обработки языка, состоящие из отдельных компонентов
- Каркас на языке программирования java, обеспечивающий базовые интерфейсы api
- Графическая среда разработки. Проект можно найти по адресу *http://gate.Ac.Uk/*

Возможно, вас также заинтересует проект MinorThird, написанный (главным образом) Уильямом Коэном (William Cohen), профессором университета Карнеги – Меллон на кафедре машинного обучения [Cohen, W. W.]. Вот что говорят об этом проекте его основатели: «MinorThird – это коллекция Java-классов, предназначенных для хранения текста, аннотирования текста и обучения тому, как извлекать сущности и классифицировать текст». Проект можно найти по адресу *http://sourceforge.net*[1]; он распространяется с лицензией BSD[2], но прежде чем встраивать эти программные средства в свой коммерческий код, прочтите замечания, касающиеся использования библиотек сторонних производителей. Несмотря на эти оговорки, данная библиотека включает широкий набор NLP-методов, облегчающих анализ текста. Среди других примечательных NLP-библиотек с открытым исходным кодом: предшественник библиотеки MinorThird – проект SecondString и еще один «зонтичный» проект, который называется openNLP; оба эти проекта также можно найти на портале *http://sourceforge.net*.

D.1. Ссылки

- Cohen, W. W. «Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data», *http://minorthird.sourceforge.net*, 2004.
- Jurafsky, D. and J. Martin «Speech and Language Processing», Second Edition. Prentice Hall (Series in Artificial Intelligence), 2008.
- Saygin, A. P., I. Cicekli, and V. Akman «Turing Test: 50 Years Later», Minds and Machines, 10 (4) pp. 463–518, 2000.
- Turing, A. «Computing Machinery and Intelligence», Mind, 59 (236), pp. 433–460, 1950.

---

[1] Точный адрес MinorThird: *http://sourceforge.net/apps/trac/minorthird/wiki*. – *Прим. науч. ред.*

[2] BSD (Berkley Software Distribution license) – программная лицензия университета Беркли. – *Прим. перев.*

E

Нейронные сети

В главе 5 мы познакомили вас с основными понятиями нейронных сетей. Мы также продемонстрировали, как построить простую нейронную сеть, обучить ее и использовать в практическом сценарии. Тема нейронных сетей в целом неисчерпаема; мы затронули только верхушку айсберга. Трудно обсуждать фундаментальные основы нейронных сетей, не используя математическую терминологию. Мы надеемся, что неплохо объяснили элементарные понятия, но более глубокое понимание работы внутренних механизмов требует погружения в специальную литературу.

Чтобы облегчить переход от «Алгоритмов интеллектуального Интернета» к узкоспециальной литературе, посвященной нейронным сетям, мы хотели бы рекомендовать две книги, обеспечивающих доступное введение в математическое описание нейронных сетей. Книга [Mitchell, T.] – превосходное введение в машинное обучение, и мы настоятельно рекомендуем ее как универсальный справочник. В частности, материал главы, посвященной искусственным нейронным сетям, включает алгоритм обратного распространения и его математическое происхождение, подробный пример распознавания лиц, альтернативные функции ошибок, альтернативные процедуры минимизации ошибок и динамическую модификацию сетевой структуры.

В качестве первого шага в общую литературу мы можем рекомендовать [MacKay, D. J. C.]. Эта книга содержит более сложное (и более современное) вводное рассмотрение искусственных нейронных сетей. Это рассмотрение начинается с обзора нейронных сетей и подробного представления единственного нейрона как классификатора. В книге тщательно анализируются возможности одиночного нейрона и предлагается вероятностная интерпретация процесса обучения нейронной сети. [MacKay, D. J. C.] представляет так называемые *машины Больцмана* (Boltzmann machines) (интереснейшая тема для тех, кто обладает познаниями в физике) как стохастическую сеть Хопфил-

да (Hopfield). Байесовские нейронные сети также используются как трамплин для введения в гауссовы процессы, традиционные статистические модели. Следует отметить, что алгоритмы в [MacKay, D. J. C.] реализованы на бесплатном языке математического программирования Octave с открытым исходным кодом. Платформа для выполнения кода на этом языке доступна для всех основных операционных систем, так что с этим кодом можно легко экспериментировать.

В категории книг, посвященных изучению нейронных сетей, обязательной для чтения по этой теме является книга одного из великих мастеров нейронных сетей [Kohonen, T.]. Помимо мастерски изложенной темы обучения без учителя с помощью нейронных сетей эта книга предлагает ясную и лаконичную главу, в которой изложены математические предпосылки и дано одно из лучших представлений методики дискретизации вектора обучения (learning vector quantization). Эта компактная, на уровне университетского учебника, книга столь же приятна для чтения, сколь и оригинальна.

В книге [McNelis Paul D.] все внимание, как и предполагает название, уделяется применению нейронных сетей в финансовой сфере. После нескольких вводных глав в этой книге рассматриваются: моделирование ряда задач из области финансов, таких как модель ценообразования опционов Блэка – Шоулза (Black – Sholes), моделирование корпоративных облигаций, прогнозирование с помощью временных рядов, невыполнение обязательств по кредитным картам и банкротство банков, а также моделирование инфляции и дефляции.

Мы также хотим упомянуть книгу [Hirose Akira], представляющую систематическое раскрытие новой развивающейся темы комплексных нейронных сетей. Эти сети могут быть полезны в таких областях, как адаптивные радарные системы, обработка послойных цифровых карт и синтез речи. В общем случае, любая сфера деятельности, чья предметная область поддается естественному представлению с помощью комплексных величин, может выиграть от присущего этим сетям отображения в понятиях комплексных чисел. Конечно, что эта книга не является вводной, но она может предложить новые идеи и раскрыть возможности универсальной структуры нейронных сетей, а также вскрывает проблемы этой универсализации. Подробный список книг по изучению нейронных сетей приведен в разделе ссылок.

Е.1. Ссылки

- Arbib, M. A., S. Amari, NetLibrary, Inc., and P. H. Arbib «The Handbook of Brain Theory and Neural Networks», MIT Press, 2003.
- Bishop, C. M. «Neural Networks for Pattern Recognition», Oxford University Press, 1996.
- Dreyfus, G. «Neural Networks», Springer, 2005.

- Haykin, S. «Neural Networks: A Comprehensive Foundation», Second Edition, Prentice Hall, 1998.
- Hirose Akira «Complex-valued Neural Networks», Studies in Computational Intelligence, Springer, 2006.
- Kohonen, T. «Self-Organizing Maps»[1], Third Edition, Springer, 2000.
- MacKay, D. J. C. «Information Theory, Inference, & Learning Algorithms», Cambridge University Press, 2003.
- Maier, K. D., C. Beckstein, R. Blickhan, W. Erhard, and D. Fey «A multilayer-perceptron neural network hardware based on 3D massively parallel optoelectronic circuits», Proceedings of the 6th International Conference on Parallel Interconnects, pp. 73–80, 1999.
- Mandic, D. P., and J. A. Chambers «Recurrent Neural Networks for Prediction: Learnning algorithms, architecture, and stability», John Wiley & Sons, Inc., 2001.
- McNelis Paul D. «Neural Networks in Finance: Gaining Predictive Edge in the Market», Elsevier Academic Press, 2005.
- Mitchell, T. «Machine Learning», McGraw Hill Higher Education, 1997.
- Neapolitan, R. E. «Learning Bayesian Networks», Prentice Hall, 2003.
- Ripley, B. D. «Pattern Recognition and Neural Networks», Cambridge University Press, 2008.

