Пратик Джоши

Искусственный интеллект с примерами на Python

Создание приложений искусственного интеллекта с помощью Python для взаимодействия с окружающим миром

Packt›

Искусственный интеллект с примерами на Python

Artificial Intelligence with Python

Build real-world Artificial Intelligence applications with Python to intelligently interact with the world around you

Prateek Joshi

Packt>

BIRMINGHAM - MUMBAI

Искусственный интеллект с примерами на Python

Создание приложений искусственного интеллекта с помощью Python для взаимодействия с окружающим миром

Пратик Джоши

Москва • Санкт-Петербург
2019

ББК 32.973.26-018.2.75
Д42
УДК 681.3.07

Компьютерное издательство "Диалектика"
Перевод с английского канд. хим. наук *А.Г. Гузикевича*
Под редакцией *В.Р. Гинзбурга*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Джоши, Пратик.

Д42 Искусственный интеллект с примерами на Python. : Пер. с англ. — СПб. : ООО "Диалектика", 2019. — 448 с. — Парал. тит. англ.

ISBN 978-5-907114-41-8 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Пратик Джоши

Искусственный интеллект с примерами на Python

Подписано в печать 04.12.2019.
Формат 70x100/16. Гарнитура Palatino Linotype.
Усл. печ. л. 36,12. Уч.-изд. л. 20,26.
Тираж 500 экз. Заказ № 16502.

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907114-41-8 (рус.)
ISBN 978-1-78646-439-2 (англ.)

Оглавление

Введение 15

Глава 1. Введение в искусственный интеллект 21

Глава 2. Классификация и регрессия посредством обучения с учителем 45

Глава 3. Предсказательная аналитика на основе ансамблевого обучения 79

Глава 4. Распознавание образов с помощью обучения без учителя 111

Глава 5. Создание рекомендательных систем 137

Глава 6. Логическое программирование 163

Глава 7. Методы эвристического поиска 185

Глава 8. Генетические алгоритмы 211

Глава 9. Создание игр с помощью искусственного интеллекта 245

Глава 10. Обработка естественного языка 267

Глава 11. Вероятностный подход к обработке последовательных данных 295

Глава 12. Создание систем распознавания речи 323

Глава 13. Обнаружение и отслеживание объектов 345

Глава 14. Искусственные нейронные сети 379

Глава 15. Обучение с подкреплением 407

Глава 16. Глубокое обучение и сверточные нейронные сети 421

Предметный указатель 441

Содержание

Об авторе 13
О рецензенте 13

Введение 15

О чем эта книга 15
Для кого предназначена книга 17
Программная часть 17
Принятые соглашения 18
Файлы примеров и файл цветной вклейки 19
Ждем ваших отзывов! 19

Глава 1. Введение в искусственный интеллект 21

Что такое искусственный интеллект 22
Зачем нужно изучать ИИ? 22
Области применения ИИ 25
Направления исследований ИИ 27
Оценка уровня искусственного интеллекта с помощью теста Тьюринга 30
Как научить машины мыслить подобно людям 32
Создание рациональных агентов 34
Универсальный решатель задач 35
Решение задач с помощью GPS 36
Создание интеллектуальных агентов 37
Типы моделей 38
Установка Python 3 39
Установка в Ubuntu 39
Установка в Mac OS X 39
Установка в Windows 40
Установка пакетов 40
Загрузка данных 41
Резюме 43

Глава 2. Классификация и регрессия посредством обучения с учителем 45

Обучение с учителем и без учителя 45
Что такое классификация 46
Предварительная обработка данных 47
Бинаризация 48
Исключение среднего 48

Масштабирование 49
Нормализация 50
Кодирование меток 51
Логистический классификатор 52
Наивный байесовский классификатор 57
Матрица неточностей 61
Машины опорных векторов 64
Классификация данных о доходах с помощью машин опорных векторов 65
Что такое регрессия 69
Создание регрессора одной переменной 70
Создание многомерного регрессора 73
Оценка стоимости недвижимости с использованием регрессора на основе машины опорных векторов 75
Резюме 77

Глава 3. Предсказательная аналитика на основе ансамблевого обучения 79

Что такое ансамблевое обучение 79
Построение моделей обучения посредством ансамблевого метода 80
Что такое деревья принятия решений 81
Создание классификатора на основе дерева принятия решений 81
Случайные и предельно случайные леса 85
Создание классификаторов на основе случайных и предельно случайных лесов 86
Оценка мер достоверности прогнозов 91
Обработка дисбаланса классов 95
Нахождение оптимальных обучающих параметров с помощью сеточного поиска 100
Вычисление относительной важности признаков 103
Прогнозирование интенсивности дорожного движения с помощью классификатора на основе предельно случайных лесов 106
Резюме 109

Глава 4. Распознавание образов с помощью обучения без учителя 111

Что такое обучение без учителя 111
Кластеризация данных с помощью метода k-средних 112
Оценка количества кластеров с использованием метода сдвига среднего 116
Оценка качества кластеризации с помощью силуэтных оценок 121
Что такое смешанные гауссовские модели 124
Создание классификатора на основе гауссовской смешанной модели 126

Нахождение подгрупп на фондовом рынке с использованием модели распространения сходства 130
Сегментирование рынка на основе моделей совершения покупок 133
Резюме 136

Глава 5. Создание рекомендательных систем 137

Создание обучающего конвейера 137
Извлечение ближайших соседей 140
Создание классификатора методом K ближайших соседей 143
Вычисление оценок сходства 150
Поиск пользователей с похожими предпочтениями методом коллаборативной фильтрации 155
Создание рекомендательной системы фильмов 158
Резюме 162

Глава 6. Логическое программирование 163

Что такое логическое программирование 163
Конструкции логического программирования 165
Решение задач с помощью логического программирования 166
Установка пакетов Python 167
Сопоставление математических выражений 167
Проверка простых чисел 169
Парсинг генеалогического дерева 170
Анализ географических данных 176
Создание решателя головоломок 180
Резюме 183

Глава 7. Методы эвристического поиска 185

Что такое эвристический поиск 185
Неинформированный и информированный виды поиска 186
Задачи с ограничениями 187
Методы локального поиска 187
Алгоритм имитации отжига 188
Конструирование строк с использованием жадного поиска 189
Решение задачи с ограничениями 193
Решение задачи о раскраске областей 197
Создание головоломки "8" 200
Создание решателя для прохождения лабиринта 205
Резюме 210

Глава 8. Генетические алгоритмы 211

Эволюционные и генетические алгоритмы 211
Фундаментальные понятия генетических алгоритмов 212
Генерация битовых образов с предопределенными параметрами 214
Визуализация хода эволюции 221
Решение задачи символической регрессии 230
Создание контроллера интеллектуального робота 235
Резюме 242

Глава 9. Создание игр с помощью искусственного интеллекта 245

Использование поисковых алгоритмов в играх 245
Комбинаторный поиск 246
Алгоритм MiniMax 247
Альфа-бета-отсечение 247
Алгоритм NegaMax 248
Установка библиотеки **easyAI** 249
Создание робота для игры Last Coin Standing ("Последняя монета") 249
Создание робота для игры Tic-Tac-Toe ("крестики-нолики") 253
Создание двух роботов, играющих между собой в игру Connect Four ("Четыре в ряд") 257
Создание двух роботов, играющих между собой в игру Hexapawn ("Шесть пешек") 261
Резюме 265

Глава 10. Обработка естественного языка 267

Введение и установка пакетов 267
Токенизация текстовых данных 269
Преобразование слов в их базовые формы с помощью стемминга 270
Преобразование слов в их корневые формы с помощью лемматизации 272
Разбиение текстовых данных на информационные блоки 274
Извлечение частотности слов с помощью модели Bag of Words 276
Создание прогнозатора категорий 280
Создание анализатора грамматических родов 283
Создание сентимент-анализатора 286
Тематическое моделирование с использованием латентного размещения Дирихле 290
Резюме 294

Глава 11. Вероятностный подход к обработке последовательных данных 295

Что такое последовательные данные 295
Обработка временных рядов с помощью библиотеки Pandas 297
Извлечение срезов временных рядов данных 300
Выполнение операций над временными рядами 302
Извлечение статистик из временных рядов данных 305
Генерация данных с использованием скрытых марковских моделей 309
Идентификация буквенных последовательностей с помощью условных случайных полей 313
Анализ биржевого рынка 317
Резюме 321

Глава 12. Создание систем распознавания речи 323

Работа со звуковыми сигналами 323
Визуализация аудиосигналов 324
Преобразование аудиосигналов в частотные интервалы 326
Генерирование аудиосигналов 329
Синтезирование звуков для генерации музыки 331
Извлечение речевых признаков 333
Распознавание произносимых слов 337
Резюме 343

Глава 13. Обнаружение и отслеживание объектов 345

Установка библиотеки OpenCV 346
Вычисление разности между кадрами 346
Отслеживание объектов с помощью цветовых пространств 349
Отслеживание объектов путем вычитания фоновых изображений 353
Создание интерактивного трекера объектов с помощью алгоритма CAMShift 357
Отслеживание объектов с использованием оптических потоков 364
Обнаружение и отслеживание лиц 370
Использование каскадов Хаара для обнаружения лиц 371
Использование интегральных изображений для извлечения признаков 372
Отслеживание глаз и определение координат взора 375
Резюме 378

Глава 14. Искусственные нейронные сети 379

Введение в искусственные нейронные сети 379
Создание нейронной сети 380
Тренировка нейронной сети 380
Создание классификатора на основе перцептрона 381
Построение однослойной нейронной сети 384
Построение многослойной нейронной сети 388
Создание векторного квантизатора 392
Анализ последовательных данных с помощью рекуррентных нейронных сетей 395
Визуализация символов с использованием базы данных оптического распознавания символов 399
Создание системы оптического распознавания символов 401
Резюме 404

Глава 15. Обучение с подкреплением 407

Основы обучения с подкреплением 407
Обучение с подкреплением и обучение с учителем 408
Реальные примеры обучения с подкреплением 409
Строительные блоки обучения с подкреплением 410
Создание окружения 411
Создание агента обучения 416
Резюме 420

Глава 16. Глубокое обучение и сверточные нейронные сети 421

Что такое сверточные нейронные сети 421
Архитектура CNN 422
Типы слоев CNN 423
Создание линейного регрессора на основе перцептрона 424
Создание классификатора изображений на основе однослойной нейронной сети 431
Создание классификатора изображений на основе сверточной нейронной сети 433
Резюме 439

Предметный указатель 441

Об авторе

Пратик Джоши — специалист по проблемам искусственного интеллекта, автор пяти книг и постоянный докладчик на конференциях TEDx. Учредитель компании Pluto AI — венчурного стартапа из Силиконовой долины, занимающегося созданием аналитической платформы для интеллектуальной системы управления водоснабжением на основе методов глубокого обучения. Его технический блог (`https://prateekvjoshi.com/`), насчитывающий более 7500 подписчиков, посетили свыше 1,8 млн человек из более чем 200 стран. Часто публикует статьи, посвященные искусственному интеллекту, программированию на языке Python и прикладной математике. Окончил Университет Южной Калифорнии, получив диплом специалиста в области искусственного интеллекта. Работал в таких компаниях, как Nvidia и Microsoft.

О рецензенте

Ричард Марсден имеет более чем 20-летний опыт профессиональной разработки программного обеспечения. Последние десять лет управляет компанией Winwaed Software Technology LLC, независимым поставщиком программного обеспечения, который специализируется на создании инструментов и приложений для обработки картографических данных. Поддерживает сайт `www.mapping-tools.com`, на котором предлагаются инструментальные средства для таких картографических приложений, как Caliper Maptitude и Microsoft MapPoint.

Введение

Искусственный интеллект становится неотъемлемым атрибутом современного мира, управляемого технологиями и данными. Он интенсивно применяется в таких областях, как поисковые системы, распознавание образов, робототехника, беспилотные автомобили и т.п. В этой книге исследуются различные сценарии, взятые из реальной жизни. Прочитав ее, вы будете знать, какие алгоритмы искусственного интеллекта следует применять в том или ином контексте, и научитесь писать функциональный код.

Мы начнем с рассмотрения общих концепций искусственного интеллекта, после чего перейдем к обсуждению более сложных тем, таких как предельно случайные леса, скрытые марковские модели, генетические алгоритмы, сверточные нейронные сети и др. Эта книга предназначена для программистов, которые пишут код на языке Python и хотели бы применять алгоритмы искусственного интеллекта для создания прикладных программ. Книга написана так, чтобы излагаемый материал был доступен даже для тех, кто только начинает работать с Python, но хорошее знание языка Python, несомненно, будет не лишним при изучении примеров. Книга будет полезной и для опытных программистов на языке Python, стремящихся освоить методики искусственного интеллекта.

Вы узнаете о том, как принимать обоснованные решения при выборе необходимых алгоритмов, а также о том, как реализовывать эти алгоритмы для достижения наилучших результатов. Если вы хотите создавать многоцелевые приложения для обработки информации, содержащейся в изображениях, тексте, голосовых и других данных, то эта книга станет для вас надежным подспорьем.

О чем эта книга

Глава 1, *"Введение в искусственный интеллект"*, познакомит вас с рядом вводных понятий, относящихся к теме искусственного интеллекта. В ней будет рассказано о применениях искусственного интеллекта, его разновидностях и способах моделирования. Кроме того, вы пройдете через все этапы процедуры установки необходимых пакетов Python.

Глава 2, *"Классификация и регрессия посредством обучения с учителем"*, охватывает несколько методик обучения с учителем, предназначенных для решения задач классификации и регрессии. Вы узнаете о том, как анализировать данные о доходе и прогнозировать цены на недвижимость.

Глава 3, *"Предсказательная аналитика на основе ансамблевого обучения"*, содержит описание методик аналитического прогнозирования, основанных на ансамблевом моделирования, в частности на случайных лесах. Применение этих методик будет рассмотрено на примере прогнозирования интенсивности дорожного движения вблизи стадионов.

Глава 4, *"Распознавание образов с помощью обучения без учителя"*, содержит описание алгоритмов обучения без учителя, включая кластеризацию с применением метода k-средних и алгоритма сдвига среднего. Использование этих алгоритмов будет рассмотрено на примере анализа данных фондового рынка и сегментации клиентов.

Глава 5, *"Создание рекомендательных систем"*, посвящена алгоритмам, используемым для создания рекомендательных систем. Вы узнаете о том, как применять указанные алгоритмы для коллаборативной фильтрации и прогнозирования популярности кинофильмов.

Глава 6, *"Логическое программирование"*, посвящена принципам логического программирования и содержит ряд примеров его практического применения, включая сопоставление выражений, анализ генеалогических деревьев и решение головоломок.

Глава 7, *"Методы эвристического поиска"*, содержит описание методов эвристического поиска, используемых для выполнения поиска в пространстве решений. Будут показаны такие примеры их применения, как алгоритм имитации отжига, раскраска областей и прохождение лабиринтов.

Глава 8, *"Генетические алгоритмы"*, охватывает эволюционные алгоритмы и генетическое программирование. Вы познакомитесь с такими понятиями, как кроссовер, мутация и функции приспособленности, которые будут применены для решения задач символической регрессии и создания интеллектуальных систем управления роботами.

Глава 9, *"Создание игр с помощью искусственного интеллекта"*, посвящена применению искусственного интеллекта в играх. Вы узнаете о том, как создаются такие игры, как Tic-Tac-Toe ("Крестики-нолики"), Connect Four ("Четыре в ряд") и Hexapawn ("Шесть пешек").

Глава 10, *"Обработка естественного языка"*, посвящена таким методам анализа текстовых данных, как токенизация, стемминг, создание наборов слов (модель Bag of Words) и др. Вы узнаете о том, как применять эти методики для анализа тональности текста (сентимент-анализ) и тематического моделирования.

Глава 11, *"Вероятностный подход к обработке последовательных данных"*, содержит описание методик, используемых для анализа временных рядов и последовательных данных, включая скрытые марковские модели и условные случайные поля. Вы научитесь применять эти методики для анализа текстовых последовательностей и прогнозирования биржевых котировок.

Глава 12, *"Создание систем распознавания речи"*, демонстрирует применение алгоритмов для анализа речевых данных. Вы узнаете о том, как создавать системы визуализации аудиосигналов и распознавания голосовых команд.

Глава 13, *"Обнаружение и отслеживание объектов"*, посвящена алгоритмам, предназначенным для обнаружения и отслеживания объектов в живом видео. Вы познакомитесь с рядом методик, таких как оптические потоки, обнаружение лиц в кадре и отслеживание глаз.

Глава 14, *"Искусственные нейронные сети"*, охватывает алгоритмы создания нейронных сетей. Вы узнаете о том, как применять нейронные сети для создания систем оптического распознавания текста.

Глава 15, *"Обучение с подкреплением"*, содержит описание методик создания систем обучения с подкреплением. В ней рассказывается о том, как разрабатывать интеллектуальные агенты, способные к обучению на основе взаимодействия с окружением.

Глава 16, *"Глубокое обучение и сверточные нейронные сети"*, охватывает алгоритмы, предназначенные для создания систем глубокого обучения с использованием сверточных нейронных сетей. Вы узнаете о том, как создавать нейронные сети с помощью библиотеки TensorFlow. В частности, мы построим классификатор изображений на основе сверточных нейронных сетей.

Для кого предназначена книга

Эта книга адресована разработчикам на языке Python, которые интересуются созданием приложений с использованием искусственного интеллекта. Книга написана так, чтобы излагаемый материал был доступен даже тем, кто только осваивает Python. Хорошее знание этого языка будет лишь дополнительным положительным фактором, способствующим более эффективной работе с файлами примеров. Но книга будет полезна и опытным программистам, которые хотят применять методы искусственного интеллекта в рамках уже изученных ими платформ.

Программная часть

Темой книги является не сам язык Python, а разработка с его помощью различных приложений искусственного интеллекта. Для этого мы будем задействовать платформу Python 3. Наша цель — рассмотреть наиболее оптимальные способы использования библиотек Python для создания реальных приложений. Следуя этой цели, автор попытался придать всему коду как можно более удобочитаемый вид, чтобы читателям было проще понимать примеры и применять их в различных сценариях.

Принятые соглашения

В книге применяется ряд соглашений относительно оформления текста, облегчающих восприятие материала. Примеры использования соответствующих стилей и объяснение их назначения приведены ниже.

Встречающиеся в тексте элементы программного кода выделяются так: "Для включения других модулей можно использовать директиву `include`".

Блоки кода выделяются моноширинным шрифтом.

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Чтобы сфокусировать внимание на каком-то фрагменте кода, он выделяется полужирным шрифтом.

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Информация, вводимая или выводимая в командной строке, также выделяется полужирным моноширинным шрифтом.