[1] Тойво Кохонен «Самоорганизующиеся карты» – М., Бином. Лаборатория знаний, 2008 г.

Алфавитный указатель

A

AdaBoost, алгоритм, 325, 365, 367, 379
alpha, 104
Amazon.com, 162, 163
  один из первых сайтов, предложивших рекомендации, 25
  подход с объединением предмет–предмет, 142
Android, 393
Anscombes quartet, 167
ANTLR, проект, 273
Apache Axis, 41
Apache CXF и поддержка множественных форматов, 42
Apache Shindig, 32
Apache-проект POI, 387
arc-fs, алгоритм, 379
arc-x4, алгоритм, 325, 365, 367, 379
assignTopicToCluster, метод, 405, 406
assignTopicToStory, метод, 405, 406
AST (abstract syntax tree), 273
Atom, 39
  не базируется на модели RDF, 40
  формат синдикации, 40
Attribute, алгоритм, 250
AverageLinkAlgorithm, класс, 196

B

BadUserType, класс, 332
bagging, 355, 362
  алгоритм, 357
  исходная предпосылка, 355
  маленькие хитрости и советы, 378
  независимые классификаторы, 365
  разброс данных, 356
  тонкая настройка, 359
BaggingCreditClassifier, класс, 357, 358, 359
  правильность и время выполнения, 358
BaggingCreditClassifierbalanced, класс, 193
BaseConcept, класс, 404
BaseInstance, класс, 253
BaseInstance, объект, 404
BaseLayer, класс, 301
BaseNN, класс, 291, 297, 299
BasicWebCrawler, 58, класс, 386
BeanShell, оболочка, 429
BIRCH, алгоритм, 193
  кластеризация, 227
Boorah, поисковая система, 39
BoostCreditClassifier, класс, 367
boosting, алгоритм, 64, 325, 365, 374
  основная идея, 365
  производительность вычислений, 368
BoostingARCX4Classifier,класс, 368, 370
BoostingCreditClassifier, класс, 368
bootstrap, методика, 309, 355
  агрегирование, 325, 355
  процесс самонастройки, 357
BootstrapTrainingSetBuilder, класс, 360, 379
BSD, лицензия, 62

C

C5.0, алгоритм, 244
Census of Fatal Occupational Injuries, (CFOI), документ, 238
CF, алгоритм, 142
  на основе сходства предметов, 138
  требования, 127
Chi2, переменная, 348
Cinematch, система выработки рекомендаций, 26, 161
ClassificationStrategyImpl, класс, 401, 402, 405, 407, 426
ClassificationStrategy, интерфейс, 405
ClassificationStrategy, класс, 425

ClassifierEnsemble, класс, 359, 360, 362, 368, 372
ClassifierResults, 347
classifier selection, методика, 323
Classify-Cluster-DS, окно, 399
classifyClusters, метод, 407
classifyStories, метод, 407
ClassLoader, класс, 274
Clearspring, фирма, 34
Clementine, система анализа данных, 244
CLIPS, библиотека, 243
Cluster-Classify-DS, окно, 399
Concept, класс, 85, 239, 250
CONCEPT_LABEL_FRAUD, 295
CONCEPT_LABEL_VALID, 295
ConceptMajorityVoter, класс, 363
conceptPriors, карта, 88, 259
content, поле, 58, 63
  индексируемое, но не сохраняемое, 58
  сохраняемое, 59
CosineDistance, класс, 219
CosineSimilarity, класс, 215, 264
CosineSimilarityMeasure, класс, 146
crawler, поисковый робот, 39, 385, 432
CrawlResultsNewsDataset, класс, 389
createClustersWithinTopics, класс, 414
createClustersWithinTopics, метод, 408
createClusters, метод, 408, 409
credibility, параметр, 308
CreditErrorEstimator, класс, 338, 366
CreditErrorEstimator, метод, 377
CreditScore, 328
CriminalRecord, 329
CustomAnalyzer, 145
Cutter number, шифр, 237

D

DAG, 246, 284
DangerousUserType, класс, 332
DataGenerator, класс, 333
DataPoint, класс, 196, 212
DataPoint, массив, 221
DatasetAdapter, класс, 421
DBSCAN, 218
  eps, переменная, 221
  minPoints, переменная, 222
  алгоритм, 232
  аналогия с каплями чернил, 218
  непосредствнная плотно-достижимость, 222
  окруженная точка, 222
  пограничная точка, 222
DBSCANAlgorithm, класс, 219, 221, 222
decisionTree, команда, 342 243
Delphi
  внутренние механизмы, 134
  система выработки рекомендаций, 127, 128
  сходство между пользователями, 130
Delphi, класс, 420, 421
  Dataset, интерфейс, 128
DelphiUC, класс, 156
DelphiUR, класс, 156
Diff2PropTest, класс, 350
Digg, утилита
  API, 150, 151, 212
  службы RESTful, 41
DiggCategory, класс, 153
DiggDelphi, класс
  findSimilarUsers, метод, 156
  getTopNFriends, метод, 156
  recommend, метод, 158
  внутренние механизмы, 155
Distance, интерфейс, 221
docid, поле, 58
DocRank, 94, 95, 383, 392
  внутренние механизмы, 96
  повторно использованные значения, 100
  построитель матрицы, 97
doctype, поле, 58
DownPayment, 329
Drools, 236, 243, 267, 271
  ReteOO, 268
DTCreditClassifier, класс, 341, 367
DTCreditClassifier, классификатор, 358

E

ECLiPSe, язык программирования, 267
EmailClassifier, класс, 250, 251, 252, 261
EmailData, 251
EmailDataset, класс
  getTrainingSet, метод, 253
  setBinary(false), метод, 264
EmailDataset, набор, 251
EmailInstance, класс, 253
EmailRuleClassifier, 271
Engage, 32
ESPN, сайт, 426
EuclideanDistance, 123, 186
ExcellentUserType, класс, 332
expectation-maximization, EM, алгоритм, 231

F

Facebook, 24, 25, 30
  API-интерфейс RESTful, 41
false, логическое состояние, 306
False Negative, FN, алгоритм, 307
False Positive, FP, алгоритм, 307
FASTCLUS, процедура, 210
feedback, вид топологии графа, 284
feedforward, вид топологии графа, 284
FetchAndProcessCrawler, класс, 53, 386
  addUrl, метод, 58
  назначение, 54
FileListNewsDataset, класс, 389, 398, 404
findSimilarUsers, метод, 128
FN, 307, 310, 320
FoodieBytes, 39
FP, 307, 309, 310, 320
FPR, 307
FraudErrorEstimator, класс, 285, 288
Friendster, 32
F-балл, 307
F-распределение, 354
F-статистика, 353
F-тест, 325, 346, 352, 353

G

GATE, 443
generated-test-txns.txt, файл, 282
getNoisyType, метод, 334
Global Positioning System (GPS), 282
GoodUserType, класс, 332
Google, 66
  News, 391
  Top stories, 383
  главные новости, 383
  матрица, 70
  начало эры интеллектуальных приложений, 24
Google Finance, служба, 25
Google maps, служба, 38
Google News, служба, 25, 32, 382, 383
Google-алгоритм PageRank, *см.* PageRank, 67
GroupLens, лаборатория, 162

H

Hadoop, проект, 104
HDFS, 105
hi5, сайт, 32
higher-order, эффекты, 174
Hits, объект, 60, 67
HousingMaps, 38
HTML, 61, 387
  парсер, 61
Hypertext Induced Topic Search, алгоритм, 67
Hyves, сайт, 32

I

IBM DeveloperWorks, 31
IETF, 40
if-then, операторы, 243
imeem, сайт, 32
Income, атрибут, 330
indexDocument, метод, 58
IndexWriter, класс, 57
information retrieval, IR, процедура, 51, 106
  традиционные шаги, 52
Instance, объект, 250
Integrated Development Environment, IDE, 19
Item, объект, 115
ItemBasedSimilarity, класс
  calculate, метод, 139
Items, объект, 418

J

JaccardCoefficient, класс, 215, 264, 298, 318
jaccardThreshold, переменная, 264
Janino, встроенный Java-компилятор, 273
JAR, расширение файла, 128
Java, встроенный компилятор Janino, 273
Java Development Kit, JDK, 429
JavaScript, 41
Java-архив, *см.* JAR, 128
JAX-WS 2.0, 42
JAX-WSA, 42
JBoss, процессор правил
  Drools, 236
  Rules, 243, 267
Jess, программа, 243, 267
JFlex, 63
JobClass, 330
JSON, 23, 41
JSR-181, 42

K

Kruskals, алгоритм, 206
k-ближайших соседей алгоритм (kNN), 245

k-средних++, 211
k-средних, алгоритм, 190, 206

L

L2-норма, 438
LGPL, 94
LinkedIn, сайт, 32
LinkMatrix, класс, 215
linkThreshold, переменная, 411
Lp-норма, 438
Lucene, программа, 387, 391
  выражение запроса, 64
  классы
    Document, 58
    Field, 58
    QueryParser, 60
    StandardAnalyzer, 63
  объекты
    Document, 62
    Query, 64
  оценка документа, 391
  повышение степени важности, 64
  поиск, 52, 59, 65
  файлы индексов, 54
LuceneIndexBuilder, 54, 110
Lucene, PageRank и NaiveBayes
  объединение оценок, 78, 80–82, 90
LucenePDFDocument, класс, 62, 387
Lucene-объекты
  разнообразие, 64
  удаление и обновление, 64
Lucene-текст, анализаторы, 63
LVQ, 245

M

MapReduce, 104
mashup, тип сайта, 31
maxBatchSize, 58
McNemarTest, класс, 347
McNemar, класс, 350
MegaUpload, сайт, 34
mergeClusters, метод, 202
MergeGoodnessMeasure, класс, 216
metaclassifier, схема, 247
Microsoft, 42
Microsoft OLE 2 Compound Document, 63
Microsoft Word, 61
  версии, 63, 387
  документы, 94
  парсер, 62
  синтаксический разбор документов, 94
MinorThird, 444
MotorcycleOwnership, 331
MovieLens, 162
MovieLensData, 162
MovieLensDelphi, класс, 163, 169, 172
MovieLensItemSimilarity, класс, 169
minimum spanning tree, MST, тип алгоритмов, 190, 203
  findMinimumEdge, 206
  алгоритмы, 231
    Boruvka_fs, 206
    Kruskal_f, 206
  рандомизированный алгоритм, 231
  цепочечный эффект, 206
MSTSingleLinkAlgorithm, класс, 196
MST-алгоритм связи
  временная сложность, 206
MSWordDocumentParser, 94
multiclass, принцип классификации, 241, 248
MusicData, 128
MusicUser, 116, 118
  getSimilarity, 117
MusicUser, класс
  getSimilarity, метод, 120
  plot, метод, 120
MyDiggSpace.com, 150
  конкретный случай, 151
  статистика данных, 153
MyDiggSpaceDataset, 212
MySearcher, класс, 59, 64, 89
MySpace, 25, 30, 32
MySQL, 183

N

NaiveBayes, 85, 88, 89, 260
  классификатор, 85, 401, 404
NaiveBayes, класс, 250, 251, 255, 316, 338
NBCreditClassifier, класс, 338, 358, 367
NBLanguageDetector, классификатор, 388
NBStoryClassifier, класс, 401, 402, 404, 407
nearest neighbor, метод, 190
neighbor selection, метод, 162
neighbor weights, метод, 163
NekoHTML, 61, 387
Netflix.com, сайт, 26, 142, 162, 163
  Cinematch, 26
  выбор фильмов, 161
Netscape, Rich Site Summary, 40
Neural Networks, NN, тип алгоритмов, 243, 245
  архитектура, 285

NewsCategory, объект, 404
NewsClusterBuilder, класс, 411
NewsCrawler, класс, 385, 386, 422
NewsDataset, интерфейс, 389
NewsDataset, класс, 398
NewsDataset, набор данных, 409
NewsProcessor, класс, 391, 402, 407
  этап обучения, 400
NewsStory, объект, 404
Niemeyer, Pat, 429
Ning, сайт, 32
NLP, 148, 385, 387
NNCreditClassifier, класс, 343, 358, 367
NNFraudClassifier, класс, 285, 287, 291, 294
no-loop, атрибут правила, 277
null hypothesis, предпосылка, 346
N первых часто встречающихся термов, 254

O

OASIS, 443
Object Management Group, *см.* OMG, 241
Occupational Injury and Illness Classification (OIIC), документ, 237
Octave, язык программирования, 446
OLE 2, 387
OMG, 241
OpenSocial, исходная предпосылка, 32
Oracle, 32, 42
Orkut, социальная сеть, 32, 126

P

Package, класс, 273
PageRank, 67, 81, 383, 392
  alpha, коэффициент, 70, 71
    выбор, 73
    влияние на сходимость, 73
  вектор, 68
  висячие узлы, 70, 110
  влияние эпсилон, 78
  вычисление, 71
  декремент, 71
  ключевая идея, 67
  масштабирование, 110
  матрица гиперссылок, 68
  методика квадратичной экстраполяции, 104
  поправка примитивности, 71
  приближенные методики агрегирования, 104
  прямые методы (решатели), 73
  степенной метод, 67, 69, 73
  стохастическая поправка, 70
  сходимость и уникальность, 70
  шкала оценок, 81
  экстраполяция Айткена, 104
  эффект телепортации, 70, 71, 111
PageRankMatrixH, класс, 73
PDF, 61, 387
  документы, 94
  индексирование, 61
PDFBox, 61, 387
PearsonCorrelation, класс, 168, 169
  ошибка округления, 176
PhraseQuery, класс, 64
  параметр slope, 65
phylum, атрибут, 238
Plaxo, сайт, 32
PorterStemFilter, класс, 254
PredictedNewsStoryRating, класс, 421
PredictWallStreet, 26
ProgrammableWeb, сайт, 31
PropertyOwnership, атрибут, 331

Q

quality assurance, QA, процесс, 105
QueryParser, класс, 64
Q-тест Кохрана, 325, 346, 352–354