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
    /etc/asterisk/cdr_mysql.conf
```

Важные понятия выделяются полужирным шрифтом. *Новые термины* выделяются курсивом.

Этой пиктограммой обозначены предостережения и важные замечания.

Этой пиктограммой обозначены полезные советы и подсказки.

Файлы примеров и файл цветной вклейки

Исходные коды примеров книги доступны на сайте GitHub:

`https://github.com/PacktPublishing/Artificial-Intelligence-with-Python`

Файлы примеров можно также скачать с веб-страницы книги на сайте издательства "Диалектика":

`http://www.williamspublishing.com/Books/978-5-907114-41-8.html`

По этому же адресу можно скачать файл цветной вклейки, содержащий цветные версии ряда иллюстраций, приведенных в книге в черно-белом виде.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info@williamspublishing.com`

WWW: `www.williamspublishing.com`

1

Введение в искусственный интеллект

В этой главе мы обсудим понятие искусственного интеллекта (ИИ) и способы его применения для решения реальных задач. Значительную часть своей повседневной жизни мы проводим, взаимодействуя с интеллектуальными системами. Такое взаимодействие происходит во время поиска информации в Интернете, биометрического распознавания лиц, отдавания голосовых команд. Все эти виды взаимодействий основаны на использовании систем искусственного интеллекта, которые становятся важным фактором современного стиля жизни. Подобные системы представляют собой сложные приложения, в которых для решения конкретных задач с помощью искусственного интеллекта привлекаются математические методы и программные алгоритмы. В этой книге вы ознакомитесь с фундаментальными принципами, лежащими в основе создания приложений подобного рода, и изучите примеры их практической реализации. Конечная цель — научить вас не бояться браться за новые и трудные задачи, поддающиеся решению с помощью искусственного интеллекта, с которыми вы можете столкнуться в процессе своей практической деятельности.

В этой главе вы ознакомитесь со следующими темами:

- что такое ИИ и почему следует его изучать;
- применения ИИ;
- разновидности ИИ;
- тест Тьюринга;
- рациональные агенты;

- универсальные решатели задач;
- создание интеллектуальных агентов;
- установка Python 3 в различных операционных системах;
- установка необходимых пакетов Python.

Что такое искусственный интеллект

Искусственный интеллект (ИИ) позволяет наделять машины возможностями, имитирующими интеллектуальное поведение человека и его способность рассуждать. Машины управляются программным обеспечением, поэтому ИИ имеет много общего с интеллектуальными программами, контролирующими поведение машин. Наука об ИИ разрабатывает теории и методологии, позволяющие машинам оценивать окружающую обстановку и реагировать на различные ситуации так, как на них реагировал бы человек.

Как показывает тщательный анализ направлений развития науки об ИИ, в своих попытках дать определение ИИ ученые применяли различные подходы. В современном мире ИИ задействуется во многих областях, принимая самые разнообразные формы. Мы хотим, чтобы машины могли ощущать и рассуждать, думать и действовать. Кроме того, мы хотим, чтобы поведение машин было рациональным.

Работы в области ИИ тесно связаны с изучением свойств человеческого мозга. Исследователи полагают, что понимание принципов работы мозга сделает создание ИИ вполне осуществимой задачей. Имитируя процессы, происходящие в человеческом мозге в процессе обучения, мышления и принятия решений, мы можем создать машину, способную делать то же самое. Такая машина послужит платформой для создания систем, способных к обучению.

Зачем нужно изучать ИИ?

Успехи в создании ИИ способны повлиять на все аспекты нашей жизни. Исследования в этой области направлены на изучение свойств и закономерностей поведения объектов. С помощью ИИ мы хотим создавать умные системы и стремимся понять, как заставить машины выполнять творческие функции. Реализуя интеллектуальные системы, мы приближаемся к пониманию того, каким образом одни интеллектуальные системы, подобные нашему мозгу, способны справляться с созданием других.

Рис. 1.1 дает некоторое представление о процессе обработки информации нашим мозгом.

Рис. 1.1

По сравнению с другими областями науки, такими как математика или физика, которые существуют столетиями, наука об ИИ сравнительно молодая. За последние два десятилетия она продемонстрировала замечательные достижения, примерами которых могут служить беспилотные автомобили и интеллектуальные шагающие роботы. Уже достигнутые результаты делают довольно очевидным тот факт, что исследования в области ИИ способны коренным образом изменить нашу жизнь в ближайшие годы.

Можно лишь удивляться тому, как человеческий мозг справляется с обработкой огромных объемов информации с минимальными усилиями. Мы распознаем объекты, понимаем другие языки, учимся новому и выполняем множество разнообразных сложных задач. Каким образом нашему мозгу удается это делать? Пытаясь делать то же самое с помощью машин, мы видим, что они остаются далеко позади! Например, рассуждая о таких вещах, как внеземная жизнь или путешествия во времени, мы даже не уверены в том, могут ли они существовать на самом деле. Хорошая новость относительно Святого Грааля ИИ заключается в том, что нам достоверно известно о его существовании. Этим Святым Граалем является наш мозг! Он служит ярчайшим примером интеллектуальной системы. Нам нужно лишь сымитировать его функциональность для создания систем, способных делать нечто похожее, а возможно, даже большее.

Отдельные этапы преобразования исходных данных в знания можно условно представить в виде следующей схемы (рис. 1.2).

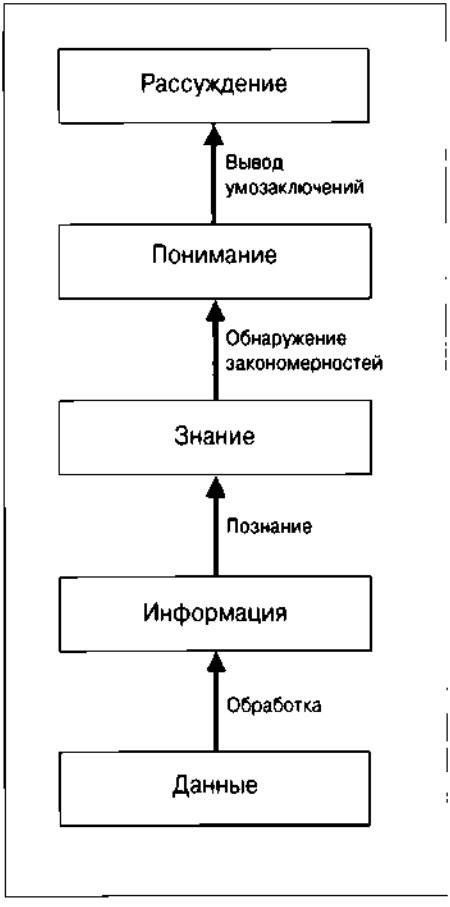

Рис. 1.2

Одна из основных причин нашего стремления к изучению ИИ — возможность автоматизации многих процессов. Мы живем в мире, в котором:

- приходится иметь дело с огромными объемами данных, с обработкой которых человеческий мозг просто не в состоянии справиться;
- данные поступают одновременно из множества источников;
- эти данные не структурированы и поступают хаотично;
- знания, полученные на основе этих данных, должны непрерывно обновляться, поскольку сами данные подвержены постоянным изменениям;
- восприятие данных и ответная реакция на них должны с высокой точностью осуществляться в режиме реального времени.

Несмотря на то что человеческий мозг проявляет замечательные способности в отношении анализа окружающей обстановки, его возможностей недостаточно для удовлетворения всех перечисленных выше условий. Следовательно, мы вынуждены изобретать и разрабатывать интеллектуальные системы, позволяющие преодолеть это ограничение. Нам нужны системы искусственного интеллекта, которые могли бы:

- эффективно обрабатывать большие объемы данных, хранение которых в настоящее время стало возможным благодаря облачным вычислениям;
- получать данные одновременно из нескольких источников без каких-либо задержек;
- индексировать и организовывать данные способами, обеспечивающими возможность их осмысления;
- обучаться на новых данных и постоянно обновлять получаемые знания, используя подходящие алгоритмы обучения;
- принимать решения и реагировать на изменяющиеся обстоятельства в режиме реального времени.

Методики ИИ активно используются для совершенствования существующих машин, чтобы они становились все умнее и могли быстрее и эффективнее выполнять возложенные на них функции.

Области применения ИИ

Теперь, когда вы уже знаете, как обрабатывается информация, мы можем перейти к рассмотрению применения ИИ в реальной жизни. ИИ проявляет себя в разных формах, поэтому очень важно понимать, чем именно он может быть полезен для той или иной сферы деятельности. ИИ интенсивно используется во многих областях, и круг его применений чрезвычайно быстро расширяется. Рассмотрим наиболее популярные из них.

- **Компьютерное зрение.** Разработаны системы, предназначенные для обработки таких визуальных данных, как изображения и видео. Такие системы анализируют содержание изображений и извлекают полезную информацию на основании предоставленных типовых образцов. Например, Google использует технологию *реверсивного (обратного) поиска изображений* для нахождения визуально подобных изображений в Интернете (рис. 1.3).
- **Обработка естественного языка.** Системы этого типа предназначены для распознавания текстов, написанных на естественных языках. Мы можем взаимодействовать с машиной, передавая ей команды в виде

текстовых предложений. Поисковые системы интенсивно используют эту технологию для доставки релевантных результатов поиска пользователям.

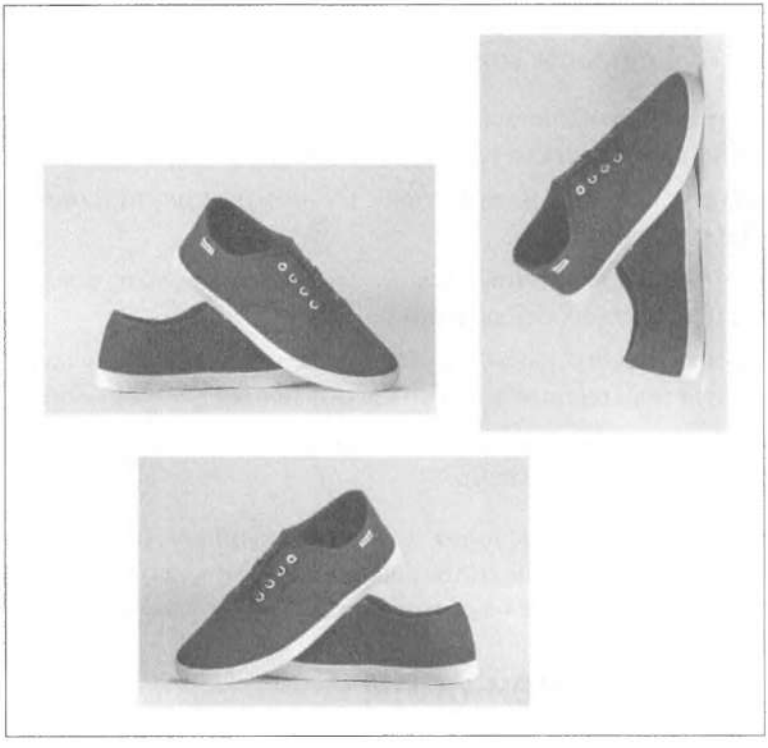

Рис. 1.3

- **Распознавание речи.** Эти системы способны воспринимать звуковую информацию и понимать произносимые слова. Например, наши смартфоны оборудованы интеллектуальными персональными помощниками, которые понимают голосовые команды и реагируют на них предоставлением соответствующей информации или выполнением запрошенных действий.
- **Экспертные системы.** В этих системах методики ИИ используются для принятия решений или предоставления соответствующих рекомендаций. Как правило, в таких областях, как финансы, медицина, маркетинг и др., для этой цели используют базы знаний. На рис. 1.4 иллюстрируется, что собой представляет экспертная система и как она взаимодействует с пользователем.
- **Игры.** ИИ широко применяется в индустрии игр. Он используется для проектирования интеллектуальных агентов, способных состязаться

в мастерстве игры с человеком. В качестве примера можно привести AlphaGo — компьютерную программу, которая умеет играть в стратегическую игру Go. ИИ также используется для проектирования игр другого типа, в которых от компьютера ожидается интеллектуальное поведение.

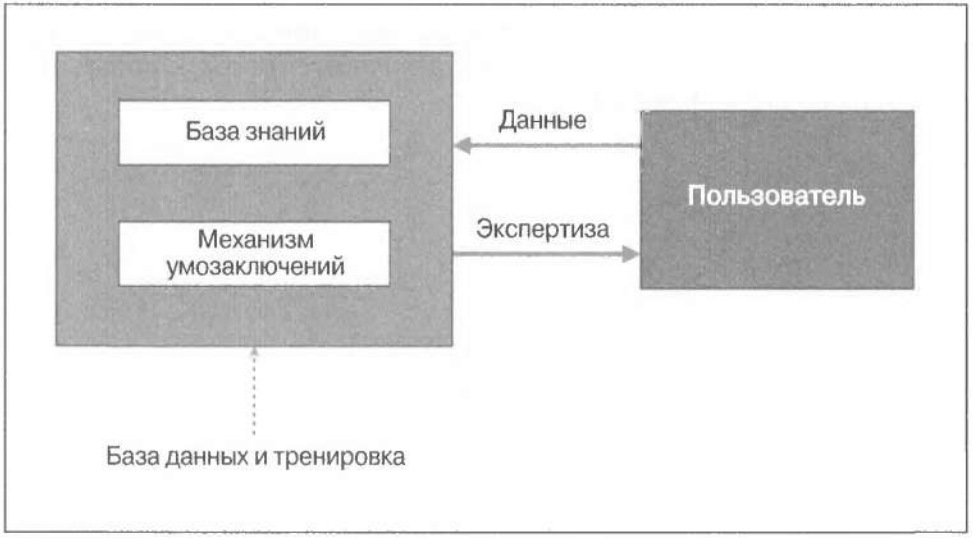

Рис. 1.4

- **Робототехника.** Робототехнические системы в действительности объединяют в себе многие концепции ИИ. Эти системы способны выполнять множество самых разнообразных задач. В зависимости от ситуации, роботы могут оборудоваться датчиками и приводными элементами, обеспечивающими выполнение всевозможных действий. Датчики могут распознавать предметы, находящиеся в поле их зрения, измерять их температуру, реагировать на выделяемое ими тепло или совершаемые ими движения и т.п. Вмонтированные в электронные платы процессоры выполняют необходимые расчеты в режиме реального времени. Кроме того, роботы могут адаптировать свое поведение к изменению внешних условий.

Направления исследований ИИ

Очень важно понимать суть различных направлений исследования ИИ, поскольку это позволит вам выбрать наиболее подходящую платформу для решения стоящей перед вами задачи. Ниже перечислены темы, которые доминируют в этой области.

- **Машинное обучение и распознавание образов.** Возможно, это наиболее популярное направление разработок в области ИИ. Мы проектируем и разрабатываем программы, способные учиться на предоставляемых им данных. На основании моделей обучения мы можем составлять прогнозы, касающиеся неизвестных данных. В этом отношении одним из основных ограничений является то обстоятельство, что эффективность подобных программ ограничивается мощностью данных. Принцип функционирования типичной системы машинного обучения иллюстрируется на рис. 1.5.

Рис. 1.5

Получая результаты наблюдения, система тренируется, сравнивая их с теми результатами, которые уже наблюдались в предоставленных ей примерах с известным ответом. Например, в случае системы распознавания лиц программа будет пытаться найти соответствие образцам глаз, носа, губ, бровей и т.п. с целью идентификации личности на основе соответствующих изображений, хранящихся в базе данных пользователей.

- **Логический ИИ.** Для выполнения программ в системах логического ИИ используются методы математической логики. Программа логического ИИ в основном представляет собой набор утверждений в

логической форме, которые выражают факты и правила, относящиеся к конкретной предметной области. Подобные подходы интенсивно используются для установления соответствия шаблонам, парсинга текста, семантического анализа, а также при решении ряда других задач.

- **Поиск.** В программах ИИ широко используются методы поиска. Такие программы исследуют большое количество всевозможных вариантов и выбирают наиболее оптимальный из них. Например, такой подход часто применяется во многих стратегических играх, в частности в шахматах, а также при управлении компьютерными сетями, распределении ресурсов, планировании и т.п.
- **Представление знаний.** Чтобы факты, относящиеся к окружающему нас миру, имели для системы смысл, они должны предоставляться ей в той или иной форме. Для этой цели часто используют языки математической логики. При удачно выбранной форме представления знаний система способна функционировать более интеллектуальным образом. Близкой к этому областью исследований является онтология, которая имеет дело с формализацией описания существующих видов объектов. Информационные онтологии дают формальные определения свойств объектов и отношений между ними, существующих в конкретной предметной области. Обычно это делается с привлечением определенной таксономии или некоторой иерархической структуры. Диаграмма, приведенная на рис. 1.6, поясняет различие между информацией и знаниями.

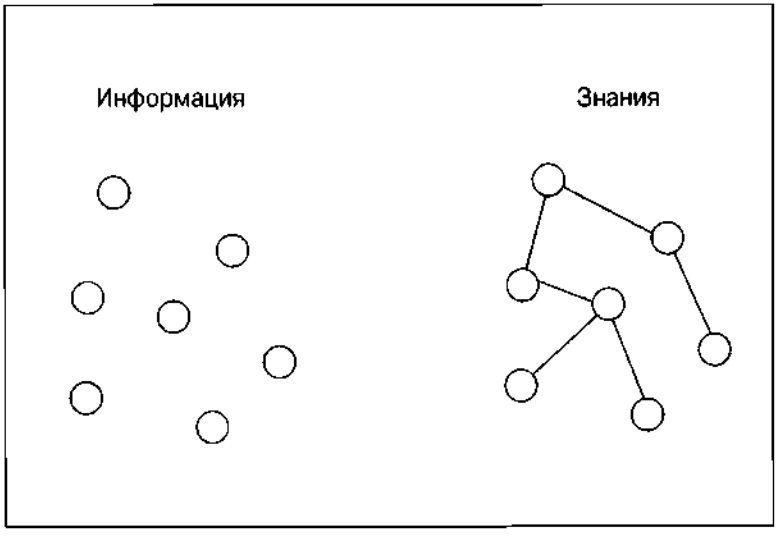

Рис. 1.6

- **Планирование.** Это направление связано с оптимальным планированием, обеспечивающим максимально возможную отдачу при минимальных затратах. В подобных программах исходными данными служат факты, относящиеся к конкретной ситуации, и формулировка цели. Таким программам также должны быть известны факты о внешнем мире, чтобы они ориентировались в том, какими правилами следует руководствоваться. На основании предоставленной информации программа генерирует наиболее оптимальный план, обеспечивающий достижение поставленной цели.
- **Эвристика.** Эвристика — это методология, использовать которую наиболее целесообразно в тех случаях, когда решение задачи должно быть найдено в кратчайшие сроки, но при этом не требуется, чтобы полученное решение было оптимальным. Здесь речь идет скорее о выдвижении правдоподобной гипотезы относительно подхода, который должен быть предпринят для решения задачи. В исследованиях ИИ часто встречаются ситуации, когда мы не можем проверить каждую из имеющихся возможностей, чтобы затем выбрать наилучший вариант. Именно в таких случаях и приходится прибегать к эвристическим методам для достижения своих целей. Эти методы широко используются в таких областях ИИ, как робототехника, поисковые системы и т.п.
- **Генетическое программирование.** Это автоматический подбор программы для решения определенной задачи путем объединения возможностей программ и выбора их комбинированного варианта, наиболее пригодного для данной задачи. Код таких программ пишется в виде набора генов с использованием алгоритма, который обеспечивает получение программы, способной найти действительно эффективное решение поставленной задачи.

Оценка уровня искусственного интеллекта с помощью теста Тьюринга

Легендарный математик и компьютерный ученый Алан Тьюринг предложил тест, названный в его честь *тестом Тьюринга*, который позволяет определить уровень "интеллектуальности" машины. С помощью этого теста можно проверить, способен ли компьютер имитировать поведение мыслящих существ. Тьюринг определил как *разумное* такое поведение компьютера, которое в процессе "беседы" с ним невозможно отличить от поведения человека. Этого было бы достаточно для того, чтобы убедить интервьюера в том, что ответы на поставленные им вопросы даются человеком.

Чтобы проверить, способны ли на подобное машины, Тьюринг предложил следующую организацию теста: адресованные машине вопросы должны задаваться человеком посредством текстового интерфейса. В качестве еще одного ограничения он указал, что человек, задающий вопросы, не должен знать, кто именно выступает в качестве его собеседника — человек или машина, т.е. предполагается, что собеседником может быть как машина, так и человек. Чтобы реализовать описанную схему теста, человек должен взаимодействовать с обоими объектами посредством текстового интерфейса. Эти два объекта называются *респондентами*. Одним из них должен быть человек, другим — машина.

Машина-респондент проходит тест, если интервьюеру не удается определить, от кого исходили ответы — от машины или от человека. Схема организации теста Тьюринга представлена на рис. 1.7.

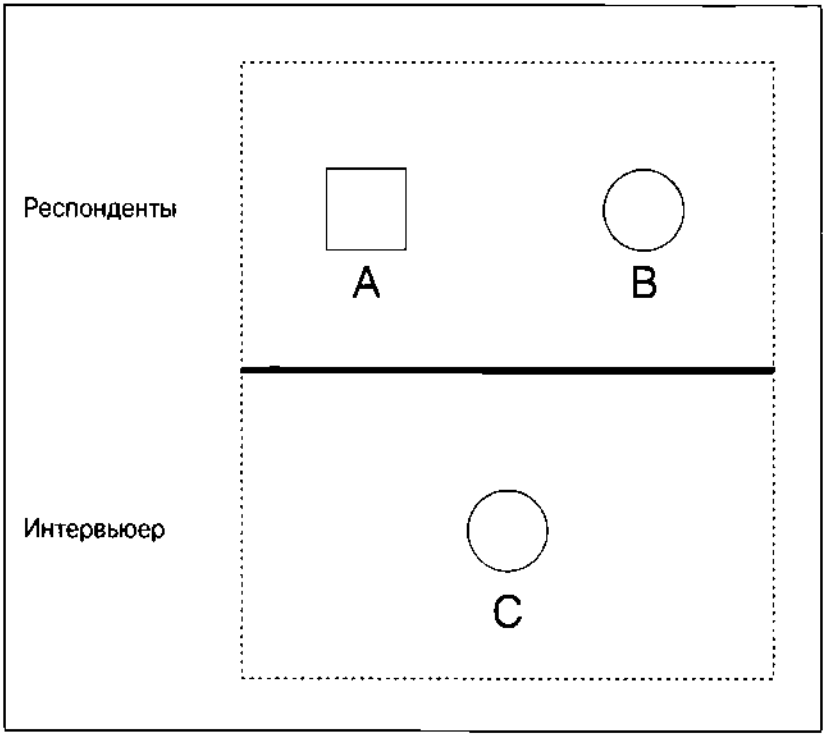

Рис. 1.7

Как нетрудно догадаться, для машины-респондента это задание является довольно трудным. В процессе беседы происходит очень многое. Как минимум машина должна хорошо справляться со следующими задачами.

- **Обработка естественного языка.** Это необходимо машинам для того, чтобы они могли общаться с интервьюером. Машина должна уметь выполнять синтаксический анализ предложений, определять контекст и адекватно отвечать на вопросы.
- **Представление знаний.** Машина нуждается в сохранении информации, предоставленной ей до проведения диалога. Она также должна отслеживать информацию, полученную в процессе диалога, чтобы использовать ее повторно, если в этом возникнет необходимость.
- **Вывод умозаключений.** Очень важно, чтобы машина понимала, каким образом следует интерпретировать сохраненную информацию. Обычно у людей это происходит автоматически, что позволяет им выводить умозаключения в режиме реального времени.
- **Машинное обучение.** Это необходимо для того, чтобы машина могла приспосабливаться к новым условиям в режиме реального времени. Машина должна анализировать и обнаруживать закономерности, чтобы выводить соответствующие умозаключения.

Вы, должно быть, задумались над тем, почему человек, участвующий в тесте, должен использовать текстовый интерфейс в процессе общения. Согласно Тьюрингу, физическая имитация человека является излишней для данного теста. В этом и заключается причина, по которой тест Тьюринга проводится без прямого физического контакта между человеком и машиной. Существует также полный тест Тьюринга, включающий эффекты зрения и движения. Чтобы пройти такой тест, машина должна видеть объекты, используя машинное зрение, и перемещаться, используя робототехнику.

Как научить машины мыслить подобно людям

На протяжении десятилетий мы пытаемся создать машину, которая могла бы мыслить, как человек. Для этого нам прежде всего нужно понять, как мыслят люди. Как мы должны действовать, чтобы понять природу человеческого мышления? Например, можно было бы вести записи, фиксирующие нашу реакцию на то, что происходит вокруг. Но этот способ очень быстро обнаружит свою бесперспективность из-за огромного количества необходимых записей. Другой подход основан на проведении экспериментов предопределенного формата. Мы разрабатываем серию вопросов, которые охватывают широкий спектр тем, имеющих непосредственное отношение к человеку, а затем анализируем, как люди на них отвечают.

Как только будет собран достаточно большой объем данных, мы сможем построить модель, имитирующую поведение человека. Далее эту модель можно будет использовать для создания программного обеспечения, способного думать, как люди. Конечно, легко сказать, да трудно сделать! Все, что нас интересует, — это выходная информация, предоставляемая программой в ответ на входную информацию. Если поведение программы согласуется с поведением человека, то мы можем сказать, что мышление человека описывается аналогичным механизмом.

Человеческое мышление можно условно разбить на несколько уровней, которые представлены на рис. 1.8, иллюстрирующем их иерархическую структуру.

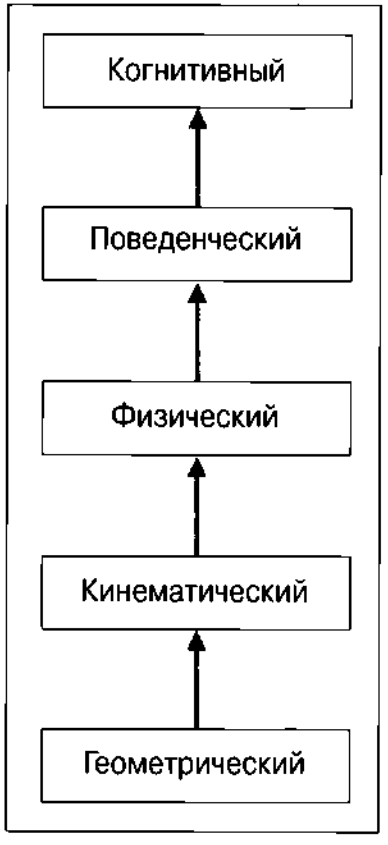

Рис. 1.8

Существует целый раздел компьютерной науки — *когнитивное моделирование*, которое занимается имитацией мыслительного процесса человека. Его целью является достижение понимания того, как люди решают задачи. Специалисты по когнитивному моделированию анализируют, какие

умственные процессы оказываются вовлеченными в поиск решения, и на основании этого строят программную модель. Впоследствии эта модель применяется для имитации поведения человека. Когнитивное моделирование задействуется во многих приложениях ИИ, таких как глубокое обучение, экспертные системы, обработка естественного языка, робототехника и др.

Создание рациональных агентов

Многие исследования в области ИИ нацелены на создание *рациональных агентов*. Что такое рациональный агент? Прежде чем обсудить это понятие, определим смысл термина "рациональность". *Рациональность* — это выполнение действий, которые соответствуют данным конкретным обстоятельствам. Действия должны выполняться таким образом, чтобы принести максимальную выгоду выполняющему их объекту. Говорят, что агент действует рационально, если он, придерживаясь установленного набора правил, выполняет действия, направленные на достижение поставленной цели. Он принимает решения и действует исключительно в соответствии с имеющейся информацией. Подобные системы широко применяются для проектирования роботов, предназначенных для выполнения различных заданий в незнакомой местности.

Но что такое *соответствующие действия*? Ответ зависит от того, какие задачи поставлены перед агентом. Предполагается, что агент действует интеллектуально, без вмешательства человека. Мы хотим наделить агента способностью приспосабливаться к новым ситуациям. Он должен адекватно оценивать окружающую обстановку и действовать так, чтобы добиться наилучших с его точки зрения результатов. Наилучшие результаты диктуются общей целью, которую он пытается достигнуть. На рис. 1.9 представлена условная схема того, как входная информация преобразуется в действия.

А как нам измерить эффективность действий рационального агента? Кто-то скажет, что такой мерой могла бы служить степень достижения успеха. Агент настраивается на решение определенной задачи, поэтому оценка эффективности зависит от того, какой процент задачи выполнен. Но мы должны задуматься над тем, что именно составляет сущность рационализма во всей полноте этого понятия. Если говорить только о результатах, то может ли агент предпринять какие-либо действия для получения данного результата?

Несомненно, что способность делать правильные умозаключения является необходимым компонентом рационального поведения, поскольку агент должен действовать рационально для достижения своих целей. Это свойство позволит ему приходить к правильным выводам, которыми можно последовательно руководствоваться. А что можно сказать о ситуациях, в которых

отсутствует возможность предпринимать доказуемо правильные действия? Бывают такие ситуации, в которых агент не знает, что ему делать, но в то же время он должен как-то действовать. В подобных случаях мы не может использовать концепцию логических выводов для определения рационального поведения.

Рис. 1.9

Универсальный решатель задач

Универсальный решатель задач (General Problem Solver — GPS) — программа ИИ, предложенная Гербертом Саймоном, Джоном Клиффордом Шоу и Алленом Ньюэллом. Это была первая компьютерная программа, появившаяся в мире ИИ. Она разрабатывалась для создания универсальной машины, способной решать задачи. Разумеется, к тому времени существовало множество компьютерных программ, но все они выполняли вполне определенные задачи. GPS была первой программой, предназначенной для решения задач произвольного типа. По замыслу ее авторов для решения любой задачи программа должна была использовать один и тот же алгоритм.

Совершенно очевидно, что реализовать эту амбициозную цель оказалось не так-то просто. Для этого авторам программы потребовалось создать новый язык — IPL (Information Processing Language — язык обработки информации). Работа программы базировалась на возможности формулирования любой задачи с помощью набора формул. Эти формулы становились частью направленного графа, имеющего множество источников и стоков. Применительно к графам термин *источник* относится к начальному узлу, а термин *сток* — к конечному. В случае GPS источниками служат аксиомы, а стоками — логические выводы.

Хотя GPS задумывалась в качестве универсального средства, она могла решать лишь такие хорошо определенные задачи, как доказательство геометрических и логических теорем. Она также могла решать головоломки и играть в шахматы. Это обусловлено тем, что подобные задачи могли быть в достаточной степени формализованы. Но в случае реальных задач формализация очень быстро становится трудноразрешимой проблемой из-за наличия большого количества возможных дальнейших шагов, которые приходится учитывать. При попытке решения задачи методом грубой силы, т.е. путем подсчета количества путей в графе, вычисление решения становится практически нереализуемым.

Решение задач с помощью GPS

Рассмотрим структурирование конкретной задачи для получения ее решения с помощью GPS.

1. Первый шаг состоит в определении целей. Пусть нашей целью будет покупка молока в магазине.
2. Следующим шагом является определение предусловий. Эти предусловия связаны с целями. Чтобы доставить молоко из магазина, оно должно быть в нем в наличии и мы должны располагать средством его транспортировки.
3. После этого мы должны определить операторы. Если средством транспортировки является автомобиль, который нуждается в дозаправке горючим, то мы должны быть уверены в том, что сможем заплатить нужную сумму на автозаправке. Мы также должны убедиться в том, что у нас останется достаточно денег для того, чтобы заплатить за молоко в магазине.

Оператор заботится об условиях и обо всем, что на них влияет. Он состоит из действий, предусловий и изменений, являющихся результатом предпринимаемых действий. В данном случае действием является уплата денег магазину.

Конечно же, возможность совершения этого действия зависит в первую очередь от наличия денег, что служит предусловием. Выплачивая магазину деньги, вы изменяете их состояние, в результате чего получаете молоко.

GPS будет работать только в том случае, если вам удастся структурировать задачу, как мы это только что сделали. Основным ограничением является необходимость проведения трудоемкой процедуры поиска, без которой GPS не может обойтись в процессе выполнения своей работы и которая отнимает много времени при решении любой реальной задачи.

Создание интеллектуальных агентов

Существуют разные способы создания интеллектуальных агентов. К числу наиболее распространенных из них относятся машинное обучение, базы знаний, наборы правил и др. В этом разделе мы ограничимся подходом, основанным на машинном обучении. Данный метод предполагает наделение агента интеллектуальными способностями посредством тренировки (обучения) на известных данных.

На рис. 1.10 представлена общая схема взаимодействия интеллектуального агента с окружением.

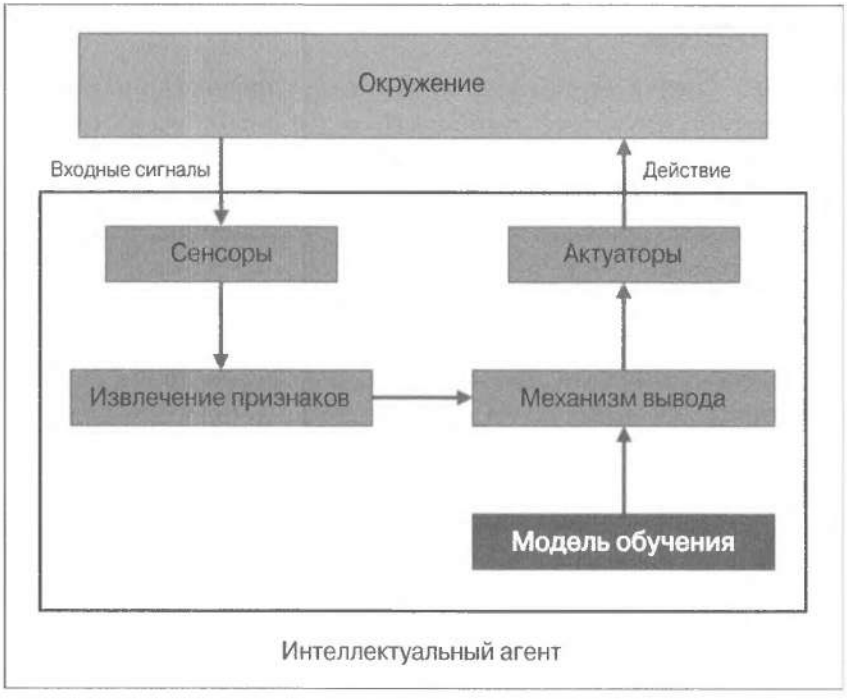

Рис. 1.10

Применяя машинное обучение, мы стремимся программировать машины так, чтобы они могли использовать маркированные данные для решения любой конкретной задачи. Прогоняя через машину данные и ассоциированные с ними маркеры, мы учим ее распознавать образы и отношения между ними.

В предыдущем примере работа интеллектуального агента зависит от модели обучения, используемой для работы механизма логического вывода. Получив входной сигнал, сенсор посылает его в блок извлечения признаков. После извлечения соответствующих признаков тренируемый механизм вывода строит соответствующий прогноз на основании модели обучения. Эта модель обучения создается с использованием машинного обучения. Затем механизм вывода принимает решение и передает его актуатору, который выполняет требуемое действие в реальном окружении.

В настоящее время машинное обучение находит множество применений. Оно используется для распознавания образов и речи, прогнозирования рыночных тенденций, в робототехнике и других областях. Чтобы понять сущность машинного обучения и научиться создавать полные решения, вам потребуется знакомство со множеством технологий, таких как распознавание образов, искусственные нейронные сети, добыча данных, статистика и т.п.

Типы моделей

В мире ИИ существует два типа моделей: аналитические и обучаемые. До появления машин, способных выполнять необходимые вычисления, обычно использовались *аналитические модели*. Они основывались на математических формулировках, представляющих собой, по сути, описание последовательных шагов, которые требовалось выполнить для получения окончательного уравнения. Проблемой этого подхода является то, что он зависит от суждений человека. Как следствие, подобные модели были упрощенными и страдали неточностью, обусловленной недостаточно большим количеством параметров.

Затем наступила эра компьютеров. Компьютеры хорошо справлялись с анализом данных. Поэтому со временем все шире стали использоваться *обучаемые модели*. Такие модели создаются посредством тренировки. Для получения уравнения в процессе тренировки машины просматривают множество примеров входных и соответствующих выходных данных. Подобным обучаемым моделям свойственны сложность и высокая точность, поскольку в них учитываются тысячи параметров. Это приводит к тому, что результирующее уравнение, управляющее данными, оказывается чрезвычайно сложным.

Методы машинного обучения позволяют получать такие обучаемые модели, которые могут быть использованы в механизме вывода. Наиболее благоприятным для нас следствием этого факта является то, что в данном случае

мы избавлены от необходимости выводить базовые математические формулы. От нас не требуется владение сложным математическим аппаратом, поскольку машина извлекает эти формулы на основании данных. Все, что мы должны сделать, — предоставить соответствующие списки входных и выходных значений. Обученная модель, которую мы при этом получаем, всего лишь отражает отношения между маркированными входными и желаемыми выходными значениями.

Установка Python 3

Все примеры, приведенные в книге, были получены с использованием Python 3. Убедитесь в том, что в вашей системе установлена последняя версия Python 3. Чтобы это проверить, введите в окне терминала следующую команду:

```
$ python3 --version
```

Если отобразится нечто вроде `Python 3.x.x` (где *x.x* — номера версии), то устанавливать Python не потребуется. В противном случае выполнить процедуру установки будет совсем несложно.

Установка в Ubuntu

В версиях Ubuntu 14.*xx* и выше Python 3 установлен по умолчанию. Если это не так, выполните установку с помощью следующей команды:

```
$ sudo apt-get install python3
```

После этого выполните вышеупомянутую проверку версии.

```
$ python3 --version
```

Номер установленной версии отобразится в окне вашего терминала.

Установка в Mac OS X

Если вы работаете в системе Mac OS X, то для установки Python 3 рекомендуется использовать пакет Homebrew. Это замечательный пакет-установщик, предназначенный для Mac OS X, которым действительно легко пользоваться. В случае отсутствия этого пакета вы сможете установить его с помощью следующей команды:

```
$ ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

После этого можно обновить менеджер пакетов.

```
$ brew update
```

Теперь можно установить Python 3.

```
$ brew install python3
```

Выполните вышеупомянутую проверку версии.

```
$ python3 --version
```

Номер установленной версии отобразится в окне вашего терминала.

Установка в Windows

Если вы работаете в Windows, то рекомендуется использовать дистрибутив Python 3, соответствующий спецификации SciPy-stack. В этом отношении весьма популярен и легок в использовании дистрибутив Anaconda. Соответствующие инструкции по его установке вы найдете по адресу `https://www.continuum.io/downloads`.

Если вы хотите ознакомиться с возможностями других дистрибутивов Python 3, совместимых со спецификацией SciPy-stack, посетите веб-страницу `http://www.scipy.org/install.html`. В этих дистрибутивах хорошо то, что они поставляются со всеми необходимыми предустановленными пакетами. При использовании любой из этих версий вам не придется устанавливать пакеты по отдельности.

Завершив установку, выполните вышеупомянутую команду проверки.

```
$ python3 --version
```

Номер установленной версии отобразится в окне вашего терминала.

Установка пакетов

На протяжении всей книги мы будем использовать различные пакеты, такие как NumPy, SciPy, *scikit-learn* и *matplotlib*. Убедитесь в том, что эти пакеты установлены в вашей системе, прежде чем продолжить чтение.

Если вы используете Ubuntu или Mac OS X, то установка указанных пакетов не составит труда. Каждый из них устанавливается с помощью команды, умещающейся в одной строке в окне терминала. Соответствующие ссылки, касающиеся установки, приведены ниже.

- **NumPy**
  `http://docs.scipy.org/doc/numpy-1.10.1/user/install.html`
- **SciPy**
  `http://www.scipy.org/install.html`

- **scikit-learn**
  `http://scikit-learn.org/stable/install.html`
- **matplotlib**
  `http://matplotlib.org/1.4.2/users/installing.html`

Если вы работаете в Windows, то вам достаточно установить версию Python 3, соответствующую спецификации SciPy-stack.

Загрузка данных

Для построения модели обучения нам нужны данные, представляющие внешний мир. Теперь, когда у нас установлены необходимые пакеты Python, рассмотрим, как использовать их для взаимодействия с данными. Перейдем к командной строке Python, введя в окне терминала следующую команду:

```
$ python3
```

Импортируем пакет, содержащий все наборы данных.

```
>>> from sklearn import datasets
```

Загрузим набор данных с ценами на недвижимость.

```
>>> house_prices = datasets.load_boston()
```

Выведем данные.

```
>>> print(house_prices.data)
```

В окне терминала должен отобразиться следующий вывод (рис. 1.11).

```
>>> print(house_prices.data)
[[  6.32000000e-03   1.80000000e+01   2.31000000e+00 ...,   1.53000000e+01
    3.96900000e+02   4.98000000e+00]
 [  2.73100000e-02   0.00000000e+00   7.07000000e+00 ...,   1.78000000e+01
    3.96900000e+02   9.14000000e+00]
 [  2.72900000e-02   0.00000000e+00   7.07000000e+00 ...,   1.78000000e+01
    3.92830000e+02   4.03000000e+00]
 ...,
 [  6.07600000e-02   0.00000000e+00   1.19300000e+01 ...,   2.10000000e+01
    3.96900000e+02   5.64000000e+00]
 [  1.09590000e-01   0.00000000e+00   1.19300000e+01 ...,   2.10000000e+01
    3.93450000e+02   6.48000000e+00]
 [  4.74100000e-02   0.00000000e+00   1.19300000e+01 ...,   2.10000000e+01
    3.96900000e+02   7.88000000e+00]]
```

Рис. 1.11

Проверим маркерные значения.

```
>>> print(house_prices.target)
```

В окне терминала должен отобразиться следующий вывод (рис. 1.12).

```
>>> print(house_prices.target)
[ 24.   21.6  34.7  33.4  36.2  28.7  22.9  27.1  16.5  18.9  15.   18.9
  21.7  20.4  18.2  19.9  23.1  17.5  20.2  18.2  13.6  19.6  15.2  14.5
  15.6  13.9  16.6  14.8  18.4  21.   12.7  14.5  13.2  13.1  13.5  18.9
  20.   21.   24.7  30.8  34.9  26.6  25.3  24.7  21.2  19.3  20.   16.6
  14.4  19.4  19.7  20.5  25.   23.4  18.9  35.4  24.7  31.6  23.3  19.6
  18.7  16.   22.2  25.   33.   23.5  19.4  22.   17.4  20.9  24.2  21.7
  22.8  23.4  24.1  21.4  20.   20.8  21.2  20.3  28.   23.9  24.8  22.9
  23.9  26.6  22.5  22.2  23.6  28.7  22.6  22.   22.9  25.   20.6  28.4
  21.4  38.7  43.8  33.2  27.5  26.5  18.6  19.3  20.1  19.5  19.5  20.4
  19.8  19.4  21.7  22.8  18.8  18.7  18.5  18.3  21.2  19.2  20.4  19.3
  22.   20.3  20.5  17.3  18.8  21.4  15.7  16.2  18.   14.3  19.2  19.6
  23.   18.4  15.6  18.1  17.4  17.1  13.3  17.8  14.   14.4  13.4  15.6
  11.8  13.8  15.6  14.6  17.8  15.4  21.5  19.6  15.3  19.4  17.   15.6
  13.1  41.3  24.3  23.3  27.   50.   50.   50.   22.7  25.   50.   23.8
  23.8  22.3  17.4  19.1  23.1  23.6  22.6  29.4  23.2  24.6  29.9  37.2
  39.8  36.2  37.9  32.5  26.4  29.6  50.   32.   29.8  34.9  37.   30.5
  36.4  31.1  29.1  50.   33.3  30.3  34.6  34.9  32.9  24.1  42.3  48.5
  50.   22.6  24.4  22.5  24.4  20.   21.7  19.3  22.4  28.1  23.7  25.
  23.3  28.7  21.5  23.   26.7  21.7  27.5  30.1  44.8  50.   37.6  31.6
  46.7  31.5  24.3  31.7  41.7  48.3  29.   24.   25.1  31.5  23.7  23.3
```

Рис. 1.12

Размер фактического массива больше, поэтому на данном экранном снимке представлена лишь часть его значений.

В пакете `sklearn` также доступны наборы графических данных. Каждый набор представлен матрицей размером 8×8. Загрузим эти изображения.

```
>>> digits = datasets.load_digits()
```

Выведем пятое изображение.

```
>>> print(digits.images[4])
```

В окне терминала отобразится следующий вывод (рис. 1.13).

Нетрудно заметить, что эти данные располагаются в восьми строках и восьми столбцах.

```
>>> print(digits.images[4])
[[ 0.  0.  0.  1. 11.  0.  0.  0.]
 [ 0.  0.  0.  7.  8.  0.  0.  0.]
 [ 0.  0.  1. 13.  6.  2.  2.  0.]
 [ 0.  0.  7. 15.  0.  9.  8.  0.]
 [ 0.  5. 16. 10.  0. 16.  6.  0.]
 [ 0.  4. 15. 16. 13. 16.  1.  0.]
 [ 0.  0.  0.  3. 15. 10.  0.  0.]
 [ 0.  0.  0.  2. 16.  4.  0.  0.]]
```

Рис. 1.13

Резюме

Из этой главы вы узнали, что такое ИИ и зачем его нужно изучать. Мы обсудили различные применения и подвиды ИИ. Мы рассмотрели тест Тьюринга и показали, как его можно выполнять. Вы узнали о том, как заставить машины рассуждать подобно людям. Мы обсудили понятие рационального агента и рассказали о способах его проектирования. Вы также узнали, что такое универсальный решатель задач (GPS) и как решать задачи с помощью GPS. Мы затронули тему разработки интеллектуальных агентов с помощью машинного обучения, указав при этом на существование различных типов моделей.

Кроме того, мы рассмотрели процесс установки Python 3 в различных операционных системах. Вы узнали о том, как установить пакеты, необходимые для создания приложений ИИ, и использовать их для загрузки данных, доступных в библиотеке `scikit-learn`. В следующей главе вы познакомитесь с методами обучения с учителем и узнаете о том, как создавать модели, предназначенные для решения задач классификации и регрессии.

2

Классификация и регрессия посредством обучения с учителем

Эта глава посвящена решению задач классификации и регрессии данных с помощью методов обучения с учителем. Вы ознакомитесь со следующими темами:

- в чем состоит различие между обучением с учителем и без учителя;
- что такое классификация данных;
- методы предварительной обработки данных;
- кодирование меток;
- создание классификаторов на основе логистической регрессии;
- что такое наивный байесовский классификатор;
- матрица неточностей;
- создание классификаторов на основе метода опорных векторов;
- что такое линейная и полиномиальная регрессия;
- построение линейного регрессора в случае простой и множественной регрессии;
- оценка стоимости недвижимости с использованием регрессора на основе опорных векторов.

Обучение с учителем и без учителя

Одним из наиболее распространенных способов наделения машины искусственным интеллектом является машинное обучение. Методы машинного обучения в целом делятся на две категории: обучение с учителем и обучение без учителя. Существуют и другие способы подобного разделения, но мы обсудим только этот.

Обучение с учителем (supervised learning; другое название: контролируемое обучение) — это процесс создания модели машинного обучения, основанной на маркированных (помеченных) обучающих (тренировочных) данных. Предположим, мы хотим создать систему, которая будет автоматически прогнозировать доход человека исходя из таких его характеристик, как возраст, образование, место жительства и т.п. Для этого мы должны создать базу данных, содержащую подробные сведения о людях, и пометить эти характеристики (признаки) соответствующими маркерами. Тем самым мы сообщаем нашему алгоритму, какие параметры соответствуют тому или иному доходу. На основании этого соответствия алгоритм будет учиться рассчитывать уровень дохода человека, используя предоставленные параметры.

Обучение без учителя (unsupervised learning; другие названия: неконтролируемое обучение, самообучение, спонтанное обучение) — это процесс создания модели машинного обучения, не основанный на использовании маркированных тренировочных данных. В некотором смысле это прямая противоположность процесса обучения с учителем, который обсуждался выше. В данном случае, ввиду отсутствия каких-либо отличительных меток, приходится обнаруживать отношения между данными, исходя лишь из самих данных. Предположим, мы хотим создать систему, предназначенную для разделения наборов точек данных на несколько групп. Проблема в том, что мы не можем точно указать, что именно должно использоваться в качестве критерия такого разделения. Следовательно, алгоритм обучения без учителя должен пытаться самостоятельно структурировать заданный набор данных путем разбиения на группы с использованием способа, который представляется наилучшим из всех возможных.

Что такое классификация

В этой главе мы обсудим методы классификации, основывающиеся на обучении с учителем. Сущность классификации заключается в разбиении данных на заданное количество классов. В ходе этого процесса мы распределяем данные по заданному фиксированному количеству категорий, чтобы обеспечить их наиболее целесообразное и эффективное использование.

В машинном обучении классификация решает задачу идентификации категории, к которой относится новая точка данных. Мы строим модель классификации на основании тренировочного набора, содержащего точки данных и соответствующие метки. Предположим, мы хотим проверить, содержит ли данная фотография изображение лица человека. Мы создаем тренировочный набор, включающий данные двух классов: `face` и `no-face`. Затем мы тренируем модель, задействуя тренировочные образцы, которыми располагаем.

После этого модель используется для выведения соответствующих умозаключений.

Хорошая классифицирующая система упрощает поиск и извлечение данных. Такие классификаторы широко применяются в системах распознавания лиц, идентификации спама, рекомендательных механизмах и т.п. Алгоритм классификации данных стремится определить наиболее подходящий критерий для разбиения данных на заданное количество классов.

Количество предоставляемых образцов должно быть достаточно большим, чтобы путем их *обобщения* можно было определить критерий разбиения данных на классы. При слишком малом количестве образцов алгоритм будет тяготеть в своих оценках к тренировочным данным. Это означает, что он не будет работать достаточно хорошо для неизвестных данных из-за *переобучения* модели на тренировочном наборе. В действительности в мире машинного обучения с этой проблемой приходится сталкиваться довольно часто. Этот фактор следует всегда учитывать при построении различных моделей машинного обучения.

Предварительная обработка данных

На практике мы имеем дело с большими объемами необработанных исходных данных. Алгоритмы машинного обучения рассчитаны на то, что, прежде чем они смогут начать процесс тренировки, получаемые данные будут отформатированы определенным образом. Чтобы привести данные к форме, приемлемой для алгоритмов машинного обучения, мы должны предварительно подготовить их и преобразовать в нужный формат. Покажем, как это делается.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from sklearn import preprocessing
```

Определим некоторую выборку данных.

```
input_data = np.array([[5.1, -2.9, 3.3],
                       [-1.2, 7.8, -6.1],
                       [3.9, 0.4, 2.1],
                       [7.3, -9.9, -4.5]])
```

Мы обсудим несколько различных методов предобработки данных:

- бинаризация;
- исключение среднего;

- масштабирование;
- нормализация.

Рассмотрим поочередно каждый из этих методов, начиная с первого.

Бинаризация

Этот процесс применяется в тех случаях, когда мы хотим преобразовать наши числовые значения в булевы. Воспользуемся встроенным методом для бинаризации входных данных, установив значение 2.1 в качестве порогового.

Добавим следующие строки в тот же файл Python.

```
# Бинаризация данных
data_binarized =
preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\nBinarized data:\n", data_binarized)
```

Выполнив этот код, вы получите следующий вывод.

```
Binarized data:
 [[ 1. 0. 1.]
 [ 0. 1. 0.]
 [ 1. 0. 0.]
 [ 1. 0. 0.]]
```

Как видите, все значения свыше 2.1 принудительно устанавливаются равными 1. Остальные значения становятся равными 0.

Исключение среднего

Исключение среднего — методика предварительной обработки данных, обычно используемая в машинном обучении. Как правило, из *векторов признаков* (feature vectors) целесообразно исключать средние значения, чтобы каждый *признак* (feature) центрировался на нуле. Это делается с той целью, чтобы исключить из рассмотрения смещение значений в векторах признаков.

Добавим следующие строки в тот же файл Python, который использовался в предыдущем разделе.