R

Random, класс, 336
Rank, 76
  оценка ошибки, 78
RapidShare, файлообменник, 34
Rating, атрибут, 115
RatingCountMatrix, 132, 141
RDF, модель, 40
RDF Site Summary, 40
Real-time blackhole lists, RBL, 250
RecommendationType, тип рекомендаций, 421
REST, алгоритм, 41
Rete, алгоритм, 243, 268, 271
ReteOO, 268
RetirementAccounts, атрибут, 331
RFC 4287, документ, 40
RMSE, 172, 176
RMSEEstimator, класс, 172
ROC–графики, 309
ROCK, алгоритм, 191, 211, 231, 232, 408, 413
  детализация, 215
  ключевая идея, 213
  критерий завершения процесса, 215
  критерий качества, 215

настройки алгоритмов, 427
объяснение формулы, 217
структура связей, 216
этап инициализации, 215
ROCKAlgorithm, класс, 212, 213, 411, 427
RSS, 39
версия 2.0, 40
RuleEngine, класс, 317
RuleQuest, 244
runtime, этап, 273
R-деревья, 227
r Пирсона, 165
контрпример, 167

S

SAAJ, 42
Salesforce, 32
SAP, 42
SAS, 210
ScanScout, 34
SearchResult, объект, 60
See5, 244
selectBestMatchingTopic, метод, 406, 407
selectLongestStory, метод, 406, 407, 426
selectRepresentativeStory, метод, 426
self-organizing maps, SOM, 319
Semantic Web, 240
setTopTerms, метод, 404
SFDataset, класс, 196
SingleLinkAlgorithm, класс, 196
Six Apart, 32
SOAP, 41
версия 1.1, 42
версия 1.2, 42
Soar, 243
SOII, 238
SortedArrayClustering, 186
SourceForge.net, платформа, 181
Spam Assassin, 250
spamRules.drl, 269
spamRulesWithConflict.drl, 279
sparse, матрица, 69
StandardAnalyzer, класс, 63, 145, 254, 387
StandardTokenizer, класс, 63
StoryRecommender, класс, 420, 421
SVG, 23
SVM, 44
Swing, 399
клиент, 409, 410

T

test-users.txt, файл, 337
TextMining, принцип, 63, 94, 387
см. также tm-extractor, 94
Tianji, сайт, 32
title, поле, 58
tm-extractor, библиотека, 94
TN, 307
TopicalNewsClusterBuilder, класс, 416, 428
TP, 307, 309
TP rate, TPR, 309
TrainingSet, класс, 85, 88
TrainingSet, набор, 250, 253
TrainingSet, объект, 404
training-txns.txt, 286
training-users.txt, 337
trainOnAttribute, метод, 259
TransactionDataset, класс, 286
TransactionInstanceBuilder, класс, 294, 298
TransactionNN, класс, 287, 291, 294, 295–298, 318
True Negative, TN, 307
True Positive,TP, 307

U

UI, 16
UIMA, 443
url, поле, 58
UseCaseData, createUserTypes, метод, 335
UseCaseData, класс, 356
UserBasedSimilarity, 130
user-clicks.csv, файл, 83, 423
UserClick, класс, 85
UserContentBasedSimilarity, класс, 144
UserCreditNN, класс, 343, 345, 376
UserInstanceBuilder, класс, 339, 340, 360
UserStatistics, класс, 287
UserType, метод, 335

V

VeryGoodUserType, класс, 332
Viadeo, программа, 32
VLDB, алгоритм, 193

W

WaveCluster, 191
WhitespaceAnalyzer, класс, 63
WHO, 238
Wikipedia, 33

World Health Organization, WHO, 238
WS-Addressing, 42
WS-I Basic Profile, 42
WS-Policy, 42
WS-RM, 42
WS-Security, 42

X

Xerox
  Palo Alto Research Center, 126
XForms, 23
Xignite, финансовые веб-службы, 42
XING, 32
XML, 41, 61, 387
XORNetwork, класс, 297
XPath, 23
XSLT, 23
XUL, 23

Y

Yahoo!
  RSS-каналы, 40
YouTube
  общий доступ к медиафайлам, 34

Z

z-статистика, 352

А

абстрактное синтаксическое дерево, *см.* AST, 273
автоматическая категоризация, 249
агломеративно-иерархические алгоритмы, 190
агрегированный контент, 28
адаптивная повторная выборка, 365, 379
акции, прогнозирование, 26
алгоритмы, 28
  arc-x4, 370
    четвертая степень, 373
    ядро, 372
  BIRCH, 193
  Boruvka_fs, 206
  DBSCAN, 218
  EM
    E-шаг, 231
    M-шаг, 231
  KISS, 47
  Kruskal_fs, 206
  k-средних, 206, 232, 413
    pickInitialMeanValues, 210
    центроиды, 210
    ядро алгоритма, 208
  Rete, 271
  ROCK, 213, 408
  SQLEM, 231
  агломеративно-иерархические, 190
  ближайших соседей, 190
  дивизимно-иерархические, 190
  классификации, 235
  кластеризации, 410
    BIRCH, 227
    k-средних, 190
    комбинации, 232
    максимизации ожиданий, 231
  контекст приложения, 381
  максимизации ожиданий, 231
  масштабируемость, 46
  наивный байесовский, 246
  на основе плотности, 218
  ограничения применения, 47
  обратного распространения, 304
  обучения по методу градиентного спуска, 304
  одной связи, 198
    MST, 203
    вычислительная сложность, 199
    порог близости, 198
    цепочечный эффект, 206
  оценка времени вычисления, 47
  плоские, 190, 206
  Прима–Ярника, 205
  пространственной кластеризации на основе плотности для приложений с шумами, 218
  ранжирования, 392
  распараллеливание, 46
  регрессионные, 245
  связей, 193, 196
    визуализация, 198
    сравнение, 203
  средней связи, 200, 202
  структурные, 244
  теория графов, 203
  фильтрующей кластеризации, 191
анализ, 61, 387
  экранных данных, 39, 52, 66
  текста, лексический, 63
    лексическая неоднозначность, 148
    морфологический поиск, 145
анализаторы
  используемые по умолчанию, 63
  Lucene, 63
    другие языки, кроме английского, 64

стоп-слова, 63
текст, 63
аналогия с файловой системой
атрибуты, 85
концепты, 85
образцы, 85
ансамбль классификаторов, 362
инкрементное наращивание, 365
показатель правильности, 358
апостериорная вероятность, 86, 260
эвристика, 88
априорная вероятность
эвристика, 89
арифметика с плавающей запятой, 176
Армстронг, Лэнс, 53, 249
архитектура нейронной сети, 285
атрибуты, 239
Drools
no-loop, 277
ruleflow-group, 277
salience, 277
классификации
последствия большого количества, 238
кластеризации
возраст, 181
диапазон годового дохода, 181
образовательный уровень, 181
оплачиваемое участие, 182
профессиональное мастерство, 181
правил, 277
аттестация поиска
график точность/выборка, 107
ациклический направленный граф, 284

Б

базовые рекомендатели, 161
байесовские сети, 246
доверительные, 258
нейронные, 446
банкротство, 328
значимость, 328
Бернулли процесс, 307
библиотека, шифр, 237
ближайших соседей, алгоритм, 190
близость, 115
относительная, 196
порог, 197, 198
Блэка–Шоулза модель ценообразования опционов, 446
большая размерность, 228
большие базы данных
свойства алгоритмов, 192
Борувка, Отакар, 231
Брейман, Лео, 365
Брин, Сергей, 67, 103
бритва Оккама, 316
броузер новостей
окно, 391
создание и вывод на экран новостей, 398

В

варианты использования алгоритмов, 36
веб-роботы, 433
веб-сайты
логический вывод, 24
синдикация, 39
способность к обучению, 24
веб-службы, 42
векторы, 437
обучения, 446
персонализация, 111
термов, 99
вероятности
апостериорные, 88
априорная, 258, 259
условная, 258, 259, 260
вероятностное распределение, 348
вероятность, 258
существования связи, 427
на входе теоремы Байеса
априорная вероятность, 86
подтверждение, 86
на выходе теоремы Байеса
апостериорная вероятность, 86
перехода матриц, 69
вес соседа, 163
вид, 238
визуализация, 338
Википедия, 33
висячие узлы, 68, 70, 103, 393
эвристика, 111
владение
автомобилем, 328
землей, 331
внешние ссылки, 54, 68
внутреннее произведение, 439
апплет, 439
возможности портала
помеченные, 384
возможность добычи данных, 384
возраст, 327
волновая кластеризация, 191

временная метка, 84
встраивание интеллекта, 36
выбор атрибутов, 375
    новостной портал, 424
выборка, 106, 108
    оценка распределения, 355
    с замещением, 355
выбор соседей, 162
выбор статистики, 347
вычисления
    r Пирсона, 166
    векторного произведения, 166
    сложность, 226
    стоимость, 428
вычислительные
    кластеры, 105
    лингвистика, 442
    нейронные сети, 283
выяснение языка, 388

Г

Габов, Гарольд, 231
Галил, Цви, 231
гарантия качества, 400
гауссово распределение, 281, 362
гауссовы процессы, 258, 446
генератор меток, 63
геометрия на плоскости и в пространстве, 126
гибридные веб-приложения, 31
    агрегированный контент, 31
    определение, 31
Господнетик, Отис, 52
графы
    алгоритмы теории, 203
    направленные, 67
группы новостей, 413
Гуха, Раманатан, 232

Д

данные
    важность понимания, 291, 382
    двоичные, 34
    для обучения, 247
    достоверность, 45
    изменчивость, 45
    несопоставимые, 45
    неточности репрезентативной выборки, 45
    нормализация, 45, 225, 286
    перенормировка, 171
    подобные данным сайта SourceForge, 196
    предварительная обработка, 286
    проблемы, связанные с размером, 46
    пропущенные значения, 45
    разброс, 364
    сжатие, 227
    с шумами, 358
двоичная классификация, 248, 252
двоичные данные, 34
декларативное программирование, 266
дендрограмма, 193
    визуальное представление, 194
    две связанные хеш-карты, 194
    инициализированная, 202
    структура данных, 193, 196
дерево решений, 243, 340, 356, 375
    алгоритмы, 244
    классификатор, 325, 366, 368
    правильность, 341
десятикратная перекрестная проверка, 308
диагностика
    заболеваний, 237
    травм, 237
диапазон значений, 175
дивизимно-иерархические алгоритмы, 190
дискурс, 393, 443
доверительный интервал, 308
документ
    термы, 391, 393
    эвристическая важность, 99
допустимые транзакции, 287
допущение о независимости, 352
достоверность классификации, 305

Е

евклидово расстояние, 123, 187, 191, 210, 229, 438
естественные языки, обработка, 148

Ж

Жаккард
    коэффициент, 191, 212, 264
    метрика, 132, 141
    показатель сходства, 191
    сходство, 125, 174, 215

З

заблуждения и интеллектуальные приложения, 44
заказ еды, 24
закон Бенфорда, 282
залог жилой недвижимости
    кредитные линии, 328

запрос
google ads, 84
контекст, 392
термы, 393
зашумленные данные, 358
значение, 278

И

игры онлайн, 35
иерархическая кластеризация, 189
иерархические ссылочные структуры, 239
иерархический агломеративный алгоритм, 191
изменчивые классификаторы, 356
индексирование, 52, 61, 387
поиск вне индексов, 65
этап, 64
интеллект, 27
в отличие от координации совместной деятельности, 26
встраивание, 36
коллективный, 27
контрольная точка, 67
краулинг, 422
построение, 37
потенциал объединения, 413
применение, 382
треугольник интеллекта, 29
интеллектуальная система подготовки документов, 27
интеллектуальное приложение
заблуждения, 44
предпосылки, 36
элементы, 28
интеллектуальный поиск, 393
Интернет
поведенческая характеристика, 66
структурная характеристика, 66
семантическая интерпретация, 40
интернет-серфер, случайный, 69
интерполяция, 319
информационный поиск, *см.* IR, 51
ипотека, 328
авансовый платеж, 329
финансирование, 326
приложение, 325
ипотечные ставки
реклама, 326
«исключение-по-одному», 309
искусственные данные, 335
создание, 332
искусственный интеллект, 43
проблема эффективности, 268
итеративная оптимизация, 190
календарный возраст, 327

К

Каргер, Дэвид Р., 231
категоризация
автоматическая, 249
электронной почты, 248, 252, 264
категории, 235
интернет-конференции, 237
меню ресторана, 237
новостей, 414
статьи в газете, 237
категорийные данные, 191
качество поиска, 105
показатели, 106
квантизация векторов при обучении, *см.* LVQ, 245
квартет Энскомба, 167
классы, 238
Edge, 205
IndexSearcher, 60
McNemarTest, 347
McNemar, 350
MergeGoodnessMeasure, 216
MST, 204
NaiveBayes, класс, 250, 251, 255, 316, 338
NewsClusterBuilder, 411
NewsCrawler, 385, 386, 422
NewsDataset, 398
Package, 273
PageRankMatrixH, 73
PhraseQuery, 64
PorterStemFilter, 254
QueryParser, 64
Random, 336
RMSEEstimator, 172
ROCKAlgorithm, 212, 213, 411, 427
ROCKClusters, 215
SingleLinkAlgorithm, 196
StandardAnalyzer, 63, 145, 254, 387
StandardTokenizer, 63
StoryRecommender, 420, 421
TransactionDataset, 286
TransactionInstanceBuilder, 294, 298
UseCaseData, 356
UserClick, 85
UserContentBasedSimilarity, 144
UserCreditNN, 343, 345, 376
UserInstanceBuilder, 339, 340,
UserStatistics, 287
VeryGoodUserType, 332
XORNetwork, 297