```
# Вывод среднего значения и стандартного отклонения
print("\nBEFORE:")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
```

Две предыдущие строки отображают среднее значение и стандартное отклонение для входных данных. Исключим среднее.

```
# Исключение среднего
data_scaled = preprocessing.scale(input_data)
print("\nAFTER:")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))
```

После выполнения этого кода в окне терминала отобразится следующая информация.

```
BEFORE:
Mean = [ 3.775 -1.15 -1.3 ]
Std deviation = [ 3.12039661 6.36651396 4.0620192 ]
AFTER:
Mean = [ 1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [ 1. 1. 1.]
```

Нетрудно заметить, что среднее значение практически равно 0, а стандартное отклонение — 1.

Масштабирование

В нашем векторе признаков каждое значение может меняться в некоторых случайных пределах. Поэтому очень важно масштабировать признаки, чтобы они представляли собой ровное игровое поле для тренировки алгоритма машинного обучения. Мы не хотим, чтобы любой из признаков мог принимать искусственно большое или малое значение лишь в силу природы измерений.

Добавим в тот же файл Python следующие строки.

```
# Масштабирование MinMax
data_scaler_minmax =
preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax =
data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)
```

После выполнения этого кода в окне терминала отобразится следующая информация.

```
Min max scaled data:
 [[ 0.74117647  0.39548023  1.        ]
 [ 0.          1.          0.        ]
 [ 0.6         0.5819209   0.87234043]
 [ 1.          0.          0.17021277]]
```

Каждая строка масштабирована так, чтобы максимальным значением было 1, а все остальные значения определялись относительно него.

Нормализация

Процесс нормализации заключается в изменении значений в векторе признаков таким образом, чтобы для их измерения можно было использовать одну общую шкалу. В машинном обучении используются различные формы нормализации. В наиболее распространенных из них значения изменяются так, чтобы их сумма была равна 1. **L1-нормализация**, использующая **метод наименьших абсолютных отклонений** (Least Absolute Deviations), обеспечивает равенство 1 суммы абсолютных значений в каждом ряду. **L2-нормализация**, использующая **метод наименьших квадратов**, обеспечивает равенство 1 суммы квадратов значений.

Вообще говоря, техника L1-нормализации считается более надежной по сравнению с L2-нормализацией, поскольку она менее чувствительна к выбросам. Очень часто данные содержат выбросы, и с этим ничего не поделаешь. Мы хотим использовать безопасные методики, позволяющие игнорировать выбросы в процессе вычислений. Если бы мы решали задачу, в которой выбросы играют важную роль, то, вероятно, лучшим выбором была бы L2-нормализация.

Добавим следующие строки в тот же самый файл Python.

```
# Нормализация данных
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nL1 normalized data:\n", data_normalized_l1)
print("\nL2 normalized data:\n", data_normalized_l2)
```

Выполнив этот код, вы увидите в окне терминала следующий вывод.

```
L1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]
L2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Код всех примеров, приведенных в этом разделе, содержится в файле `data_preprocessor.py`.

Кодирование меток

Как правило, в процессе классификации данных мы имеем дело со множеством *меток* (labels). Ими могут быть слова, числа или другие объекты. Функции машинного обучения, входящие в библиотеку `sklearn`, ожидают, что метки являются числами. Поэтому, если метки — это уже числа, мы можем использовать их непосредственно для того, чтобы начать тренировку. Однако обычно это не так.

На практике метками служат слова, поскольку в таком виде они лучше всего воспринимаются человеком. Мы помечаем тренировочные данные словами, чтобы облегчить отслеживание соответствий. Для преобразования слов в числа необходимо использовать кодирование. Под *кодированием меток* (label encoding) подразумевается процесс преобразования словесных меток в числовую форму. Благодаря этому алгоритмы могут оперировать нашими данными.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from sklearn import preprocessing
```

Определим метки.

```
# Предоставление меток входных данных
input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow',
'white']
```

Создадим объект кодирования меток и обучим его.

```
# Создание кодировщика и установление соответствия
# между метками и числами
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

Выведем отображение слов на числа.

```
# Вывод отображения
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_):
    print(item, '-->', i)
```

Преобразуем набор случайно упорядоченных меток, чтобы проверить работу кодировщика.

```
# Преобразование меток с помощью кодировщика
test_labels = ['green', 'red', 'black']
```

```
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))
```

Декодируем случайный набор чисел.

```
# Декодирование набора чисел с помощью декодера
encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list(decoded_list))
```

После выполнения этого кода в окне терминала должна отобразиться следующая информация (рис. 2.1).

```
Label mapping:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']
```

Рис. 2.1

Вам не составит труда убедиться в том, что шаги кодирования и декодирования меток выполнены корректно. Код примеров для этого раздела содержится в файле `label_encoder.py file`.

Логистический классификатор

Логистическая регрессия (logistic regression) — это методика, применяемая для объяснения отношений между входными и выходными переменными. Входные переменные считаются независимыми, выходные — зависимыми. Зависимая переменная может иметь лишь фиксированный набор значений. Эти значения соответствуют классам задачи классификации.

Нашей целью является идентификация отношений между независимыми и зависимыми переменными посредством оценки вероятностей того, что та или иная зависимая переменная относится к тому или иному классу. *Логистическая функция* — это **сигмоида**, используемая для создания функций с различными параметрами. Она очень тесно связана с анализом данных на основе *обобщенной линейной модели*, в соответствии с которой делается попытка

подогнать прямую линию к группе точек таким образом, чтобы минимизировать ошибку. Вместо *линейной* регрессии мы применяем *логистическую* регрессию. В действительности сама по себе логистическая регрессия предназначена не для классификации данных, однако она позволяет упростить решение этой задачи. Ввиду ее простоты логистическую регрессию часто задействуют в машинном обучении. Рассмотрим пример применения логистической регрессии для создания классификатора. Прежде чем продолжить, убедитесь, что в вашей системе установлен пакет `Tkinter`. В случае необходимости вы сможете найти его по адресу `https://docs.python.org/2/library/tkinter.html`.

Создайте новый файл Python и импортируйте указанные ниже пакеты. Мы будем импортировать одну из функций, содержащихся в файле `utilities.py`. Вскоре мы познакомим вас с этой функцией, а пока что просто импортируйте ее.

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

from utilities import visualize_classifier
```

Определим образец входных данных с помощью двумерных векторов и соответствующих меток.

```
# Определение образца входных данных
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5],
[6, 5], [5.6, 5], [3.3, 0.4], [3.9, 0.9], [2.8, 1],
[0.5, 3.4], [1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])
```

Мы будем тренировать классификатор, используя эти помеченные данные. Создадим объект логистического классификатора.

```
# Создание логистического классификатора
classifier = linear_model.LogisticRegression(solver='liblinear',C=1)
```

Обучим классификатор, используя определенные выше данные.

```
# Тренировка классификатора
classifier.fit(X, y)
```

Визуализируем результаты работы классификатора, отследив границы классов.

```
# Визуализация работы классификатора
visualize_classifier(classifier, X, y)
```

Мы должны определить эту функцию, прежде чем ее можно будет применить. Поскольку она будет использоваться в этой главе несколько раз, целесообразно поместить ее в отдельный файл и импортировать, когда она потребуется. Данная функция предоставляется вам в файле `utilities.py`.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
```

Создадим определение вышеупомянутой функции, используя объект классификатора, входные данные и метки в качестве входных параметров.

```
def visualize_classifier(classifier, X, y):
    # Определение для X и Y минимального и максимального
    # значений, которые будут использоваться при построении сетки
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
```

Мы определили минимальное и максимальное значения для координат вдоль осей X и Y, которые будут использоваться в нашей сетке. По сути, эта сетка представляет собой набор значений для вычисления функции, чтобы можно было визуализировать границы классов. Определим шаг сетки и создадим ее, используя заданные минимальные и максимальные значения.

```
    # Определение величины шага для построения сетки
    mesh_step_size = 0.01

    # Определение сетки для значений X и Y
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x,
mesh_step_size), np.arange(min_y, max_y, mesh_step_size))
```

Запустим классификатор для всех точек сетки.

```
    # Выполнение классификатора на сетке данных
    output = classifier.predict(np.c_[x_vals.ravel(),y_vals.ravel()])

    # Переформирование выходного массива
    output = output.reshape(x_vals.shape)
```

Создадим график, выберем цветовую схему и разместим все точки.

```
    # Создание графика
    plt.figure()

    # Выбор цветовой схемы для графика
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)
```

```
    # Размещение тренировочных точек на графике
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black',
linewidth=1, cmap=plt.cm.Paired)
```

Укажем границы графика, используя минимальные и максимальные значения, добавим деления и отобразим график.

```
    # Определение границ графика
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    # Определение делений на осях X и Y
    plt.xticks((np.arange(int(X[:, 0].min() - 1),
int(X[:, 0].max() + 1), 1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1),
int(X[:, 1].max() + 1), 1.0)))

    plt.show()
```

После выполнения этого кода на экране отобразится графическое окно, представленное на рис. 2.2.

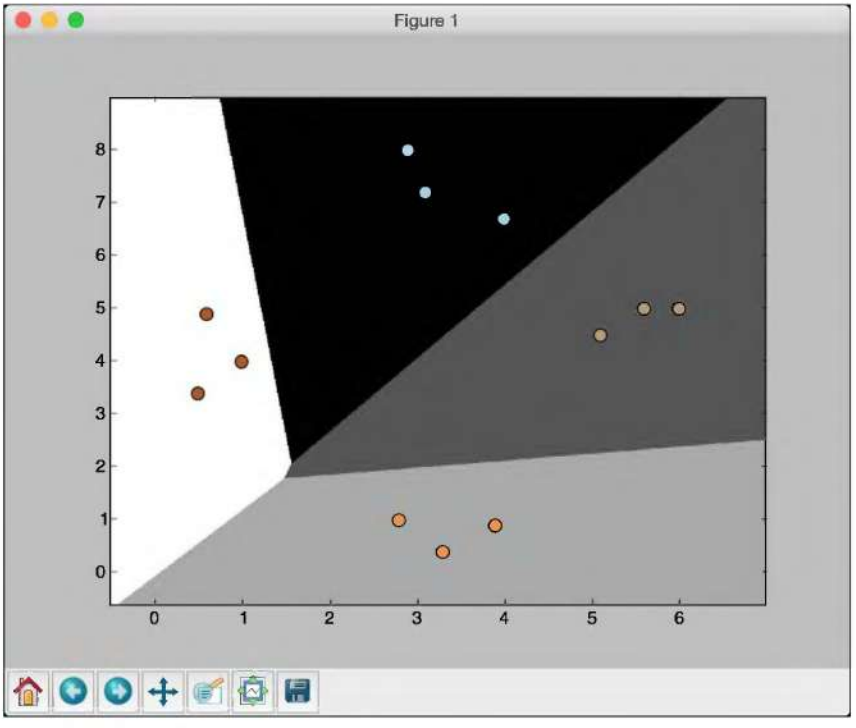

Рис. 2.2. (См. цветную вклейку; адрес указан во введении)

Если вы используете для C значение 100, как показано в следующей строке, то увидите, что границы станут более точными.

```
classifier = linear_model.LogisticRegression(solver='liblinear', C=100)
```

Это объясняется тем, что параметр C налагает определенный штраф на неточности классификации, поэтому алгоритм стремится лучше приспособиться к тренировочным данным. Значение этого параметра необходимо очень тщательно подбирать, поскольку его чрезмерное увеличение приведет к переобучению модели на тренировочных данных, и она не будет хорошо обобщаться.

Выполнив этот код с параметром C, равным 100, вы получите на экране следующее изображение (рис. 2.3).

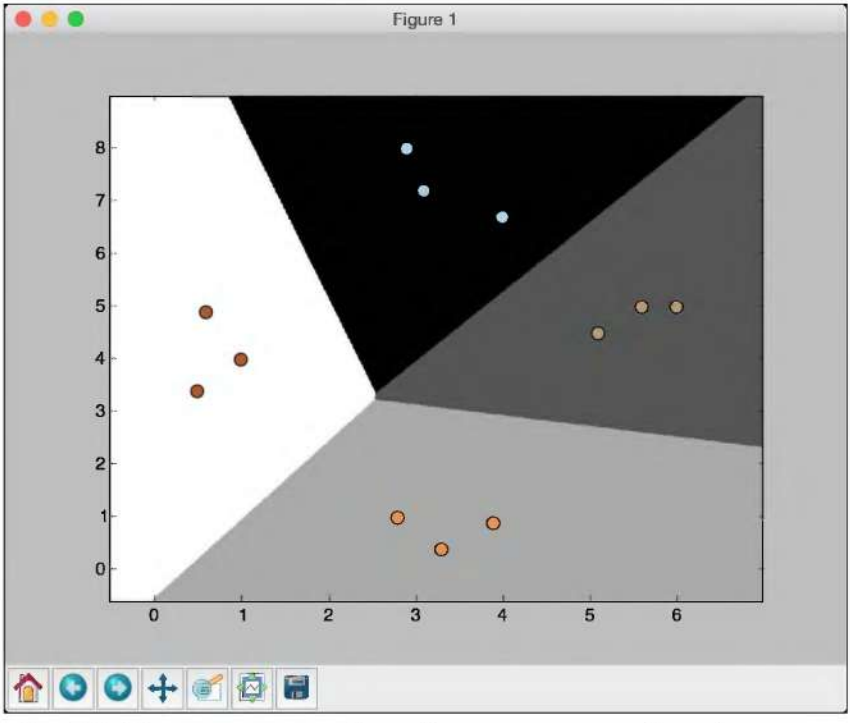

Рис. 2.3. (См. цветную вклейку; адрес указан во введении)

По сравнению с предыдущим рисунком границы улучшились. Код для этого раздела содержится в файле logistic_regression.py.

Наивный байесовский классификатор

Наивный байесовский классификатор (Naïve Bayes classifier) — это простой классификатор, основанный на использовании теоремы Байеса, которая описывает вероятность события с учетом связанных с ним условий. Такой классификатор создается посредством присваивания меток классов экземплярам задачи. Последние представляются в виде векторов значений признаков. При этом предполагается, что значение любого заданного признака не зависит от значений других признаков. Это предположение о независимости рассматриваемых признаков и составляет *наивную* часть байесовского классификатора.

Мы можем оценивать влияние любого признака переменной класса независимо от влияния других признаков. Например, мы можем считать животное гепардом, если оно имеет пятнистую кожу, четыре лапы и хвост и развивает скорость, равную примерно 70 миль в час. В случае использования наивного байесовского классификатора считается, что каждый из признаков вносит независимый вклад в конечный результат, оценивающий вероятность того, что животное является гепардом. Мы не будем утруждать себя рассмотрением корреляции между рисунком кожи, количеством лап, наличием хвоста и скоростью перемещения.

Займемся созданием наивного байесовского классификатора. Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn import cross_validation

from utilities import visualize_classifier
```

В качестве источника данных мы будем использовать файл `data_multivar_nb.txt`, каждая строка которого содержит значения, разделенные запятой.

```
# Входной файл, содержащий данные
input_file = 'data_multivar_nb.txt'
```

Загрузим данные из этого файла.

```
# Загрузка данных из входного файла
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Создадим экземпляр наивного байесовского классификатора. В данном случае мы будем использовать *гауссовский* наивный байесовский классификатор, в котором предполагается, что значения, ассоциируемые с каждым классом, следуют закону распределения Гаусса.

```
# Создание наивного байесовского классификатора
classifier = GaussianNB()
```

Обучим классификатор, используя тренировочные данные.

```
# Тренировка классификатора
classifier.fit(X, y)
```

Запустим классификатор на тренировочных данных и спрогнозируем результаты.

```
# Прогнозирование значений для тренировочных данных
y_pred = classifier.predict(X)
```

Вычислим *качество* (accuracy)[1] классификатора, сравнив предсказанные значения с истинными метками, а затем визуализируем результат.

```
# Вычисление качества классификатора
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")
# Визуализация результатов работы классификатора
visualize_classifier(classifier, X, y)
```

Предыдущий метод вычисления качества классификатора не является надежным. Нам нужно выполнить перекрестную проверку, чтобы не использовать те же самые тренировочные данные при тестировании.

Разобьем данные на обучающий и тестовый наборы. В соответствии со значением параметра `test_size`, указанным в строке кода ниже, мы отнесем 80% данных к тренировке, а оставшиеся 20% — к тестированию. Затем мы выполним тренировку наивного байесовского классификатора на этих данных.

```
# Разбивка данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
cross_validation.train_test_split(X, y, test_size=0.2, random_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)
```

[1] Отношение общего количества правильных прогнозов к общему количеству рассмотренных точек данных. — *Примеч. ред.*

Вычислим качество классификатора и визуализируем результаты.

```
# Вычисление качества классификатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

# Визуализация работы классификатора
visualize_classifier(classifier_new, X_test, y_test)
```

Воспользуемся встроенными функциями для вычисления *качества* (accuracy), *точности* (precision)[2] и *полноты* (recall)[3] классификатора на основании тройной перекрестной проверки.

```
num_folds = 3
accuracy_values = cross_validation.cross_val_score(classifier,
        X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100*accuracy_values.mean(),
        2)) + "%")

precision_values = cross_validation.cross_val_score(classifier,
         X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100*precision_values.mean(),
         2)) + "%")
recall_values = cross_validation.cross_val_score(classifier,
         X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100*recall_values.mean(),
         2)) + "%")
f1_values = cross_validation.cross_val_score(classifier,
         X, y, scoring='f1_weighted', cv=num_folds)
print("F1: " + str(round(100*f1_values.mean(), 2)) + "%")
```

Выполнив этот код, вы получите для первого тренировочного прогона следующее изображение на экране (рис. 2.4).

На этом экранном снимке показаны границы, полученные с помощью классификатора. Как видите, они корректно разделяют 4 кластера и создают области с границами, выявленными на основании распределения входных точек данных. На рис. 2.5 представлены результаты второго тренировочного прогона с перекрестной проверкой.

---

[2] Отношение количества правильно предсказанных значений данного класса к общему количеству его предсказанных значений. — *Примеч. ред.*

[3] Отношение количества правильно предсказанных значений данного класса к общему количеству его фактических значений. — *Примеч. ред.*

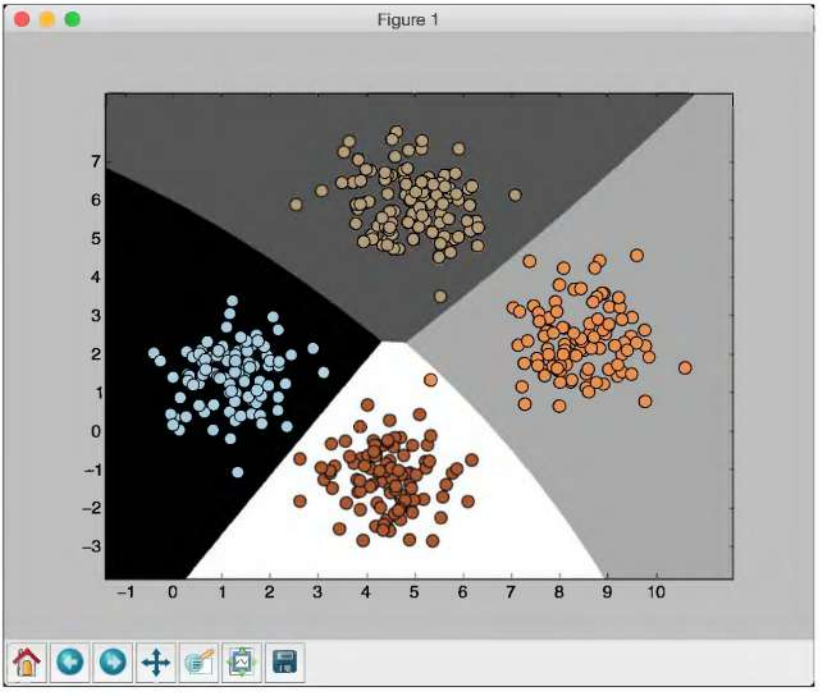

Рис. 2.4. (См. цветную вклейку; адрес указан во введении)

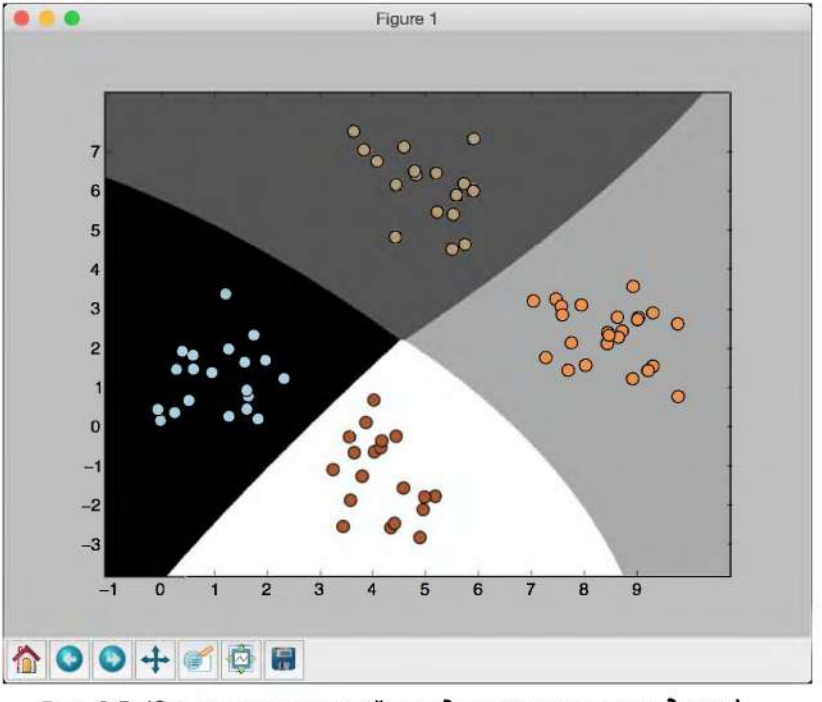

Рис. 2.5. (См. цветную вклейку; адрес указан во введении)

В окне терминала отобразится следующий вывод.

```
Accuracy of Naïve Bayes classifier = 99.75 %
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Код для этого раздела содержится в файле `naive_bayes.py`.

Матрица неточностей

Матрица неточностей (confusion matrix) — это таблица, используемая для описания эффективности классификатора. Обычно она извлекается из тестового набора данных, для которого известны базовые истинные значения. Мы анализируем результаты отнесения каждого класса и определяем долю неверно отнесенных классов. В процессе конструирования вышеупомянутой таблицы мы фактически имеем дело с несколькими ключевыми метриками, играющими очень важную роль в машинном обучении. Обратимся к бинарной классификации, когда выходная переменная имеет два возможных значения: *0* и *1*.

- **Истинноположительные результаты.** Это случаи, для которых мы предсказали 1 в качестве результата, и при этом истинный результат также равен 1.
- **Истинноотрицательные результаты.** Это случаи, для которых мы предсказали 0 в качестве результата, и при этом истинный результат также равен 0.
- **Ложноположительные результаты.** Это случаи, для которых мы предсказали 1 в качестве результата, тогда как истинный результат равен 0. Такая ситуация известна как *ошибка I рода*.
- **Ложноотрицательные результаты.** Это случаи, для которых мы предсказали 0 в качестве результата, тогда как истинный результат равен 1. Такая ситуация известна как *ошибка II рода*.

В зависимости от специфики конкретной задачи может потребоваться оптимизация алгоритма таким образом, чтобы уменьшить долю ложноположительных или ложноотрицательных результатов. Например, в системах биометрической идентификации личности очень важно избежать ложноположительных результатов, чтобы злоумышленники не могли

получить доступ к конфиденциальной информации. Давайте посмотрим, как строится матрица неточностей.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Определим выборки фактических (калибровочных) и предсказанных меток классов.