классификаторы, 85
ансамбль, 362
дерево решений, 366
изменчивые, 356
на основе нейронной сети, 343
объединение, 323, 362
отбор, 355, 377
подбор, 323
попарные сравнения, 347
слияние, 323
сравнение, 325
стадии жизненного цикла, 246
устойчивость, 340
чувствительность, 291
этап обучения, 246
этап тестирования, 246
классификация, 85, 395
алгоритмы, 235, 236
на основе правил, 243
на основе расстояний, 243
влияние порядка выполнения, 394
время выполнения, 345
время обучения, 345
групп образцов, 401
новостных сообщений, 402
репрезентативное новостное сообщение, 401
двоичная, 241, 248, 252
дискретные величины, 241
зона влияния, 324
иерархическая структура классов, 241
категории новостей, 401
Линнея, 238
многоклассовая, 241, 248, 252
на основе правил, 266
нейронные сети, 236, 243
непрерывные величины, 241
обзор, 241
обобщение против специализации, 252
ошибочная, 338
перекрестные ссылки, 413
плоская структура классов, 241
понижение уровня зашумленности контента, 400
последствия неверного решения, 310
правило принятия решения большинством голосов, 402
правильная, 338
проблема эффективности, 313
прогнозирование, 241
производительность этапа выполнения, 312
проявления шума в данных, 289
регрессионные алгоритмы, 242
репрезентативное новостное сообщение, 406
с помощью дерева решений
изменчивость, 343
интерпретация, 342
специализация против обобщения, 252
статистические алгоритмы, 242, 245
стоимость, 307
стратегия, 394
структурные алгоритмы, 242, 243
характеристики производительности, 345
электронной почты, 317
белые списки, 250
списки-карцеры, 250
тесты заголовков, 250
характерные особенности, 249
черные списки, 250
классификация-кластеризация, 400
классы деятельности, 330
кластер, 394
инвариантность, 415
обнаружение, 184
структура, 189
кластеризация, 395
DBSCAN, 218
MST, 203
ROCK, 213
R-деревья, 227
VLDB, 193
алгоритмы, 410
BIRCH, 193
k-средних, 190, 206
SQLEM, 185
агломеративно-иерархические, 190
дивизимно-иерархические, 190
иерархические, 189
на основе плотности, 218
одной связи, 198
связей, 196
средней связи, 200
анализ сайта, подобного Sourceforge, 181
базы данных большие и сверхбольшие, 192
волновые методы, 190

кластеризация, 395
высокая размерность, 225, 228
вычислительная сложность, 226
дендрограммы, 193
евклидово расстояние, 187
иерархическая, 427
итеративная оптимизация, 190
и упорядочение, 183
категоризация, 188
категорийные данные, 191
концептуальное моделирование, 190
критерий качества, 217
наглядная идентификация, 183
несколько измерений, 188
новостные статьи, 189
анализ, 410
нормализация данных, 187
обзор, 188
ограничения SQL-подхода, 184
отсутствие нормализации, 196
очень большие наборы данных, 226
параметр сдвига, 189
плоские алгоритмы, 190
плотность точек, 218
по возрасту, 187
по размеру обрабатываемых данных, 192
порог близости, 198
по структуре данных, 190
по структуре кластеров, 189
по типу данных, 189, 190
применение, реклама, нацеленная на конкретную аудиторию, 180
пример с книгами, 179
произвольные объекты, 188
проклятие размерности, 228
сжатие данных, 227
синглтоны, 204
сортировка массива, 185
спектральные методы, 190
среднее значение, 207
среднее расстояние, 200
тонкая настройка, 428
фильтрующие алгоритмы, 191
характеристики производительности, 226
цель, 217
центроид, 207
эксперт-человек, 187
эпсилон-окрестность, 221
язык запросов SQL, 183
кластеризация-классификация, 400
кластерные образования, 204
хорошее качество, 427
Клейнберг, Джон, 67
Кляйн, Филипп Н., 231
количество информации, 315
коллаборативная фильтрация, 126, 148, 156, 158, 162, 418
коллегия выборщиков в США, 402
коллективное знание, фиксация, 27
коллективный интеллект, 27
компаратор, 186
контент
агрегатор, 30, 33
аннотировать, 33
загрязнения, 388
синдицированный, 39
согласование, 32
чистка, 388
контент электронной почты
акции фирмы NVidia, 249
внутренняя политика США, 249
выборы в конгресс, 249
выборы в Никарагуа, 249
глобальное потепление, 249
Лэнс Армстронг, 249
марафон, 249
международные новости, 249
Ортега, 249
реклама в газетах, 249
спам, 249
концептуальное моделирование, 190
концепты, 239, 258
электронной почты
NOT SPAM, 252
SPAM, 252
координация совместной деятельности
в отличие от интеллекта, 26
корреляция
строго отрицательная, 167
строго положительная, 167
косинусная мера сходства, 146, 439
коэффициенты
FP, 307
Жаккарда, 191
затухания, 71
корреляции со смешанным моментом, 165
обучения, 302
ранговой корреляции Спирмана, 168
Танимото, 174
краулинг, 54, 61, 382, 385, 386, 432
Apache Tika, 435
Heritrix, 435
Nutch, 435

пользовательский поисковый робот, 434
структура извлеченного контента, 386
кредитные карты, использование, 328
кредитный балл, 328
кредитоспособность
атрибуты, 327
категории, 327
конкретный случай для анализа, 324
обзор, 325
критерии
завершения работы, 190, 215
качества, 217, 231
сходимости, 69
крошки, 126

Л

лексемы, частота появления, 99
лексикографическое упорядочивание, 191
лексический анализатор, 63
линейная регрессия, 244
линейная корреляция, коэффициент, 165
Ллойд, С. П., 210
ловушка черного ящика, 289
логистическая регрессия, 245
логистическая функция, 245
логический вывод, 24
ограничения времени отклика, 46
логический элемент XOR, 297

М

Макнемара тест, 325, 346, 376
массивы, сортировка, 185
масштабная чистка, производительность, 390
масштабный поиск
вычислительные ограничения, 102
структуры данных, 103
точность PageRank, 103
математические формулы, 437
матрица, 437
вероятность перехода, 69
неточностей, 306, 337, 358, 377
разреженная, 69
смежности, 196, 197
стоимости, 320
сходства, 197
верхняя треугольная матрица, 132
матрица H
документы Word, 97
доля висячих узлов, 75
доля основных ссылок, 75
доля телепортаций, 75
симметричное переупорядочивание, 103
субстохастический вариант, 75
машинная точность, 176
машины Больцмана, 445
международные новости, 389
мера, 120
метаалгоритм, 382
метаданные и веб-страница, 54
метрика кратчайшего пути, 191
метрические пространства, 440
механизм телепортации, 67
минимальное остовное дерево, *см.* MST, 190
многоклассовая классификация, 248, 252
сложность, 312
многомерные данные, упорядочивание, 185
модуль сопоставления с образцом, 268
мотивация с точки зрения представления, 324
мошенничество
TenUsersSample, класс, 282
аукционы в Интернете, 280
аферы в сфере телекоммуникаций, 280
обнаружение, 318
сулящие выгоду регистрационные формы и анкеты, 280
транзакции, связанные с приобретением товаров, 281
идентификация, 287
транзакционные данные, 282
мэшапы, 31

Н

набор данных
MovieLens
RMSE, 173
малый, 164
размерность, 225
набор документов
Лэнс Армстронг, 53
международные новости, 53
политика США, 53
наивные байесовские алгоритмы, 85, 388, 394
классификация, 246