```
# Определение выборочных меток
true_labels = [2, 0, 0, 2, 4, 4, 1, 0, 3, 3, 3]
pred_labels = [2, 1, 0, 2, 4, 3, 1, 0, 1, 3, 3]
```

Построим матрицу неточностей, используя только что определенные метки.

```
# Построение матрицы неточностей
confusion_mat = confusion_matrix(true_labels, pred_labels)
```

Визуализируем матрицу неточностей.

```
# Визуализация матрицы неточностей
plt.imshow(confusion_mat, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Матрица неточностей')
plt.colorbar()
ticks = np.arange(5)
plt.xticks(ticks, ticks)
plt.yticks(ticks, ticks)
plt.ylabel('Калибровочные метки')
plt.xlabel('Предсказанные метки')
plt.show()
```

В коде визуализации переменная `ticks` ссылается на количество различных классов. В нашем случае имеется пять различных меток.

Выведем отчет о результатах классификации.

```
# Отчет о результатах классификации
targets = ['Class-0', 'Class-1', 'Class-2', 'Class-3', 'Class-4']
print('\n', classification_report(true_labels, pred_labels,
      target_names=targets))
```

Выведенный отчет содержит оценки эффективности классификатора по отношению к каждому из классов. Выполнив код, вы получите следующий экранный снимок (рис. 2.6).

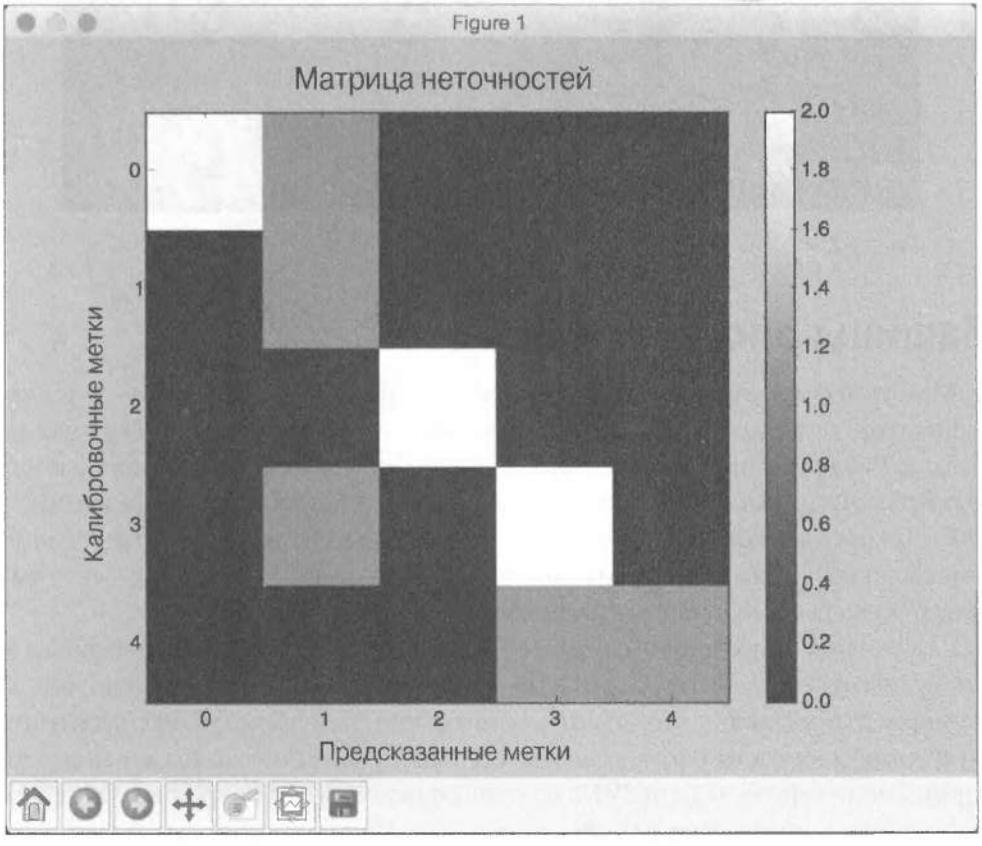

Рис. 2.6

Белому цвету соответствуют более высокие значения, черному — более низкие, на что указывает расположенная справа градиентная шкала. В идеальном сценарии все диагональные квадраты были бы белыми, а все остальные — черными, что соответствовало бы 100% качеству.

В окне терминала отобразятся следующие данные (рис. 2.7).

Код примеров для этого раздела содержится в файле `confusion_matrix.py`.

	precision	recall	f1-score	support
Class-0	1.00	0.67	0.80	3
Class-1	0.33	1.00	0.50	1
Class-2	1.00	1.00	1.00	2
Class-3	0.67	0.67	0.67	3
Class-4	1.00	0.50	0.67	2
avg / total	0.85	0.73	0.75	11

Рис. 2.7

Машины опорных векторов

Машина опорных векторов (Support Vector Machine — SVM) — это классификатор, определенный с использованием гиперплоскости, разделяющей классы. **Разделяющая гиперплоскость** представляет собой N-мерную версию прямой линии. При условии, что в задаче бинарной классификации задан маркированный обучающий набор данных, SVM находит оптимальную гиперплоскость, которая делит данные на два класса. Этот метод легко распространяется на задачу с N классами.

Рассмотрим двумерный случай с двумя классами точек. Таким образом, мы будем иметь дело только с точками и прямыми, расположенными на двумерной плоскости, что значительно проще, чем визуализировать векторы и гиперплоскости в пространстве большей размерности. Конечно же, это упрощенная версия задачи SVM, но очень важно рассмотреть и визуализировать данный случай, прежде чем переходить к наборам данных более высоких размерностей.

Рассмотрим рис. 2.8.

Как видите, мы имеем два класса точек и хотим найти оптимальную гиперплоскость, разделяющую эти классы. Однако что именно следует считать критерием оптимальности? На рисунке сплошная линия представляет оптимальную гиперплоскость. Можно провести множество прямых линий, разделяющих данные классы точек, но эта линия является наилучшим разделителем, поскольку она максимизирует расстояния, на которые точки удалены от разделяющей их линии. Точки, расположенные на пунктирных линиях, называются *опорными векторами*. Расстояние между двумя пунктирными линиями называется *максимальным зазором*.

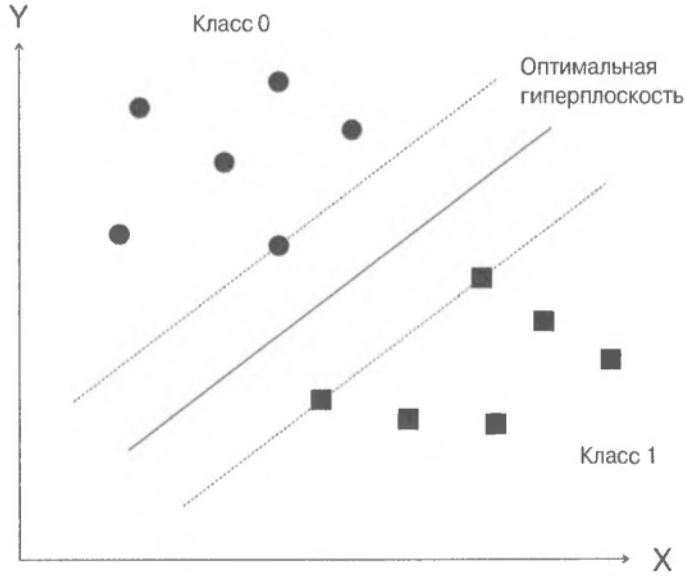

Рис. 2.8

Классификация данных о доходах с помощью машин опорных векторов

Создадим классификатор в виде машины опорных векторов, предназначенный для прогнозирования границ дохода заданного физического лица на основе 14 атрибутов. Нашей целью является выяснение условий, при которых ежегодный доход человека превышает *$50000* или меньше этой величины. Следовательно, мы имеем дело с задачей бинарной классификации. Мы воспользуемся набором данных о доходах, предоставленным Бюро переписи населения США по адресу `https://archive.ics.uci.edu/ml/datasets/Census+Income`. Следует отметить одну особенность этого набора, которая заключается в том, что каждая точка данных представляет собой сочетание текста и чисел. Мы не можем использовать эти данные в необработанном виде, поскольку алгоритмам неизвестно, как обрабатывать слова. Мы также не можем преобразовать все данные, используя кодирование меток, поскольку числовые данные также содержат ценную информацию. Следовательно, чтобы создать эффективный классификатор, мы должны использовать комбинацию кодировщиков меток и необработанных числовых данных.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn import cross_validation
```

Для загрузки данных мы используем файл `income_data.txt`, содержащий подробные сведения о доходах населения.

```
# Входной файл, содержащий данные
input_file = 'income_data.txt'
```

Загружаемые из файла данные нужно подготовить к классификации, подвергнув их предварительной обработке. Для каждого класса мы будем использовать не более 25000 точек данных.

```
# Чтение данных
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000
```

Откроем файл и прочитаем строки.

```
with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and
           count_class2 >= max_datapoints:
            break

        if '?' in line:
            continue
```

Каждая строка данных отделяется от следующей с помощью запятой, что требует соответствующего разбиения строк. Последним элементом каждой строки является метка. В зависимости от этой метки мы будем относить данные к тому или иному классу.

```
        data = line[:-1].split(', ')

        if data[-1] == '<=50K' and
           count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
```

```
        if data[-1] == '>50K' and
           count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1
```

Преобразуем список в массив `array`, чтобы его можно было использовать в качестве входных данных функций `sklearn`.

```
# Преобразование в массив numpy
X = np.array(X)
```

Если атрибут — строка, то он нуждается в кодировании. Если атрибут — число, мы можем оставить его в том виде, как он есть. Заметьте, что в конечном счете мы получим несколько кодировщиков меток, которые нам нужно будет отслеживать.

```
# Преобразование строковых данных в числовые
label_encoder = []
X_encoded = np.empty(X.shape)
for i,item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1]
            .fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

Создадим SVM-классификатор с линейным ядром.

```
# Создание SVM-классификатора
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
```

Обучим классификатор.

```
# Обучение классификатора
classifier.fit(X, y)
```

Выполним перекрестную проверку, разбив данные на обучающий и тестовый наборы в пропорции 80/20, а затем спрогнозируем результат для тренировочных данных.

```
# Перекрестная проверка
X_train, X_test, y_train, y_test = cross_validation
.train_test_split(X, y, test_size=0.2, random_state=5)
```

```
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)
```

Вычислим для классификатора F-меру.

```
# Вычисление F-меры для SVM-классификатора
f1 = cross_validation.cross_val_score(classifier, X, y,
scoring='f1_weighted', cv=3)
print("F1 score: " + str(round(100*f1.mean(), 2)) + "%")
```

Теперь, имея подготовленный классификатор, посмотрим, что произойдет, если мы выберем некоторую случайную точку данных и предскажем для нее результат. Определим одну такую точку.

```
# Предсказание результата для тестовой точки данных
input_data = ['37', 'Private', '215646', 'HS-grad', '9',
'Never-married', 'Handlers-cleaners', 'Not-in-family', 'White',
'Male', '0', '0', '40', 'United-States']
```

Прежде чем приступить к прогнозированию, мы должны присвоить этой точке данных код, используя созданный ранее кодировщик признаков.

```
# Кодирование тестовой точки данных
input_data_encoded = [-1] * len(input_data)
count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(input_data[i])
    else:
        input_data_encoded[i] =
int(label_encoder[count].transform([input_data[i]]))
        count += 1
input_data_encoded = np.array([input_data_encoded])
```

Теперь мы готовы к тому, чтобы спрогнозировать результат с помощью классификатора.

```
# Выполнение классификатора для кодированной точки данных
# и вывод результата
predicted_class = classifier.predict(input_data_encoded)
print(label_encoder[-1].inverse_transform(predicted_class)[0])
```

Тренировка классификатора посредством этого кода займет несколько секунд. После выполнения кода в окне терминала отобразится следующая F-мера.

```
F1 score: 66.82%
```

Кроме того, для тестовой точки данных отобразится следующий вывод:

```
<=50K
```

Проверив значения в этой точке данных, вы убедитесь в том, что она близко соответствует точкам данных, принадлежащим классу с доходом менее $50000. Вы можете изменить характеристики работы классификатора (F-мера, точность, полнота), используя различные ядра и тестируя несколько комбинаций параметров.

Код примеров для этого раздела содержится в файле `income_classifier.py`.

Что такое регрессия

Регрессия — это процесс оценки того, как соотносятся между собой входные и выходные переменные. Следует отметить, что выходные переменные могут иметь значения из непрерывного ряда вещественных чисел. Следовательно, существует бесконечное количество результирующих возможностей. Это резко контрастирует с процессом классификации, в котором количество выходных классов фиксировано.

В регрессии предполагается, что выходные переменные зависят от входных, и наша задача заключается в выяснении соотношения между ними. Отсюда входные переменные называют *независимыми переменными* (или *предикторами*), а выходные — *зависимыми* (или *критериальными переменными*). При этом вовсе не требуется, чтобы входные переменные были независимыми друг от друга. Существует множество ситуаций, когда между входными переменными существует корреляция.

Регрессионный анализ позволяет выяснить, как изменяется значение выходной переменной, когда мы изменяем лишь часть входных переменных, оставляя остальные входные переменные фиксированными. В случае линейной регрессии предполагается, что входные и выходные переменные связаны между собой линейной зависимостью. Это налагает ограничения на нашу процедуру моделирования, но ускоряет ее и делает более эффективной.

Иногда линейной регрессии оказывается недостаточно для объяснения соотношений между входными и выходными переменными. В подобных случаях мы используем **полиномиальную регрессию**, в которой входные и выходные переменные связаны между собой полиномиальной зависимостью. С вычислительной точки зрения такой подход более сложен, но обеспечивает более высокую точность. Выбор вида регрессии для выявления указанных соотношений определяется конкретикой задачи. Регрессию часто используют для прогнозирования цен, экономических показателей и т.п.

Создание регрессора одной переменной

Приступим к построению регрессионной модели на основе одной переменной. Создайте новый файл Python и импортируйте следующие пакеты.

```
import pickle

import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt
```

Мы также будем использовать файл `data_singlevar_regr.txt`, содержащий исходные данные.

```
# Входной файл, содержащий данные
input_file = 'data_singlevar_regr.txt'
```

В этом текстовом файле в качестве разделителя используется запятая, поэтому для загрузки данных можно воспользоваться следующим вызовом функции.

```
# Загрузка данных
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбивка данных на обучающий и тестовый наборы
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

# Тренировочные данные
X_train, y_train = X[:num_training], y[:num_training]

# Тестовые данные
X_test, y_test = X[num_training:], y[num_training:]
```

Создадим объект линейного регрессора и обучим его, используя тренировочные данные.

```
# Создание объекта линейного регрессора
regressor = linear_model.LinearRegression()

# Обучение модели с использованием обучающего набора
regressor.fit(X_train, y_train)
```

Спрогнозируем результат для тестового набора данных, используя обучаемую модель.

```
# Прогнозирование результата
y_test_pred = regressor.predict(X_test)
```

Построим выходной график.

```
# Построение графика
plt.scatter(X_test, y_test, color='green')
plt.plot(X_test, y_test_pred, color='black', linewidth=4)
plt.xticks(())
plt.yticks(())
plt.show()
```

Вычислим метрические характеристики регрессора, сравнивая истинные значения с предсказанными.

```
# Вычисление метрических характеристик
print("Linear regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test,
      y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test,
      y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(
      y_test, y_test_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(
      y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

Создав модель, мы можем сохранить ее в файле для последующего использования. Python предоставляет отличный модуль, который позволяет легко это сделать.

```
# Файл для сохранения модели
output_model_file = 'model.pkl'

# Сохранение модели
with open(output_model_file, 'wb') as f:
    pickle.dump(regressor, f)
```

Загрузим модель из файла на диске и построим прогноз.

```
# Загрузка модели
with open(output_model_file, 'rb') as f:
    regressor_model = pickle.load(f)
```

```
# Получение прогноза на тестовом наборе данных
y_test_pred_new = regressor_model.predict(X_test)
print("\nNew mean absolute error =",
      round(sm.mean_absolute_error(y_test,
      y_test_pred_new), 2))
```

Выполнив этот код, вы получите следующее изображение (рис. 2.9).

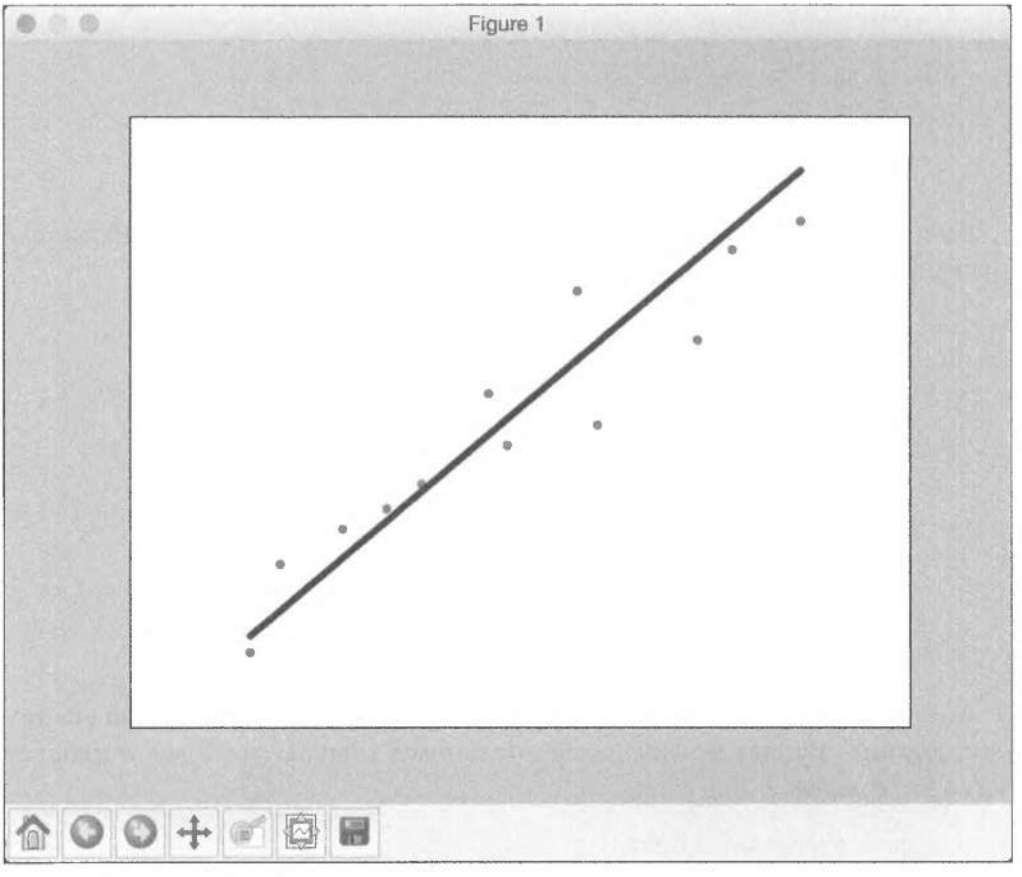

Рис. 2.9

Кроме того, в окне терминала отобразится следующая информация.

```
Linear regressor performance:
Mean absolute error = 0.59
Mean squared error = 0.49
Median absolute error = 0.51
Explain variance score = 0.86
R2 score = 0.86
New mean absolute error = 0.59
```

Код примеров для этого раздела содержится в файле regressor_singlevar.py.

Создание многомерного регрессора

В предыдущем разделе мы обсудили построение регрессионной модели для одной переменной и теперь можем перейти к рассмотрению многомерных данных. Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures
```

Мы также будем использовать файл data_multivar_regr.txt.

```
# Входной файл, содержащий данные
input_file = 'data_multivar_regr.txt'
```

В этом текстовом файле в качестве разделителя используется запятая, поэтому для загрузки данных можно воспользоваться следующим вызовом функции.

```
# Загрузка данных
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбивка данных на обучающий и тестовый наборы
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

# Тренировочные данные
X_train, y_train = X[:num_training], y[:num_training]

# Тестовые данные
X_test, y_test = X[num_training:], y[num_training:]
```

Создадим и обучим модель линейного регрессора.

```
# Создание модели линейного регрессора
linear_regressor = linear_model.LinearRegression()

# Обучение модели с использованием обучающих наборов
linear_regressor.fit(X_train, y_train)
```

Спрогнозируем результат для тестового набора данных.

```
# Прогнозирование результата
y_test_pred = linear_regressor.predict(X_test)
```

Выведем метрические характеристики.

```
# Измерение метрических характеристик
print("Linear Regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test,
      y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test,
      y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test,
      y_test_pred), 2))
print("Explained variance score =", round(sm.explained_variance_score(
      y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

Создадим полиномиальный регрессор степени 10 и обучим его на тренировочных данных. Возьмем некоторую выборочную точку данных и спрогнозируем для нее результат. Первый шаг заключается в том, чтобы преобразовать ее в полином.

```
# Полиномиальная регрессия
polynomial = PolynomialFeatures(degree=10)
X_train_transformed = polynomial.fit_transform(X_train)
datapoint = [[7.75, 6.35, 5.56]]
poly_datapoint = polynomial.fit_transform(datapoint)
```

Нетрудно заметить, что эта точка весьма близка к точке данных *[7.66, 6.29, 5.66]*, указанной в строке 11 нашего файла данных. Поэтому удачно созданный регрессор должен предсказать результат, близкий к `41.35`. Создайте объект линейного регрессора и выполните подгонку к полиному. Постройте прогноз с использованием как линейного, так и полиномиального регрессоров, чтобы увидеть разницу.

```
poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(X_train_transformed, y_train)
print("\nLinear regression:\n",
      linear_regressor.predict(datapoint))
print("\nPolynomial regression:\n",
      poly_linear_model.predict(poly_datapoint))
```

После выполнения этого кода в окне терминала отобразится следующая информация.

```
Linear Regressor performance:
Mean absolute error = 3.58
Mean squared error = 20.31
Median absolute error = 2.99
Explained variance score = 0.86
R2 score = 0.86
```

Дополнительно будут выведены следующие метрические характеристики.

```
Linear regression:
 [ 36.05286276]
Polynomial regression:
 [ 41.46961676]
```

Как нетрудно заметить, по сравнению с линейным регрессором полиномиальный регрессор обеспечивает получение результата, более близкого к значению 41.35. Код примеров для этого раздела содержится в файле regressor_multivar.py.

Оценка стоимости недвижимости с использованием регрессора на основе машины опорных векторов

Применим концепцию SVM для создания регрессора, позволяющего оценивать стоимость жилья. Для этого мы воспользуемся набором данных из пакета sklearn, в котором каждая точка определяется 13 атрибутами. Наша задача заключается в прогнозировании стоимости жилья на основе значений этих атрибутов.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from sklearn import datasets
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle
```

Загрузим набор данных с ценами на жилье.

```
# Загрузка данных с ценами на жилье
data = datasets.load_boston()
```

Перемешаем данные, чтобы усилить объективность анализа.

```
# Перемешивание данных
X, y = shuffle(data.data, data.target, random_state=7)
```

Разобьем данные на обучающий и тестовый наборы в соотношении 80/20.

```
# Разбивка данных на обучающий и тестовый наборы
num_training = int(0.8 * len(X))
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]
```

Создадим и обучим SVM-регрессор, используя линейное ядро. Параметр `C` представляет штраф за ошибки обучения. Если вы увеличите значение `C`, то модель более точно подстроится под обучающие данные. Однако это может привести к переобучению модели и потере ее общности. Параметр `epsilon` задает пороговое значение: штраф за ошибки в обучении не налагается, если расхождение между фактическим и предсказанным значениями не превышает этой величины.

```
# Создание регрессионной модели на основе SVM
sv_regressor = SVR(kernel='linear', C=1.0, epsilon=0.1)

# Обучение регрессора SVM
sv_regressor.fit(X_train, y_train)
```

Оценим эффективность работы регрессора и выведем метрические характеристики.

```
# Оценка эффективности работы регрессора
y_test_pred = sv_regressor.predict(X_test)
mse = mean_squared_error(y_test, y_test_pred)
evs = explained_variance_score(y_test, y_test_pred)
print("\n#### Performance ####")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Выберем тестовую точку данных и спрогнозируем для нее результат.

```
# Тестирование регрессора на тестовой точке данных
test_data = [3.7, 0, 18.4, 1, 0.87, 5.95, 91, 2.5052, 26, 666,
             20.2, 351.34, 15.27]
print("\nPredicted price:", sv_regressor.predict([test_data])[0])
```

После выполнения этого кода в окне терминала отобразится следующий вывод.

```
#### Performance ####
Mean squared error = 15.41
```

```
Explained variance score = 0.82
Predicted price: 18.5217801073
```

Код для этого раздела содержится в файле `house_prices.py`.

Резюме

Из этой главы вы узнали о различиях между обучением с учителем и без учителя. Мы обсудили задачу классификации данных и способы ее решения, а также рассмотрели предварительную обработку данных с использованием различных методов. Вы узнали о том, что такое кодирование меток и как создавать предназначенные для этого кодировщики. Мы изучили логистическую регрессию и создали классификатор на ее основе. Также были рассмотрены наивный байесовский классификатор и способы его создания. Кроме того, было показано, как создается матрица неточностей.

Мы также обсудили машины опорных векторов и способы создания классификаторов на их основе. Вы познакомились с регрессией и узнали о том, как использовать линейную и полиномиальную регрессии применительно к одномерным и многомерным данным. После этого мы использовали регрессор на основе машины опорных векторов для оценки стоимости жилья с использованием входных атрибутов.

В следующей главе мы поговорим о предсказательной аналитике и создании предсказательного механизма на основе ансамблевого обучения.

3

Предсказательная аналитика на основе ансамблевого обучения

В этой главе мы поговорим о методах ансамблевого обучения и их использовании в предсказательной аналитике. Вы ознакомитесь со следующими темами.

- создание моделей обучения с помощью ансамблевого обучения;
- что такое деревья решений и как создавать соответствующие классификаторы;
- что такое случайные леса и предельно случайные леса и как создавать классификаторы на их основе;
- оценки достоверности предсказанных значений;
- обработка дисбаланса классов;
- нахождение оптимальных обучающих параметров с помощью сеточного поиска;
- вычисление относительной важности признаков;
- предсказание интенсивности дорожного движения с использованием регрессора на основе предельно случайных лесов.

Что такое ансамблевое обучение

Термин **ансамблевое обучение** (ensamble learning) относится к процессу построения множества моделей и поиску такой их комбинации, которая позволяет получить лучшие результаты, чем каждая из моделей по отдельности. В качестве индивидуальных моделей могут выступать классификаторы, регрессоры и другие объекты, моделирующие данные тем или иным способом. Ансамблевое обучение широко применяется во многих областях,

включая классификацию данных, предсказательное моделирование, обнаружение аномалий и т.п.

Почему мы в первую очередь рассматриваем ансамблевое обучение? Чтобы это понять, обратимся к реальному примеру. Предположим, вы хотите купить новый телевизор, но о последних моделях вам ничего не известно. Ваша задача — купить наилучший телевизор из тех, которые предлагаются по доступной для вас цене, но вы недостаточно хорошо знаете рынок, чтобы сделать обоснованный выбор. В подобных случаях вы интересуетесь мнением нескольких экспертов в данной области. Так вам легче принять наиболее верное решение. В большинстве случаев вы не будете привязываться к мнению какого-то одного специалиста и примете окончательное решение на основе обобщения оценок, сделанных разными людьми. Мы поступаем так, потому что стремимся свести к минимуму вероятность принятия неверных или недостаточно оптимальных решений.

Построение моделей обучения посредством ансамблевого метода

При выборе модели чаще всего исходят из того, чтобы она приводила к наименьшим ошибкам на тренировочном наборе данных. Проблема заключается в том, что такой подход не всегда работает в силу возможного эффекта переобучения. Даже если перекрестная проверка модели подтверждает ее адекватность, она может приводить к неудовлетворительным результатам для неизвестных данных.

Одной из основных причин эффективности ансамблевого обучения является то, что этот метод позволяет снизить общий риск выбора неудачной модели. Благодаря тому что тренировка осуществляется на широком разнообразии обучающих наборов данных, ансамблевый подход позволяет получать неплохие результаты для неизвестных данных. Если мы создаем модель на основе ансамблевого обучения, то результаты, полученные с использованием индивидуальных моделей, должны проявлять определенный разброс. Это позволяет улавливать всевозможные нюансы, в результате чего обобщенная модель оказывается более точной.

Отмеченное разнообразие результатов достигается за счет использования разных обучающих параметров для индивидуальных моделей, благодаря чему они генерируют различные границы решений для тренировочных данных. Это означает, что каждая модель будет использовать разные правила для логического вывода, тем самым обеспечивая более эффективный способ валидации окончательного результата. Если между моделями наблюдается согласие, то мы знаем, что результат является корректным.

Что такое деревья принятия решений

Дерево принятия решений (decision tree) — это структура, позволяющая разбить набор данных на ветви с последующим принятием простых решений на каждом уровне. Окончательное решение получают, спускаясь по ветвям этого дерева. Деревья принятия решений создаются посредством обучающих алгоритмов, определяющих наилучшие способы разбиения данных.

Любой процесс принятия решений начинается в корневом узле, располагающемся в вершине дерева. Каждый узел, по сути, представляет собой правило принятия решения. Алгоритмы конструируют эти правила, исходя из соотношений между входными данными и целевыми метками в тренировочном наборе. Значения входных данных используются для оценки выходных значений.

Теперь, когда базовая концепция деревьев принятия решений вам понятна, можем перейти к рассмотрению методов автоматического построения таких деревьев. Нам нужны алгоритмы, обеспечивающие построение оптимального дерева на основе наших данных. Но чтобы понимать, как работают эти алгоритмы, нужно знать, что такое энтропия. В данном контексте термин *энтропия* относится не к термодинамической, а к информационной энтропии. В сущности, энтропия — это мера неопределенности. Одним из основных назначений дерева решений является уменьшение степени неопределенности по мере продвижения от корневого узла к листьям. Столкнувшись с неизвестной точкой данных, мы совершенно ничего не можем сказать о конечном результате. В тот момент, когда мы достигаем листового узла, результат становится достоверно известным. Отсюда следует, что дерево принятия решений необходимо конструировать так, чтобы каждый очередной шаг приводил к уменьшению неопределенности. А это означает, что по мере продвижения вниз по дереву узлов мы должны уменьшать энтропию.

Для получения более подробной информации по этому вопросу прочитайте следующую статью:

`https://prateekvjoshi.com/2016/03/22/how-are-decision-trees-constructed-in-machine-learning`

Создание классификатора на основе дерева принятия решений

Рассмотрим процесс создания классификатора на основе дерева принятия решений, используя Python. Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation
from sklearn.tree import DecisionTreeClassifier

from utilities import visualize_classifier
```

Мы будем использовать данные, которые содержатся в файле `data_decision_trees.txt`. В этом файле каждая строка содержит значения, разделенные запятой. Первые два значения соответствуют входным данным, последнее — целевым меткам. Загрузим данные из этого файла.

```
# Загрузка входных данных
input_file = 'data_decision_trees.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Разделим входные данные на два отдельных класса в зависимости от целевых меток.

```
# Разделение входных данных на два класса на основании меток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```

Визуализируем входные данные с помощью точечной диаграммы.

```
# Визуализация входных данных
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
            facecolors='black', edgecolors='black',
            linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='o')
plt.title('Входные данные')
```

Мы должны разбить данные на обучающий и тестовый наборы.

```
# Разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
cross_validation.train_test_split(X, y, test_size=0.25, random_state=5)
```

Создадим и визуализируем классификатор, используя обучающий набор данных. Параметр `random_state` определяет затравочное значение, используемое генератором случайных чисел или необходимое для инициализации алгоритма классификации на основе дерева принятия решений. Параметр `max_depth` определяет максимальную глубину дерева, которое мы хотим построить.

```
# Классификатор на основе дерева принятия решений
params = {'random_state': 0, 'max_depth': 4}
classifier = DecisionTreeClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
```

Вычислим выходной результат классификатора, полученный на тестовом наборе данных, и визуализируем его.

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Тестовый набор данных')
```

Оценим работу классификатора, выведя отчет с результатами классификации.

```
# Оценка работы классификатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#"*40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
      classifier.predict(X_train), target_names=class_names))
print("#"*40 + "\n")
print("#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
      target_names=class_names))
print("#"*40 + "\n")

plt.show()
```

Полный код этого примера содержится в файле `decision_trees.py`. В процессе его выполнения на экране отобразится ряд графиков. Первый экранный снимок — это визуализация входных данных (рис. 3.1).

Второй экранный снимок представляет границы классификатора на тестовом наборе данных (рис. 3.2).

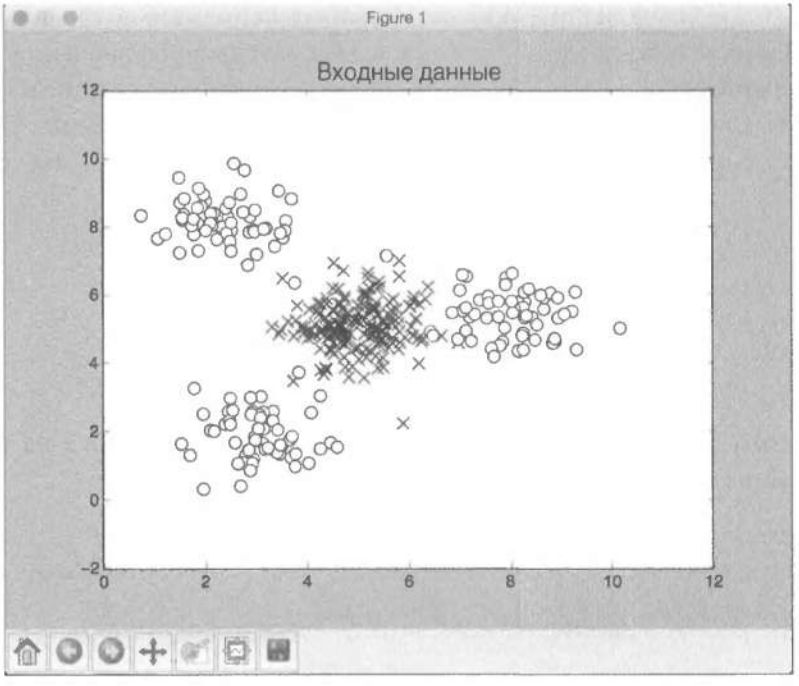

Рис. 3.1

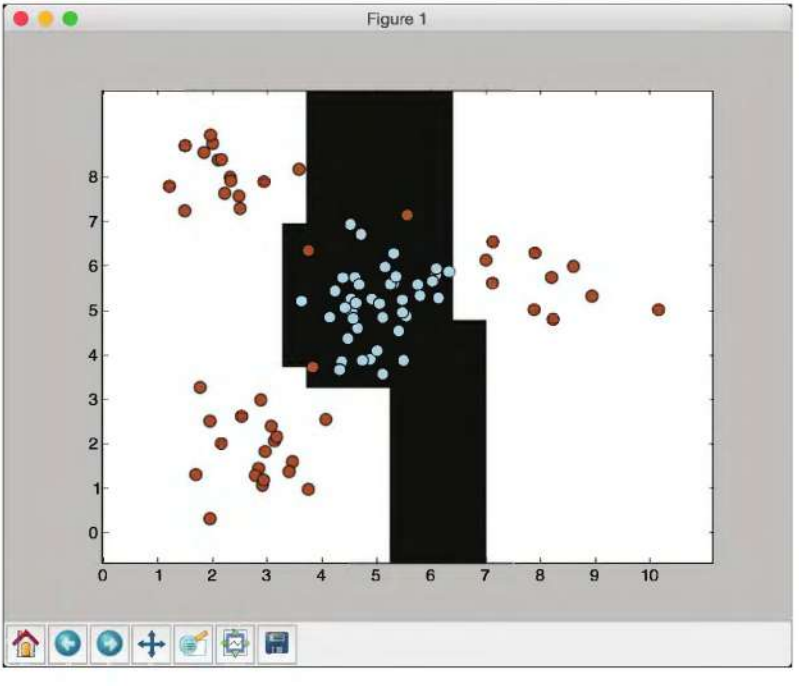

Рис. 3.2. (См. цветную вклейку; адрес указан во введении)

В окне терминала будет выведена следующая информация (рис. 3.3).

```
########################################

Classifier performance on training dataset

             precision    recall  f1-score   support

    Class-0       0.99      1.00      1.00       137
    Class-1       1.00      0.99      1.00       133

avg / total       1.00      1.00      1.00       270

########################################

########################################

Classifier performance on test dataset

             precision    recall  f1-score   support

    Class-0       0.93      1.00      0.97        43
    Class-1       1.00      0.94      0.97        47

avg / total       0.97      0.97      0.97        90

########################################
```

Рис. 3.3

Работа классификатора характеризуется параметрами `precision` (точность), `recall` (полнота) и `f1-score` (F-мера). Показатель точности — это точность классификации, показатель полноты — процентная доля количества извлеченных элементов по отношению к общему количеству элементов, которые должны были быть извлечены в соответствии с ожиданиями. Хороший классификатор обеспечивает высокие значения точности и полноты, но обычно достигается лишь некий компромиссный вариант. В связи с этим удобно пользоваться F-мерой. *F-мера* — это гармоническое среднее показателей точности и полноты, представляющее разумную сбалансированную оценку работы классификатора.

Случайные и предельно случайные леса

Случайный лес (random forest) — частный случай ансамблевого обучения, в котором индивидуальные модели конструируются с использованием деревьев решений. Полученный ансамбль используется далее для прогнозирования результата. При конструировании отдельных деревьев используют

случайные подмножества тренировочных данных. Это гарантирует разброс данных между различными деревьями решений. Как отмечалось выше, в ансамблевом обучении очень важно обеспечить разнородность ансамбля индивидуальных моделей.

Одним из наибольших достоинств случайных лесов является то, что они не переобучаются. Как вы уже знаете, в машинном обучении эта проблема встречается довольно часто. Конструируя неоднородное множество деревьев решений за счет использования различных случайных подмножеств, мы гарантируем отсутствие переобучения модели на тренировочных данных. В процессе конструирования дерева решений его узлы последовательно расщепляются, и для них выбираются наилучшие пороговые значения, снижающие энтропию на каждом уровне. В процессе расщепления узлов учитываются не все признаки, характеризующие данные входного набора. Вместо этого выбирается наилучший способ расщепления узлов, основанный на текущем случайном поднаборе рассматриваемых признаков. Включение фактора случайности увеличивает смещение случайного леса, однако дисперсия уменьшается благодаря усреднению. Это обусловливает робастность результирующей модели.

Предельно случайные леса (extremly random forests) еще более усиливают роль фактора случайности. Наряду со случайным выбором признаков случайно выбираются также пороговые значения. Эти случайно генерируемые значения становятся правилами разбиения, дополнительно уменьшающими вариативность модели. Поэтому использование предельно случайных лесов обычно приводит к более гладким границам принятия решений по сравнению с теми, которые удается получить с помощью случайных лесов.

Создание классификаторов на основе случайных и предельно случайных лесов

Перейдем к рассмотрению конкретных примеров создания классификаторов на основе случайных и предельно случайных лесов. Способы создания классификаторов обоих типов весьма схожи, поэтому для указания того, какой именно классификатор создается, мы будем использовать входной флаг.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation
from sklearn.ensemble import RandomForestClassifier,
```

```
ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report

from utilities import visualize_classifier
```

Определим синтаксический анализатор (парсер) аргументов для Python, чтобы можно было принимать тип классификатора в качестве входного параметра. Задавая соответствующее значение этого параметра, мы сможем выбирать тип создаваемого классификатора.

```
# Парсер аргументов
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify \
            data using Ensemble Learning techniques')
    parser.add_argument('--classifier-type',
            dest='classifier_type', required=True,
            choices=['rf', 'erf'], help="Type of \
            classifier to use; can be either 'rf' or \
            'erf'")
    return parser
```

Определим основную функцию и извлечем входные аргументы.

```
if __name__=='__main__':
    # Извлечение входных аргументов
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
```

Мы будем использовать данные из файла `data_random_forests.txt`, который вам предоставляется. В этом файле каждая строка содержит значения, разделенные запятой. Первые два значения соответствуют входным данным, последнее — целевой метке. В этом наборе данных содержатся три различных класса. Загрузим данные из этого файла.

```
    # Загрузка входных данных
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]
```

Разобьем входные данные на три класса.

```
    # Разбиение входных данных на три класса
    # на основании меток
    class_0 = np.array(X[y==0])
    class_1 = np.array(X[y==1])
    class_2 = np.array(X[y==2])
```

Визуализируем входные данные.

```
# Визуализация входных данных
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
      facecolors='white', edgecolors='black',
      linewidth=1, marker='s')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
      facecolors='white', edgecolors='black',
      linewidth=1, marker='o')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75,
      facecolors='white', edgecolors='black',
      linewidth=1, marker='^')
plt.title('Входные данные')
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбивка данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
      cross_validation.train_test_split(
      X, y, test_size=0.25, random_state=5)
```

Определим параметры, которые будем использовать при конструировании классификатора. Параметр `n_estimator` — это количество создаваемых деревьев. Параметр `max_depth` — это максимальное количество уровней в каждом дереве. Параметр `random_state` — это затравочное значение для генератора случайных чисел, необходимое для инициализации алгоритма классификатора на основе случайного леса.

```
# Классификатор на основе ансамблевого обучения
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
```

В зависимости от того, какое значение входного параметра мы предоставили, конструируется классификатор на основе случайного или предельно случайного леса.

```
if classifier_type == 'rf':
   classifier = RandomForestClassifier(**params)
else:
   classifier = ExtraTreesClassifier(**params)
```

Обучим и визуализируем классификатор.

```
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train,
                     'Training dataset')
```

Вычислим результат на тестовом наборе данных и визуализируем его.

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test,
                     'Тестовый набор данных')
```

Проверим, как работает классификатор, выведя отчет с результатами классификации.

```
# Проверка работы классификатора
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#"*40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
      classifier.predict(X_train), target_names=class_names))
print("#"*40 + "\n")

print("#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
      target_names=class_names))
print("#"*40 + "\n")
```

Полный код этого примера содержится в файле `random_forests.py`. Выполните данный код, запросив создание классификатора на основе случайного леса с помощью флага `rf` входного аргумента. Введите в окне терминала следующую команду:

```
$ python3 random_forests.py --classifier-type rf
```

В процессе выполнения этого кода вы получите ряд графиков. На первом экранном снимке представлены входные данные (рис. 3.4).

На предыдущем экранном снимке квадраты, окружности и треугольники представляют три класса. Как видим, классы в значительной степени перекрываются, однако на данном этапе это нормально. На втором экранном снимке отображены границы классификатора (рис. 3.5).

А теперь выполните тот же код, запросив создание классификатора на основе предельно случайного леса с помощью флага `erf` входного аргумента. Введите в окне терминала следующую команду:

```
$ python3 random_forests.py --classifier-type erf
```

Первый из выведенных графиков будет представлять те же входные данные, которые вы уже видели. На втором экранном снимке отображены границы классификатора (рис. 3.6).

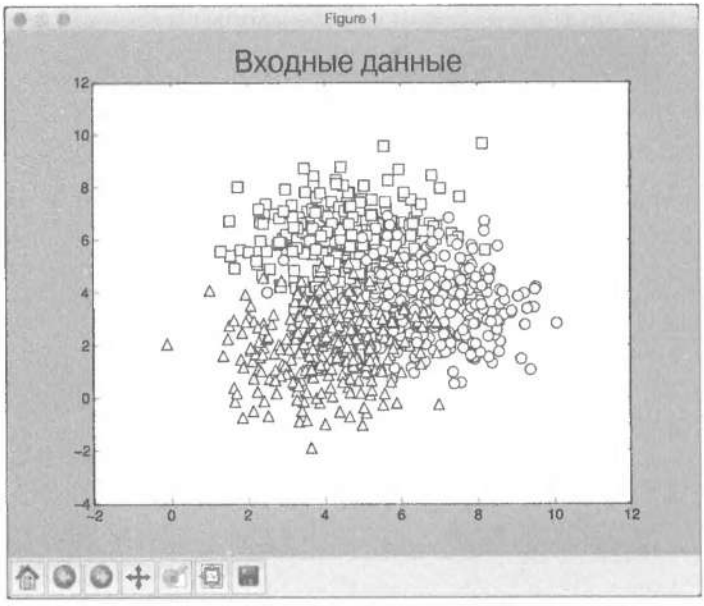

Рис. 3.4

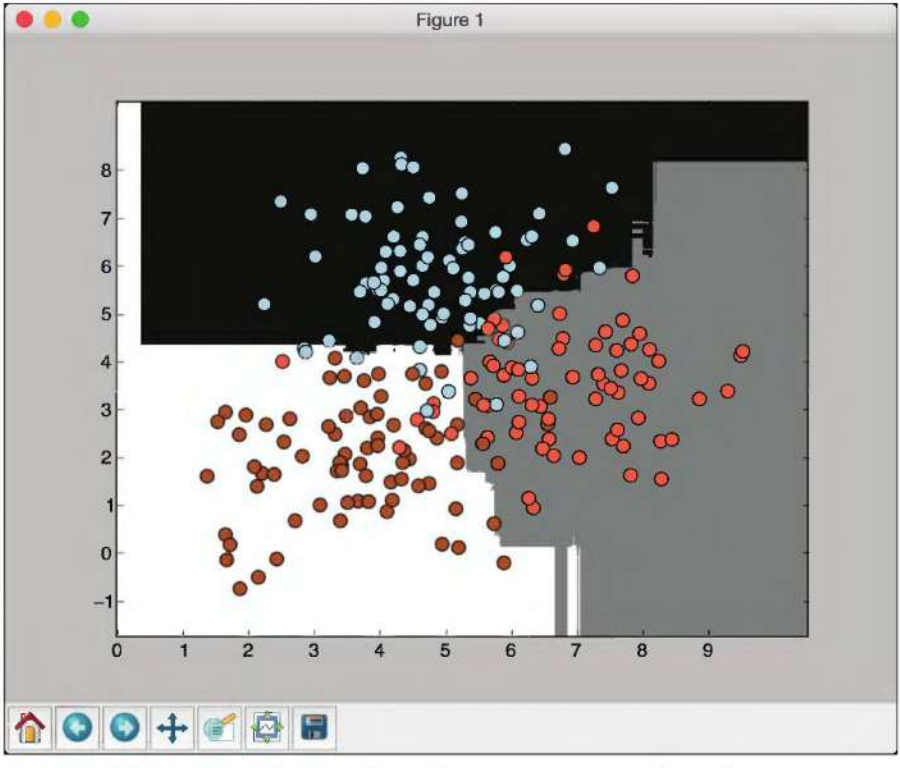

Рис. 3.5. (См. цветную вклейку; адрес указан во введении)

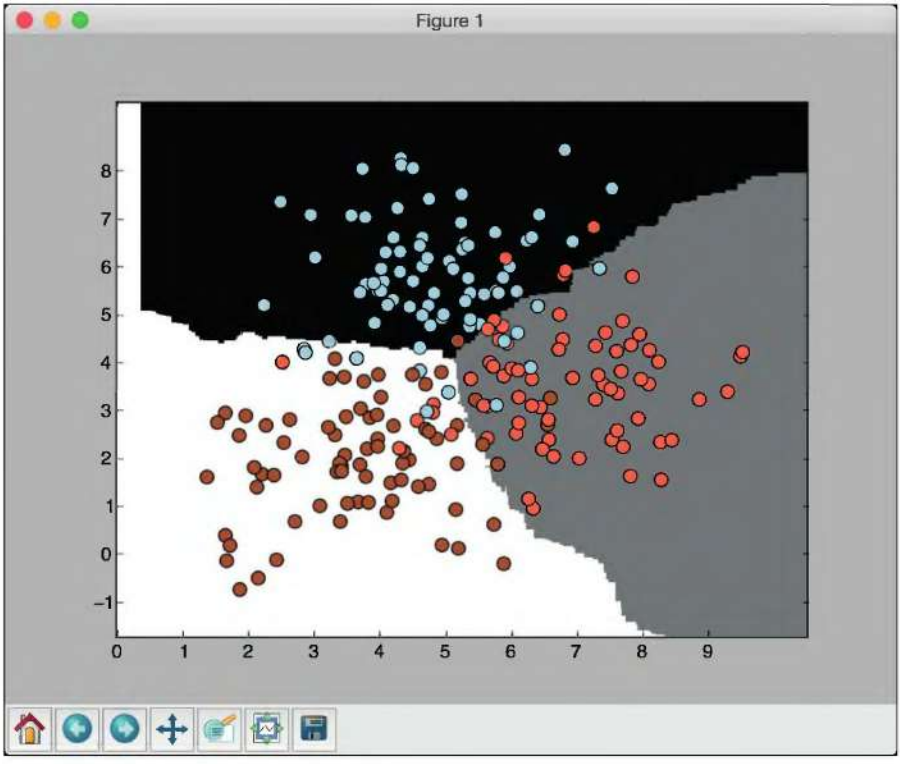

Рис. 3.6. (См. цветную вклейку; адрес указан во введении)

Если сравнить рис. 3.5 и 3.6, то становится ясно, что в последнем случае были получены более гладкие границы. Это обусловлено тем, что в процессе обучения предельно случайные леса имеют больше возможностей для выбора оптимальных деревьев решений, поэтому, как правило, они обеспечивают получение лучших границ.