надежность, 89
происхождение термина, 260
наивный байесовский классификатор, 82, 337, 368, 401
написание сценариев, 429
направленный граф, 67
ациклический, *см.* DAG, 246
неверно классифицированные новостные сообщения, 400
невозврат кредита, 326
недооценка, 315
нейронные сети, 236, 243, 245, 325, 356, 445
BaseNode, класс, 302
calculateWeightAdjustments, метод, 305
connectFully, метод, 302
fireNeuronDerivative, метод, 302
fireNeuron, метод, 302
learningRate, 302
LinearNode, класс, 302
SigmoidNode, класс, 302
updateWeights, метод, 305
архитектура, 285
классификатор кредитоспособности, 343
комплексные, 446
коэффициент обучения, 302
недостатки, 245
обзор, 283
полностью связанные уровни, 285
прямого распространения, 284
связи, 302
сложность проектирования, 345
с обратным распространением, 284
структура, 301
существенные элементы, 285
уровни, 284
этап обучения, 284
нейроны, 245, 283
некоррелированные предметы, 167
непараметрическая корреляция, 167
непараметрический метод, 355
несанкционированная массовая рассылка, *см.* спам, 248
Нимейер, Пэт, 429
новости
категории, 405
распределение, 394
контент, 384
портал, 382
тема, 405
Новости Google, сервис, 382
Новости Digg, сервис, 212
новостная группа
сопоставление, 399
устойчивость кластеризации, 400
новостные сообщения, 382
неверно классифицированные, 400
поиск, 382
систематизация, 398
нормализация данных, 286
PearsonCorrelation, 168
нормальное распределение, 336
нуль-гипотеза, 346

О

обвал ипотечного рынка США, 326
обнаружение мошенничества, 280
погрешность, 298
скрытый уровень, 298
сценарий, 281
обобщение, 316
обработка естественных языков, 52, 148
образец, 258
обратная цепочка рассуждений, 267
обратного распространения алгоритм, 304
обучение
без учителя, 188
исключение-по-одному, 309
по Хеббу, 319
самонастройка, 309
с учителем, 306
полуэмпирический подход, 311
обучение классификации
параметры масштабирования, 312
покрытие значений атрибутов, 311
репрезентативные данные, 311
статистическая оценка, 311
обход контента, 39, 54
глубина, 39
поисковым роботом, 432
этапы, 433
объединение в группы, расхождение, 400
объединение классификаторов, 323, 362
bagging, 325
boosting, 325
вычислительная устойчивость, 324
преимущество представления предметных сущностей, 324
снижение риска, 324
объединение рекомендателей на основе усреднения, 158
объекты, 239

окруженная точка, 222
онлайн-игры, 35
онтология, 236, 239
аналогия с ООП, 239
атрибуты, 239
инженерия, 236
концепты, 239
объекты, 239
пример, 239
семантика, 240
основные ярлыки, 30
остовное дерево, 203
отклонения Кульбака–Ляйблера, 210
отклоняющиеся значения, 230
отличия в показателях правильности, 347
относительная близость, 196
относительная оценка
ранжирование, 154
отрицательная классификация, 306
отряд, 238
оценка
десятикратная перекрестная проверка, 308
индексированная страница, 81
релевантности, 61, 391
обобщение, 142
объединение, 111
рекомендаций, 172
ссылок, 25, 66, 67
документы, 94
ссылка, 52
ошибки
I типа, 307
II типа, 307
RMSE, 176
округления, 166
величина, 176
минимизация, 176
ошибочная классификация, стоимость, 338

П

параметр сдвига, 189
парсер, 432
пауки, 39
Пейдж, Ларри, 67
пенсионные выплаты, 331
перекрестные ссылки, 413
перенормировка данных, 171
переобучение, 252, 315
переходы пользователей по ссылкам, 393
новостной портал, 423
персонализация, 83, 384
вектор, 111
поиска, переходы пользователя по ссылкам, 82
платформы для совместных действий, 28
плоские ссылочные структуры, 239
плоский алгоритм, 206
плотности на основе, алгоритмы, 218
плотность точек, 218
повышение степени важности
стратегия, 110
пограничная точка, 222
погрешности, 298
подтверждение, 258
подход «классификация-кластеризация», 400
поиск, 51, 61, 386, 387
PageRank, 67
вне индексов, 65
основные этапы, 61
оценки
Lucene и PageRank, 78
релевантности, 61
ссылок, 66
проблемы масштабирования, 102
проверки правильности
выборка, 106
точность, 107
релевантность, 82
ссылочное кодирование, 103
цель, 67
этап анализа, 63
поисковые системы
настройка, 106
поворотный момент в истории Интернета, 25
поисковый робот, 39, 385
известные URL-адреса, 54
извлеченные документы, 54
компоненты, 432
обработанные документы, 54
пользовательский, 110
сбор данных, 52
специализированный, 385
ссылки на страницы, 54
показатель правильности классификации
статистически незначащий, 357
полином Чебышева, 104
полнота, 307
положительная классификация, 306

пользователь, 115
помехи, 191
попарные сравнения классификаторов, 347
поправка
  примитивности, 71
  стохастическая, 70
пороговое значение, статистический тест, 346
порождающие правила, 243
порталы, рассредоточенный агрегированный контент, 32
порядок операций, 395
постинги, 30
построение интеллекта, 37
правила, 267
  AccumulateFunction, класс, 274
  ChainedProperties, класс, 274
  ClassificationResult, класс, 269
  Dialect, класс, 274
  Email, класс, 269
  global, предложение, 269
  isSpamEmail, 270
  Package, экземпляр класса, 273
  PackageBuilderConfiguration, класс, 274
  PackageBuilder, класс, 274
  RuleBase, класс, 273
  RuleEngine, класс, 271
  salience, атрибут, 277
  StatefulSession, 274
  WorkingMemory, класс, 274
  активизации, 285
  обучения, 285
  процессор, 236
правильность, проверка, 307
прагматика, 393, 443
предварительная обработка, 54
предметная область рассуждения, 28
предоставление кредита, риск, 325
предпосылки для интеллектуальных приложений, 36
представление знаний, 236
представления о мире, 235
приближенные методики агрегирования, 104
приз Netflix, конкурс, 175
применение кластеризации
  индивидуумы с похожими мнениями, 180
  создание социальных сетей, 180
пример выдачи рекомендаций, 114
  музыкальный интернет-магазин, 127
проблема эффективности, 268, 313
проверка фактов, 24
прогнозирование, пример, 241
программирование
  декларативное, 266
  процедурное, 266
проект Apache POI, 62
прозрачное обучение, алгоритмы, 44
проклятие размерности, 226, 228, 238, 312
Пролог, язык программирования, 267
пропущенное значение атрибута, 260
просмотр, 73
процесс Бернулли, 307
процессор правил, 266
  исполнение, 273
  подготовка, 273
прямая цепочка рассуждений, 267
прямые методы, 73

Р

размерность, проклятие, 226
разметка
  адреса электронной почты, 63
  акронимы, 63
  буквы и цифры, 63
  имена компьютерных хостов, 63
  китайские, японские и корейские иероглифы, 63
  строк, 254
  текст, 63
разность пропорций, тест, 325
разреженные матрицы, 69
разрешение конфликтов, 277, 278
ранговая корреляция, 176
  Спирмана коэффициент, 177
распознавание, 34
  зрительных образов, 218
  речи, 34
распределение
  c2, 353
  кластеров, 414
    оптимальность, 427
  Фишера–Снедекора, 354
  хи-квадрат, 354
распределенная файловая система Hadoop, *см.* HDFS, 104
распределенные вычисления
  заблуждения, 44
расстояние
  L2, 438
  городских кварталов, 438
  евклидово, 438