Оценка мер достоверности прогнозов

Если вы посмотрите на результаты, отображаемые в окне терминала, то увидите, что для каждой точки данных выводятся вероятности. Этими вероятностями измеряются *уровни доверительности* (уровни доверия) для каждого класса. Оценка уровней доверия играет важную роль в машинном обучении. Добавьте в тот же файл следующую строку, определяющую массив тестовых точек данных.

```
    # Вычисление параметров доверительности
    test_datapoints = np.array([[5, 5], [3, 6], [6, 4],
                                [7, 2], [4, 4], [5, 2]])
```

Объект классификатора имеет встроенный метод, предназначенный для вычисления уровней доверительности. Классифицируем каждую точку и вычислим уровни доверительности.

```
    print("\nConfidence measure:")
    for datapoint in test_datapoints:
        probabilities =
classifier.predict_proba([datapoint])[0]
        predicted_class = 'Class-' +
str(np.argmax(probabilities))
        print('\nDatapoint:', datapoint)
        print('Predicted class:', predicted_class)
```

Визуализируем тестовые точки данных на основании границ классификатора.

```
    # Визуализация точек данных
    visualize_classifier(classifier, test_datapoints,
                         [0]*len(test_datapoints),
                         'Тестовые точки данных')
    plt.show()
```

Результат выполнения этого кода с флагом `rf` представлен на рис. 3.7.

В окне терминала отобразится следующий вывод (рис. 3.8).

Для каждой точки данных вычисляется вероятность ее принадлежности каждому из трех классов. Мы выбираем тот класс, которому соответствует самый высокий уровень доверия. Результат выполнения этого кода с флагом `erf` представлен на рис. 3.9.

В окне терминала отобразится следующий вывод (рис. 3.10).

Как видите, эти результаты согласуются с нашими наблюдениями.

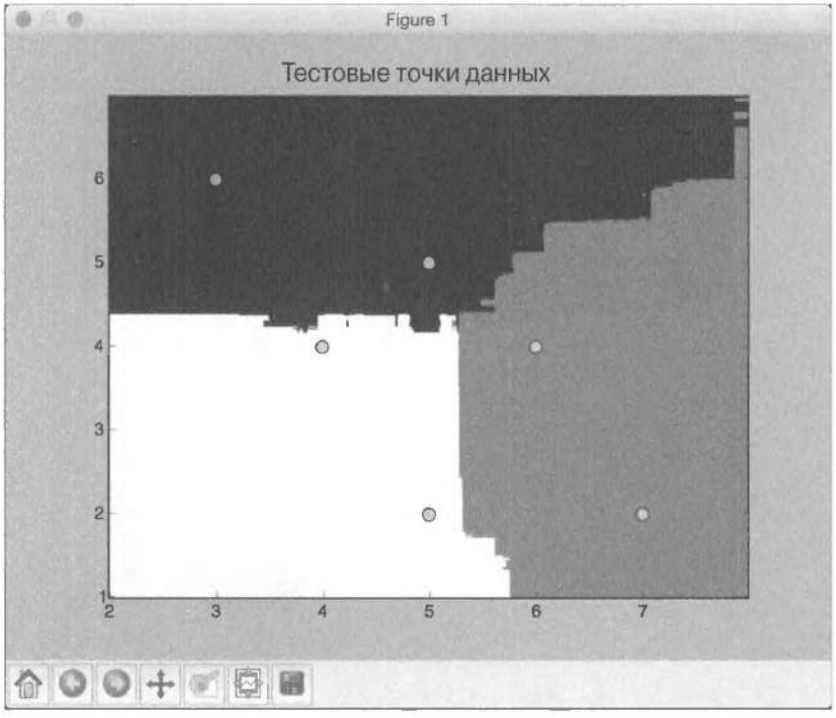

Рис. 3.7

```
Datapoint: [5 5]
Probabilities: [ 0.81427532  0.08639273  0.09933195]
Predicted class: Class-0

Datapoint: [3 6]
Probabilities: [ 0.93574458  0.02465345  0.03960197]
Predicted class: Class-0

Datapoint: [6 4]
Probabilities: [ 0.12232404  0.7451078   0.13256816]
Predicted class: Class-1

Datapoint: [7 2]
Probabilities: [ 0.05415465  0.70660226  0.23924309]
Predicted class: Class-1

Datapoint: [4 4]
Probabilities: [ 0.20594744  0.15523491  0.63881765]
Predicted class: Class-2

Datapoint: [5 2]
Probabilities: [ 0.05403583  0.0931115   0.85285267]
Predicted class: Class-2
```

Рис. 3.8

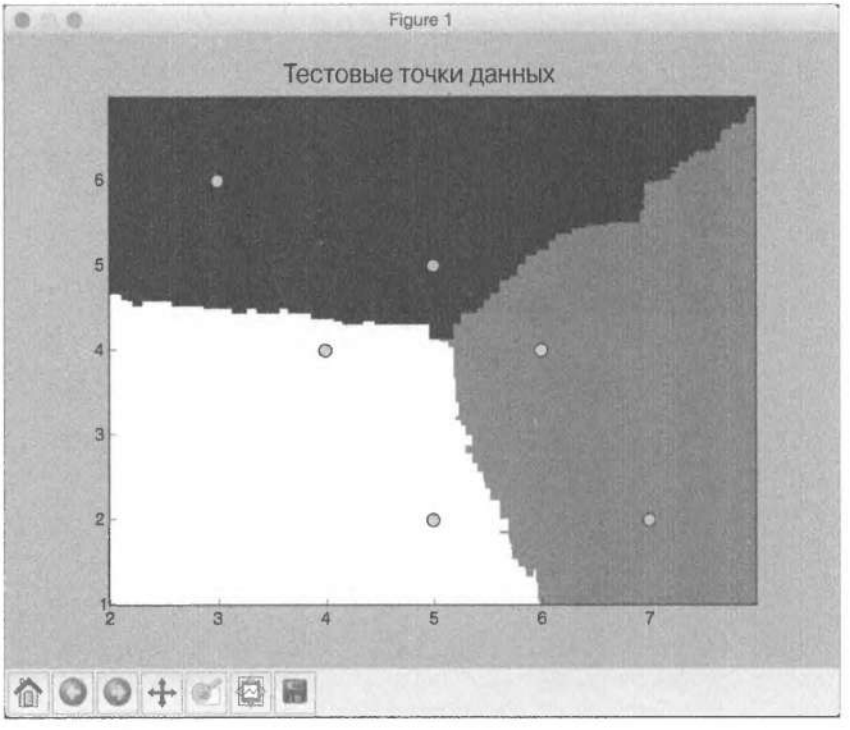

Рис. 3.9

```
Datapoint: [5 5]
Probabilities: [ 0.48904419  0.28020114  0.23075467]
Predicted class: Class-0

Datapoint: [3 6]
Probabilities: [ 0.66707383  0.12424406  0.20868211]
Predicted class: Class-0

Datapoint: [6 4]
Probabilities: [ 0.25788769  0.49535144  0.24676087]
Predicted class: Class-1

Datapoint: [7 2]
Probabilities: [ 0.10794013  0.6246677   0.26739217]
Predicted class: Class-1

Datapoint: [4 4]
Probabilities: [ 0.33383778  0.21495182  0.45121039]
Predicted class: Class-2

Datapoint: [5 2]
Probabilities: [ 0.18671115  0.28760896  0.52567989]
Predicted class: Class-2
```

Рис. 3.10

Обработка дисбаланса классов

Качество классификатора зависит от данных, используемых для его обучения. Одной из наиболее распространенных проблем, с которыми приходится сталкиваться в реальных задачах, является качество данных. Чтобы классификатор работал надежно, ему необходимо предоставить равное количество точек данных для каждого класса. Однако в реальных условиях гарантировать соблюдение этого условия не всегда возможно. Если количество точек данных для одного класса в 10 раз больше, чем для другого, то классификатор будет отдавать предпочтение первому классу. Следовательно, подобный дисбаланс необходимо учесть алгоритмически. Рассмотрим, как это делается.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import sys

import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report

from utilities import visualize_classifier
```

Используем для нашего анализа данные, которые содержатся в файле `data_imbalance.txt`. В этом файле каждая строка содержит значения, разделенные запятой. Первые два значения соответствуют данным, последнее — целевой метке. В этом наборе данных имеются два класса. Загрузим данные из этого файла.

```
# Загрузка входных данных
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Разобьем входные данные на два класса.

```
# Разделение входных данных на два класса на основании меток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```

Визуализируем входные данные, используя точечную диаграмму.

```
# Визуализация входных данных
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
```

```
                    facecolors='black', edgecolors='black',
                    linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
                    facecolors='white', edgecolors='black',
                    linewidth=1, marker='o')
plt.title('Входные данные')
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
            cross_validation.train_test_split(
            X, y, test_size=0.25, random_state=5)
```

Определим параметры для классификатора на основе предельно случайных лесов. Обратите внимание на входной параметр balance, который управляет тем, будет ли учитываться алгоритмически дисбаланс классов. В случае учета этого фактора мы должны добавить еще один параметр, class_weight, балансирующий веса таким образом, чтобы они были пропорциональны количеству точек данных в каждом классе.

```
# Классификатор на основе предельно случайных лесов
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
if len(sys.argv) > 1:
      if sys.argv[1] == 'balance':
            params = {'n_estimators': 100, 'max_depth': 4,
                      'random_state': 0, 'class_weight': 'balanced'}
      else:
            raise TypeError("Invalid input argument; should be
                            'balance'")
```

Создадим, обучим и визуализируем классификатор, используя тренировочные данные.

```
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
```

Предскажем и визуализируем результат для тестового набора данных.

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Тестовый набор данных')
```

Вычислим показатели эффективности работы классификатора и выведем отчет о результатах классификации.

```
# Вычисление показателей эффективности классификатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#"*40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
      classifier.predict(X_train), target_names=class_names))
print("#"*40 + "\n")

print("#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
      target_names=class_names))
print("#"*40 + "\n")

plt.show()
```

Полный код этого примера содержится в файле `class_imbalance.py`. В процессе выполнения этого кода вы получите ряд графиков. На рис. 3.11 представлены входные данные.

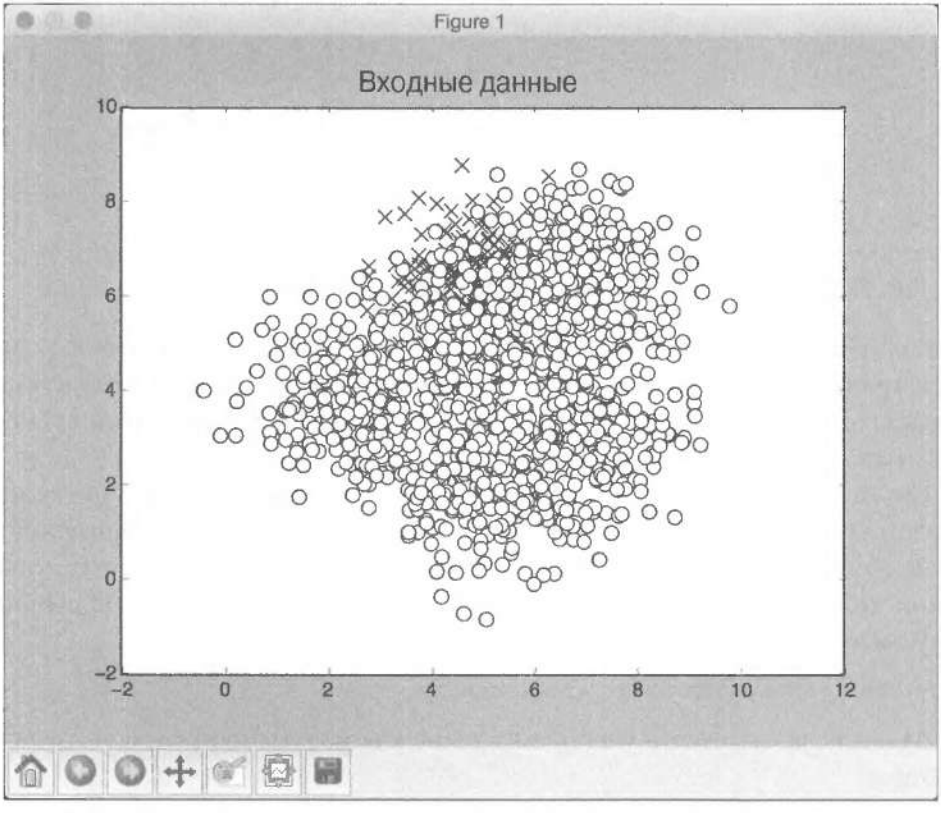

Рис. 3.11

На втором экранном снимке отображены границы классификатора для тестового набора данных (рис. 3.12).

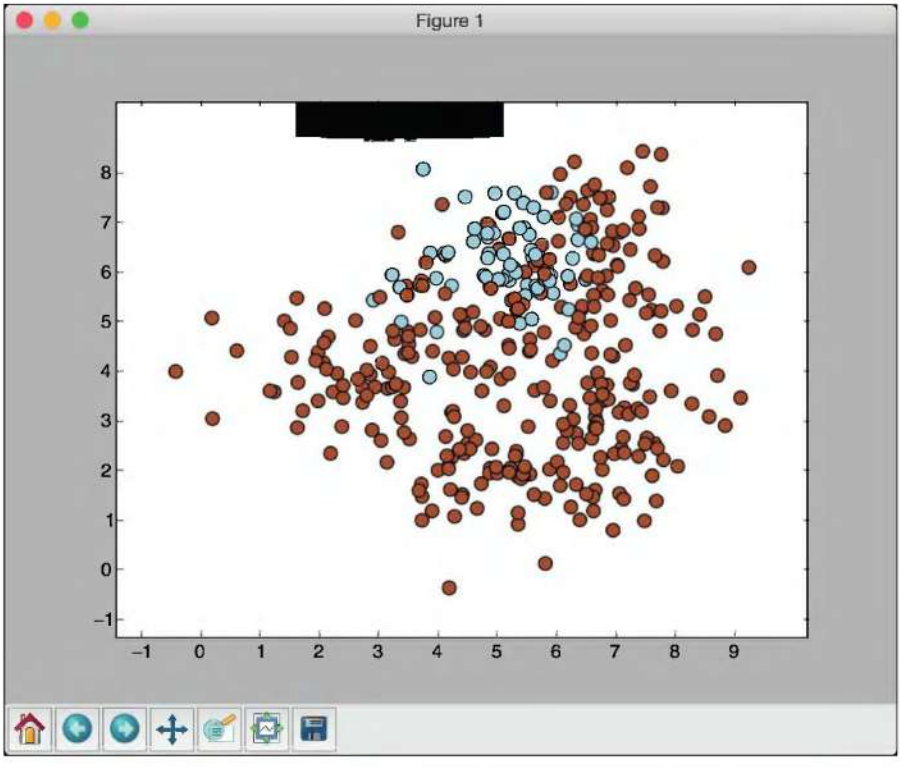

Рис. 3.12. (См. цветную вклейку; адрес указан во введении)

Как нетрудно заметить, классификатору не удалось определить фактическую границу между двумя классами. В данном случае вычисленную границу представляет черное пятно в верхней части рисунка. В окне терминала отобразится следующая информация (рис. 3.13).

Также будет выведено предупреждение о наличии нулевых значений в первой строке с числовыми данными, что приводит к возникновению ошибки деления на нуль (исключение `ZeroDivisionError`) при попытке вычисления показателя `f1-score`. Чтобы это предупреждение не появлялось, запустите код в окне терминала с флагом `ignore`.

```
$ python3 --W ignore class_imbalance.py
```

Далее, если вы хотите учесть дисбаланс классов, выполните код с флагом `balance`.

```
$ python3 class_imbalance.py balance
```

```
########################################

Classifier performance on test dataset

             precision    recall  f1-score   support

    Class-0       0.00      0.00      0.00        69
    Class-1       0.82      1.00      0.90       306

avg / total       0.67      0.82      0.73       375

########################################
```

Рис. 3.13

Теперь результаты работы классификатора должны выглядеть так (рис. 3.14).

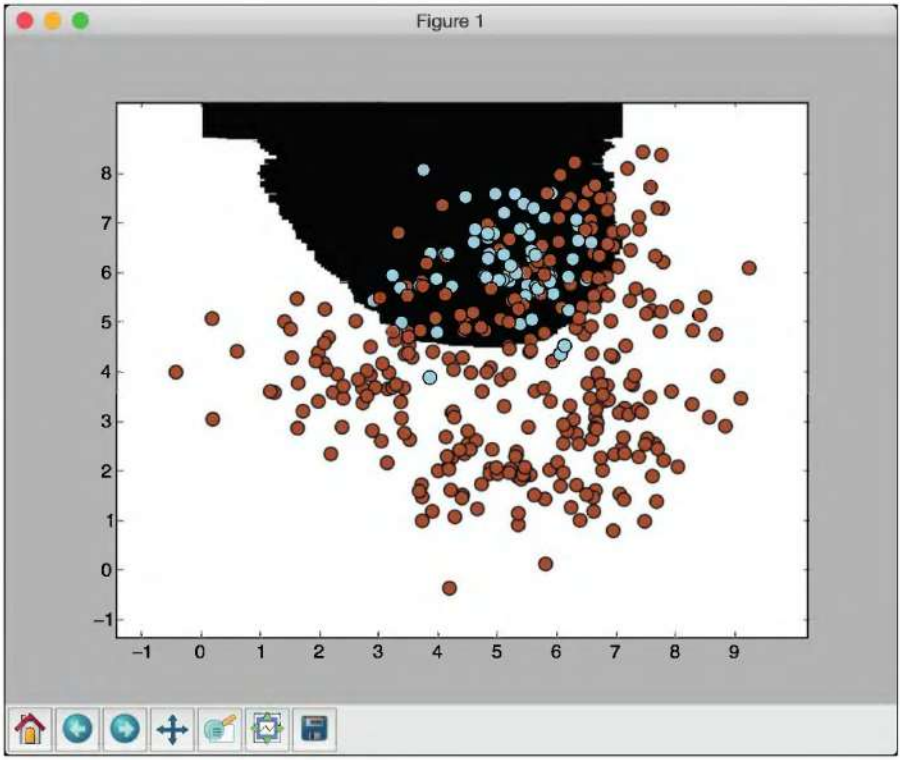

Рис. 3.14. (См. цветную вклейку; адрес указан во введении)

В окне терминала отобразится следующая информация (рис. 3.15).

```
#######################################

Classifier performance on test dataset

             precision    recall  f1-score   support

    Class-0       0.45      0.94      0.61        69
    Class-1       0.98      0.74      0.84       306

avg / total       0.88      0.78      0.80       375

#######################################
```

Рис. 3.15

Благодаря учету дисбаланса классов нам удалось классифицировать точки данных для класса `0` с ненулевым значением параметра точности.

Нахождение оптимальных обучающих параметров с помощью сеточного поиска

В процессе работы с классификаторами вам не всегда известно, какие параметры являются наилучшими. Их подбор вручную методом грубой силы (путем перебора всех возможных комбинаций) практически нереализуем. И здесь на помощь приходит *сеточный поиск* (grid search). Рассмотрим, как это делается.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation, grid_search
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report

from utilities import visualize_classifier
```

Используем для нашего анализа данные, которые содержатся в файле `data_random_forests.txt`.