неравенство треугольника, 119
свойства, 118
таксистов, 438
рассуждения, основанные на прецедентах, 266
ребро, 203
регрессионные алгоритмы, 242, 245
регулируемая ипотека, 326
рейтинговая оценка, 115
рекламные ставки, 326
рекомендатель, 143
рекомендации, 25
в реальном времени, 171
качество, 171
минимизация ошибки округления, 176
обновление в рабочем режиме, 172
оценка, 172
стоимости, 158
релевантность, субъективность, 82
решетки, 190
Ричардсон, Леонард, 41
род, 238
трудовой деятельности, 330
Руби, Сэм, 41

С

сайты общего доступа к медиафайлам
двоичный формат, 34
сайты социальных сетей
два наиболее посещаемых, 30
самонастраиваемое объединение, 325
самонастройка, 309
самоорганизующиеся карты, 319
сборка мусора, 102
связи, 283
определение, 216
связующее ПО, 267
связывания, 42
сдвиг, 189
семантика, 393, 443
семантическая онтология, 240
семейство, 238
семиотика, 278
сериализованный PHP, 41
сеть, топология, 298
синапсы, 245, 283, 304
вес, 284
синглтоны, 204, 212, 400, 410, 414
большое число, 411
синдицированный контент, 39
синтаксический разбор, 61, 386
синтез речи, 442
система выработки рекомендаций
ансамбль методов, 175
вопросы масштаба, 171
выбор соседей, 162
на основе сходства пользователей, 127
новостные сообщения, 418
нормализация данных, 162
нормализация оценок, 157
оптимизация программного кода, 138
сходство, 115
сходство предмета на основе контента, 151
элементарные концепты, 418
система классификации
CFOI, 238
ICD-10, 238
OIIC, 237
SOII, 238
библиотека Конгресса, 237
Линнея, 238
Щацкер, 237
систематическая ошибка в противовес обобщению, 48
скалярное (внутреннее) произведение, 147, 439
скачок, 70
слияние решений, 323
сложность многоклассовой классификации, 312
случаи отчуждения заложенного имущества, 326
случайные выборки, 355
случайный интернет-серфер, 69
смежность
матрица, 196
узлы, список, 103
смесь экспертов, 355, 377
сообщения электронной почты
сортировка, 248
соседи
вес, 163
выбор, 162
спам, 248
документы, 95
страницы, 65
фильтрация, 248
спам-фильтры, 64
спектральная кластеризация, 191
спектральный анализ, методы, 104

Спенсер, Томас Х., 231
специализация, 315
специфичность, 307
спортивные новости, 389
среднее значение, *см.* центроид, 207
среднее расстояние, 200
среднеквадратическое отклонение, *см.* RMSE, 166, 172
ссылочное кодирование, 103
ссылочные структуры, 28, 236
базы знаний, 28
онтологии, 28
словари, 28
стандартное отклонение, 166
стандартное нормальное распределение, 352
статистика
q, 353
хи-квадрат, 348, 376, 377
статистическая значимость, 346
статистические алгоритмы, 242
статистические данные о расстояниях между точками, 225
степенной метод, 69
ускорение, 73
число итераций, 104
степенные зависимости, 81
степень
доверия, 308, 310, 320
свободы, 348, 353, 354
стоимость
матрица, 320
функция, 310, 320
стоимость классификации, 307
стоп-слова, 63
стохастическая поправка, 70
стратегии отбора победителя, 363
структурные алгоритмы, 242, 244
ближайших соседей, 244
функциональные, 244
численная аппроксимация, 244
структуры
иерархические ссылочные, 239
плоские ссылочные, 239
судимость, 329
сумма дохода, 330
суперкластер, 411, 414
разбиение, 416
уклонение от, 417
схема, 249
метаклассификатора, 247
сходимость алгоритма, 210
сходство, 115
facade для процедуры оценки, 136
вычисление, 120
гибридные модели, 150
контента
анализ конкретного случая, 143
нормализация, 156
косинусная мера, 146
коэффициент линейной корреляции, 165
метрика Жаккарда, 125, 132
недостатки, 123
нормализация, 158
показатели, 174
сходство пользователь–предмет
на основе контента, 149
предметов, 139
большой объем данных, 142
симметричное свойство, 132

Т

таблица соответствий, проблемы подхода, 43
Тарьян, Роберт Е., 231
тау Кендалла, 168, 177
текст
понимание произвольного текста, 28
разметка, 63
на нескольких языках, 389
текстуальный анализ, 254
тема, 405
теорема Байеса, 86, 246, 258, 316
апостериорная вероятность, 258
априорная вероятность, 258
исходная предпосылка о независимости атрибутов, 260
наивное допущение, 89
подтверждение, 258
правдоподобие, 258
условная вероятность, 258
теория цепей Маркова, 69
термы, 99, 391
тест
на разность пропорций, 325, 346, 350
тестовые данные, 247
Макнемара, 346
метод bagging против метода boosting, 368
Тьюринга, 442
технические статьи, 398
тип, 238
тонкая настройка кластеризации, 428
точечные отклонения, 166

точность, 106, 307
транзакции
  допустимые, 287
  идентификация мошенничества, 287
тренировка классификатора
  переходы пользователя по ссылкам, 86
треугольник интеллекта, 29
Тьюринга тест, 442

У

уведомления о новостях, 384
узлы
  висячие, 70
  скрытые, 283
улучшение, 365
упорядочение и кластеризация, 183
уровень значимости
  статистический тест, 347, 350, 352
условные вероятности, 259, 260
  действия пользователя, 86
устойчивая кластеризация, использующая связи, *см.* ROCK, 211

Ф

факты, 266
Филдинг, Рой, 41
фильтрующая сеть, 378
финансовая поддержка, 326
финансовые активы, 331
финансы, нестабильность, 326
Фишера–Снедекора распределение, 353
Фишер, Рональд А., 352
фолксономия, 28
фонетика, 442
фонология, 442
Форги, Е. В., 210
Форджи, Чарльз, 268
форматы каналов синдикации
  Atom, 39
  RSS, 39
формула теоремы Байеса
  выходные данные, 260
фреймворк, 41
функциональный анализ, 440
функция гиперболического тангенса, 100, 124
функция стоимости, 310

Х

хи-квадрат, 348, 376
хранение рейтинговых оценок
  преимущества, 134
хранилище знаний, 33
Хэтчер, Эрик, 52

Ц

цель поиска, 67
центроид, 207, 287
  кластера, аналог центра тяжести, 207
  исходный выбор, 211
  роль, 208
цепи Маркова, 69
цепочечный эффект, 206
цепочка рассуждений, 243

Ч

частота появления, 100
Чебышева полином, 104
черный ящик, ловушка, 289
числовое представление, 191
чистка, 388
  новостных сообщений, 390
чувствительность, 291

Ш

Шацкер, система классификации, 237
шестиугольник, 207
широкомасштабный краулинг
  эффективность, 390
шифр, 237
шум, 289
  уровни, 333, 335, 337
  элементы, 219

Э

экстраполяция Айткена, 104
элементарный поиск
  загрузка, индексирование, поиск, 54
элементы естественных языков
  высокоуровневые, 393
Эпиктет, 323
эпсилон-окрестность, 221
  выбор значения, 225
эффект треугольника, 153

Я

язык
  морфология, 442
  синтаксис, 443
  специальный, синдикации, 40
язык запросов SQL
  кластеризация, 183
  предложения, 183
ярлыки основные, 30

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-186-8, название «Алгоритмы интеллектуального Интернета» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, атакже сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.