```
# Загрузка входных данных
input_file = 'data_random_forests.txt'
```

```
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Разобьем данные на три класса.

```
# Разбиение данных на три класса на основании меток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
class_2 = np.array(X[y==2])
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
            cross_validation.train_test_split(
            X, y, test_size=0.25, random_state=5)
```

Зададим сетку значений параметров, на которой будем тестировать классификатор. Обычно мы поддерживаем постоянным значение одного параметра и варьируем остальные. Затем эта процедура повторяется для каждого из параметров. В данном случае мы хотим найти наилучшие значения для параметров `n_estimators` и `max_depth`. Определим сетку значений параметров.

```
# Определение сетки значений параметров
parameter_grid = [ {'n_estimators': [100],
          'max_depth': [2, 4, 7, 12, 16]},
          {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}
                ]
```

Определим метрические характеристики, которые должен использовать классификатор для нахождения наилучшей комбинации параметров.

```
metrics = ['precision_weighted', 'recall_weighted']
```

Для каждой метрики необходимо выполнить сеточный поиск, в ходе которого мы будем обучать классификатор конкретной комбинации параметров.

```
for metric in metrics:
      print("\n##### Searching optimal parameters for", metric)

      classifier = grid_search.GridSearchCV(
                  ExtraTreesClassifier(random_state=0),
                  parameter_grid, cv=5, scoring=metric)
      classifier.fit(X_train, y_train)
```

Выведем оценку для каждой комбинации параметров.

```
print("\nGrid scores for the parameter grid:")
for params, avg_score, _ in classifier.grid_scores_:
    print(params, '-->', round(avg_score, 3))
print("\nBest parameters:", classifier.best_params_)
```

Выведем отчет с результатами работы классификатора.

```
y_pred = classifier.predict(X_test)
print("\nPerformance report:\n")
print(classification_report(y_test, y_pred))
```

Полный код этого примера содержится в файле `run_grid_search.py`. После выполнения этого кода в окне терминала отобразится следующая информация (рис. 3.16).

```
##### Searching optimal parameters for precision_weighted

Grid scores for the parameter grid:
{'n_estimators': 100, 'max_depth': 2} --> 0.847
{'n_estimators': 100, 'max_depth': 4} --> 0.841
{'n_estimators': 100, 'max_depth': 7} --> 0.844
{'n_estimators': 100, 'max_depth': 12} --> 0.836
{'n_estimators': 100, 'max_depth': 16} --> 0.818
{'n_estimators': 25, 'max_depth': 4} --> 0.846
{'n_estimators': 50, 'max_depth': 4} --> 0.84
{'n_estimators': 100, 'max_depth': 4} --> 0.841
{'n_estimators': 250, 'max_depth': 4} --> 0.845

Best parameters: {'n_estimators': 100, 'max_depth': 2}

Performance report:

             precision    recall  f1-score   support

        0.0       0.94      0.81      0.87        79
        1.0       0.81      0.86      0.83        70
        2.0       0.83      0.91      0.87        76

avg / total       0.86      0.86      0.86       225
```

Рис. 3.16

Исходя из комбинаций значений параметров, использованных в сеточном поиске, здесь выведены результаты, соответствующие наиболее оптимальной комбинации для показателя точности (`precision`). Результаты, соответствующие наилучшей комбинации для показателя полноты классификации (`recall`), приведены на рис. 3.17.

```
##### Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'n_estimators': 100, 'max_depth': 2} --> 0.84
{'n_estimators': 100, 'max_depth': 4} --> 0.837
{'n_estimators': 100, 'max_depth': 7} --> 0.841
{'n_estimators': 100, 'max_depth': 12} --> 0.834
{'n_estimators': 100, 'max_depth': 16} --> 0.816
{'n_estimators': 25, 'max_depth': 4} --> 0.843
{'n_estimators': 50, 'max_depth': 4} --> 0.836
{'n_estimators': 100, 'max_depth': 4} --> 0.837
{'n_estimators': 250, 'max_depth': 4} --> 0.841

Best parameters: {'n_estimators': 25, 'max_depth': 4}

Performance report:

             precision    recall  f1-score   support

        0.0       0.93      0.84      0.88        79
        1.0       0.85      0.86      0.85        70
        2.0       0.84      0.92      0.88        76

avg / total       0.87      0.87      0.87       225
```

Рис. 3.17

Эта другая комбинация значений параметров, обеспечивающая получение наилучшего значения показателя `recall`, отличается от первой, что вполне объяснимо, поскольку `precision` и `recall` — разные метрические характеристики, требующие использования разных комбинаций параметров.

Вычисление относительной важности признаков

Когда мы работаем с наборами данных, содержащими N-мерные точки данных, необходимо понимать, что не все признаки одинаково важны. Одни из них играют более важную роль, чем другие. Располагая этой информацией, можно уменьшить количество учитываемых размерностей. Мы можем использовать эту возможность для снижения сложности алгоритма и его ускорения. Иногда некоторые признаки оказываются совершенно излишними. Следовательно, их можно безболезненно исключить из набора данных.

Для вычисления важности признаков мы будем использовать регрессор `AdaBoost`. Сокращение "AdaBoost" происходит от названия алгоритма *Adaptive Boosting* (адаптивная поддержка), который часто применяется в

сочетании с другими алгоритмами машинного обучения для повышения их эффективности. AdaBoost извлекает обучающие точки данных для тренировки текущего классификатора, используя некоторое распределение их весов. Это распределение итеративно обновляется, поэтому последующие классификаторы фокусируются на более трудных точках. (Трудные точки — это точки, которые были классифицированы неверно.) Благодаря этому точки данных, которые ранее были неверно классифицированы, получают большие веса в выборочном наборе данных, используемом для обучения классификаторов. Алгоритм объединяет эти классификаторы в "комитет", который принимает окончательное решение на основании взвешенного большинства голосов.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
from sklearn.metrics import mean_squared_error,
    explained_variance_score
from sklearn import cross_validation
from sklearn.utils import shuffle
```

Мы будем использовать встроенный набор данных с ценами на недвижимость, доступный в библиотеке scikit-learn.

```
# Загрузка данных с ценами на недвижимость
housing_data = datasets.load_boston()
```

Перемешаем данные, чтобы повысить объективность нашего анализа.

```
# Перемешивание данных
X, y = shuffle(housing_data.data, housing_data.target,
               random_state=7)
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
cross_validation.train_test_split(
            X, y, test_size=0.2, random_state=7)
```

Определим и обучим регрессор AdaBoost, используя регрессор на основе дерева решений в качестве индивидуальной модели.

```
# Модель на основе регрессора AdaBoost
regressor = AdaBoostRegressor(
            DecisionTreeRegressor(max_depth=4),
            n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)
```

Оценим эффективность регрессора.

```
# Вычисление показателей эффективности регрессора AdaBoost
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred )
print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Этот регрессор имеет встроенный метод, который можно вызывать для вычисления относительной важности признаков.

```
# Извлечение важности признаков
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names
```

Нормализуем значения относительной важности признаков.

```
# Нормализация значений важности признаков
feature_importances = 100.0 * (feature_importances /
max(feature_importances))
```

Отсортируем эти значения для отображения в виде диаграммы.

```
# Сортировка и перестановка значений
index_sorted = np.flipud(np.argsort(feature_importances))
```

Расставим метки вдоль оси X для построения столбчатой диаграммы.

```
# Расстановка меток вдоль оси X
pos = np.arange(index_sorted.shape[0]) + 0.5
```

Построим столбчатую диаграмму.

```
# Построение столбчатой диаграммы
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Оценка важности признаков с использованием регрессора
          AdaBoost')
plt.show()
```

Полный код этого примера содержится в файле `feature_importance.py`. После выполнения этого кода на экране отобразится следующая диаграмма (рис. 3.18).

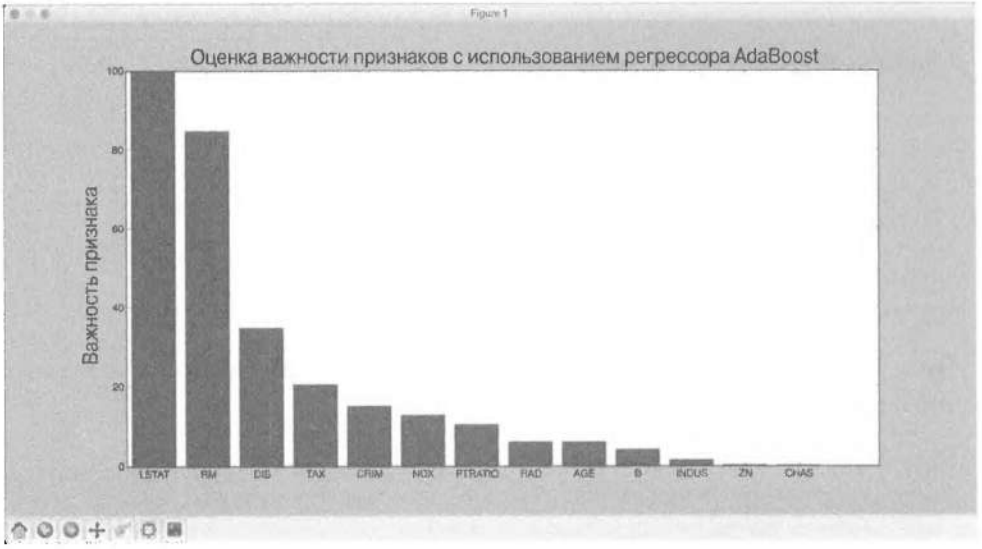

Рис. 3.18

В соответствии с проведенным анализом наиболее важную роль в этом наборе данных играет признак LSTAT.

Прогнозирование интенсивности дорожного движения с помощью классификатора на основе предельно случайных лесов

Применим концепции, изложенные в предыдущем разделе, для решения реальной задачи. Мы используем набор данных, доступный по адресу `https://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor`. В этом наборе содержатся данные об интенсивности дорожного движения во время проведения бейсбольных матчей на стадионе Доджер-стэдиум в Лос-Анджелесе. Чтобы сделать данные более пригодными для анализа, их необходимо подвергнуть предварительной обработке. Предварительно обработанные данные содержатся в файле `traffic_data.txt`. В этом файле каждая строка содержит строковые значения, разделенные запятой. В качестве примера рассмотрим первую строку:

Tuesday,00:00,San Francisco,no,3

Значения в этой строке отформатированы следующим образом: день недели, время суток, команда соперника, двоичное значение, указывающее, проходит ли в данное время матч (yes/no), количество проезжающих транспортных средств.

Нашей целью является прогнозирование количества проезжающих по дороге транспортных средств на основании предоставленной информации. Следовательно, мы должны создать регрессор, способный прогнозировать выходной результат. Создадим такой регрессор на основе предельно случайных лесов. Рассмотрим, как это делается.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,
    mean_absolute_error
from sklearn import cross_validation, preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import classification_report
```

Загрузим данные из файла `traffic_data.txt`.

```
# Загрузка входных данных
input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
      for line in f.readlines():
            items = line[:-1].split(',')
            data.append(items)

data = np.array(data)
```

Содержащиеся среди этих данных нечисловые признаки нуждаются в кодировании. Кроме того, мы должны проследить за тем, чтобы числовые признаки не подвергались кодированию. Для каждого признака, нуждающегося в кодировании, необходимо предусмотреть отдельный кодировщик. Мы должны отслеживать эти кодировщики, поскольку они понадобятся нам, когда мы захотим вычислить результат для неизвестной точки данных. Создадим указанные кодировщики.

```
# Преобразование строковых данных в числовые
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
      if item.isdigit():
            X_encoded[:, i] = data[:, i]
```

```
        else:
                label_encoder.append(preprocessing.LabelEncoder())
                X_encoded[:, i] = label_encoder[-1]
                .fit_transform(data[:, i])
X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

Разобьем данные на обучающий и тестовый наборы.

```
# Разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test =
            cross_validation.train_test_split(
            X, y, test_size=0.25, random_state=5)
```

Обучим регрессор на основе предельно случайных лесов.

```
# Регрессор на основе предельно случайных лесов
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)
```

Вычислим показатели эффективности регрессора на тестовых данных.

```
# Вычисление характеристик эффективности
# регрессора на тестовых данных
y_pred = regressor.predict(X_test)
print("Mean absolute error:",
       round(mean_absolute_error(y_test, y_pred), 2))
```

Рассмотрим, как вычисляется результат для неизвестной точки данных. Для преобразования нечисловых признаков в числовые значения мы используем кодировщики.

```
# Тестирование кодирования на одиночном примере
test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0
for i, item in enumerate(test_datapoint):
      if item.isdigit():
            test_datapoint_encoded[i] = int(test_datapoint[i])
      else:
            test_datapoint_encoded[i] =
int(label_encoder[count].transform(test_datapoint[i]))
            count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)
```

Спрогнозируем результат.

```
# Прогнозирование результата для тестовой точки данных
print("Predicted traffic:",
int(regressor.predict([test_datapoint_encoded])[0]))
```

Полный код этого примера содержится в файле `traffic_prediction.py`. Выполнив этот код, вы получите в качестве выходного результата значение `26`, которое очень близко к фактическому значению. В этом нетрудно убедиться, обратившись к файлу данных.

Резюме

Из этой главы вы узнали о том, что такое ансамблевое обучение и как оно применяется в реальных задачах. Мы обсудили деревья решений и создание классификатора на их основе.

Вы познакомились с понятиями случайных и предельно случайных лесов и методами создания классификаторов на их основе. Мы рассмотрели меры достоверности предсказаний и способы разрешения проблемы дисбаланса классов.

Также было показано, как находить наиболее оптимальные параметры обучения для построения моделей с использованием сеточного поиска. Вы узнали о том, как вычислить относительную важность признаков. Наконец, мы применили технику ансамблевого обучения для решения реальной задачи — предсказания интенсивности дорожного движения с использованием регрессора на основе предельно случайного леса.

В следующей главе мы рассмотрим обучение без учителя и покажем, как обнаруживать закономерности в данных фондового рынка.

4

Распознавание образов с помощью обучения без учителя

В этой главе мы рассмотрим, что такое обучение без учителя и как оно применяется для решения реальных задач. Вы ознакомитесь со следующими темами:

- что такое обучение без учителя;
- кластеризация данных с помощью метода k-средних;
- оценка количества кластеров с помощью алгоритма сдвига среднего;
- оценка качества кластеризации с помощью силуэтных мер;
- что такое гауссовские смешанные модели;
- создание классификаторов на основе гауссовских смешанных моделей;
- поиск подгрупп среди участников фондового рынка с использованием модели распространения сходства;
- сегментирование рынка на основе моделей совершения покупок.

Что такое обучение без учителя

Термин **обучение без учителя** (unsupervised learning) относится к процессу построения модели машинного обучения, не требующей привлечения помеченных тренировочных данных. Машинное обучение без учителя находит применение во многих областях, включая сегментирование рынка, торговля акциями, обработка естественного языка, машинное зрение и др.

В предыдущих главах мы имели дело с данными, с которыми ассоциировались метки (маркеры). В случае помеченных обучающих данных алгоритмы учатся классифицировать данные по этим меткам. В реальных задачах размеченные данные не всегда доступны. Иногда мы просто располагаем множеством данных и должны каким-то образом распределить их по категориям,

которые пока что неизвестны. И здесь на первый план выходит *обучение без учителя* (другие названия — *неконтролируемое, спонтанное обучение, самообучение*). Алгоритмы обучения без учителя пытаются строить модели, способные находить подгруппы в заданном наборе данных, используя различные метрики сходства.

Рассмотрим, как формулируется задача обучения, если оно проводится без учителя. Когда у нас имеется набор данных, не ассоциируемых с какими-либо метками, мы предполагаем, что эти данные генерируются под влиянием скрытых переменных, управляющих их распределением. В таком случае процесс обучения может следовать некой иерархической схеме, используя на начальном этапе индивидуальные точки данных. Далее можно создавать более глубокие уровни представления данных.

Кластеризация данных с помощью метода k-средних

Кластеризация — один из наиболее популярных методов обучения без учителя. Эта методика применяется для анализа данных и выделения кластеров среди них. Для нахождения кластеров задействуют различные меры сходства, такие как евклидово расстояние, позволяющие выделять подгруппы данных. Используя меру сходства, можно оценить связность кластера. Таким образом, кластеризация — это процесс организации данных в подгруппы, элементы которых сходны между собой в соответствии с некоторыми критериями.

Наша задача заключается в том, чтобы идентифицировать скрытые свойства точек данных, определяющие их принадлежность к одной и той же подгруппе. Универсальных метрических характеристик сходства, которые работали бы во всех случаях, не существует. Все определяется конкретикой задачи. Например, нас может интересовать нахождение представительной точки данных для каждой подгруппы или же выбросов. В зависимости от ситуации мы выбираем ту или иную метрику, которая, по нашему мнению, наиболее полно учитывает специфику задачи.

Метод *k-средних* (k-means) — это хорошо известный алгоритм кластеризации. Его использование предполагает, что количество кластеров заранее известно. Далее мы сегментируем данные в K подгрупп, применяя различные атрибуты данных. Мы начинаем с того, что фиксируем количество кластеров и, исходя из этого, классифицируем данные. Основная идея заключается в обновлении положений *центроидов* (центров тяжести кластеров; другое название — *главные точки*) на каждой итерации. Итеративный процесс продолжается до тех пор, пока все центроиды не займут оптимальные положения.

Как нетрудно догадаться, в этом алгоритме выбор начального расположения центроидов играет очень важную роль, поскольку это самым непосредственным образом влияет на конечные результаты. Одна из стратегий состоит в том, чтобы центроиды располагались на как можно большем расстоянии друг от друга. Базовому методу k-средних соответствует случайное расположение центроидов, тогда как в усовершенствованном варианте метода (k-means++) эти точки выбираются алгоритмически из входного списка точек данных. В начале процесса предпринимается попытка расположить центры кластеров на больших расстояниях один от другого, чтобы обеспечить быструю сходимость. Затем мы перебираем данные обучающего набора и улучшаем стартовое разбиение на кластеры посредством отнесения каждой точки к ближайшему кластерному центру.

Завершение описанного перебора всех точек набора данных означает окончание первой итерации. На этом этапе точки оказываются сгруппированными на основании начальных положений центров кластеров. Далее нам необходимо заново вычислить положения центроидов, отталкиваясь от новых кластеров, полученных в конце первой итерации. Получив новый набор K центров, мы повторяем весь процесс, вновь итерируя по набору данных и относя каждую точку к ближайшему центроиду.

В процессе повторения описанных шагов центры кластеров постепенно смещаются к своим устойчивым положениям. После выполнения некоторого количества итераций центры кластеров перестанут смещаться. Это будет свидетельствовать о том, что мы достигли устойчивого расположения центров кластеров. Полученные K центроидов и представляют окончательную модель k-средних, которые будут использоваться для *вывода суждений* (inference).

Чтобы посмотреть, как работает метод кластеризации k-средних, применим его к двумерным данным. Мы используем данные, содержащиеся в файле `data_clustering.txt`. В этом файле каждая строка содержит два числа, разделенных запятой.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics
```

Загрузим входные данные из файла.

```
# Загрузка входных данных
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

Прежде чем применять метод k-средних, необходимо определить количество кластеров.

```
num_clusters = 5
```

Визуализируем входные данные, чтобы увидеть, как выглядит распределение.

```
# Включение входных данных в график
plt.figure()
plt.scatter(X[:,0], X[:,1], marker='o', facecolors='none',
            edgecolors='black', s=80)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Входные данные')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
```

Мы можем получить наглядное подтверждение того, что наши данные состоят из пяти групп. Создадим объект KMeans, используя параметры инициализации. Параметр `init` позволяет задать способ инициализации начальных центров кластеров. Вместо того чтобы выбирать их случайным образом, мы используем для этого параметра значение `k-means++`, которое обеспечивает улучшенный способ выбора положений центроидов, гарантирующий быструю сходимость алгоритма. Параметр `n_clusters` определяет количество кластеров, тогда как параметр `n_init` позволяет указать, сколько раз должен выполниться алгоритм, прежде чем будет принято решение относительно наилучшего результата.

```
# Создание объекта KMeans
kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)
```

Обучим модель k-средних на входных данных.

```
# Обучение модели кластеризации KMeans
kmeans.fit(X)
```

Чтобы визуализировать границы, мы должны создать сетку точек и вычислить модель на всех узлах сетки. Определим шаг сетки.

```
# Определение шага сетки
step_size = 0.01
```

Далее определим саму сетку и убедимся в том, что она охватывает все входные значения.

```
# Определение сетки точек для отображения границ
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_vals, y_vals = np.meshgrid(np.arange(x_min, x_max, step_size),
        np.arange(y_min, y_max, step_size))
```

Спрогнозируем результаты для всех точек сетки, используя обученную модель k-средних.

```
# Предсказание выходных меток для всех точек сетки
output = kmeans.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
```

Отобразим на графике выходные значения и выделим каждую область своим цветом.

```
# Графическое отображение областей и выделение их цветом
output = output.reshape(x_vals.shape)
plt.figure()
plt.clf()
plt.imshow(output, interpolation='nearest',
           extent=(x_vals.min(), x_vals.max(),
                   y_vals.min(), y_vals.max()),
           cmap=plt.cm.Paired,
           aspect='auto',
           origin='lower')
```

Отобразим входные данные на выделенных цветом областях.

```
# Отображение входных точек
plt.scatter(X[:,0], X[:,1], marker='o', facecolors='none',
            edgecolors='black', s=80)
```

Отобразим на графике центры кластеров, полученные с использованием метода k-средних.

```
# Отображение центров кластеров
cluster_centers = kmeans.cluster_centers_
plt.scatter(cluster_centers[:,0], cluster_centers[:,1],
            marker='o', s=210, linewidths=4, color='black',
            zorder=12, facecolors='black')
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Границы кластеров')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

Полный код этого примера содержится в файле `kmeans.py`. В процессе выполнения этого кода на экране отобразятся два графика. Первый из них представляет входные данные (рис. 4.1).

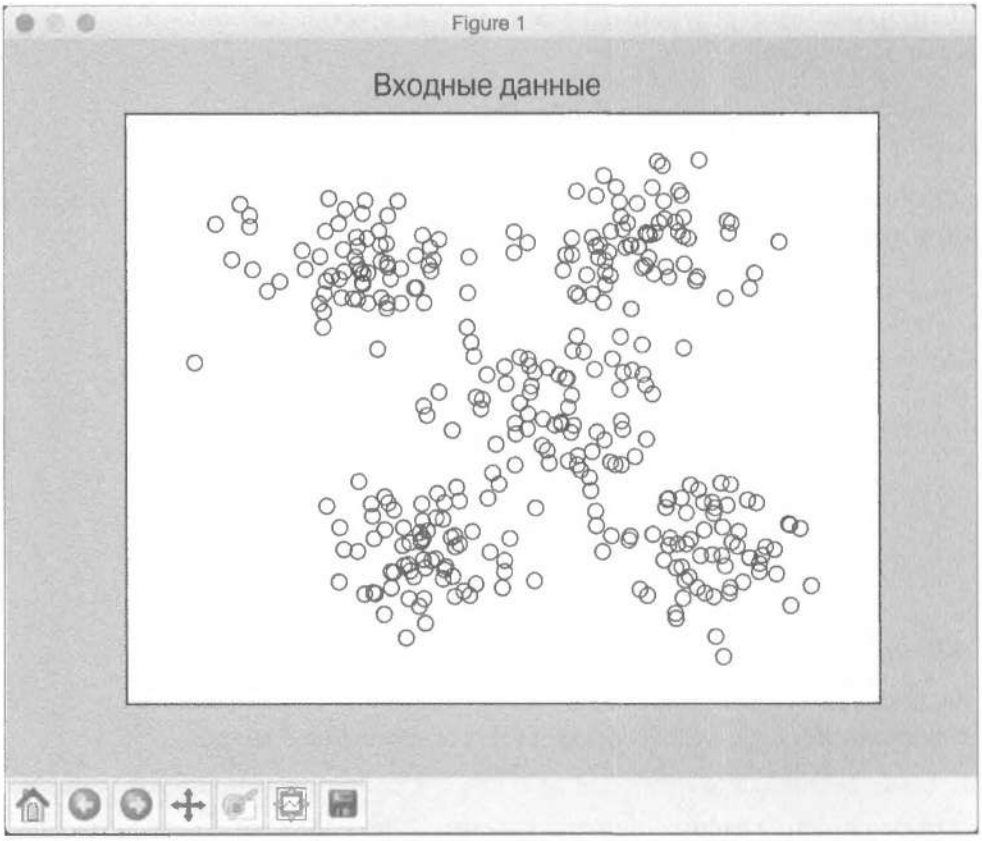

Рис. 4.1

Второй экранный снимок представляет границы, полученные по методу k-средних (рис. 4.2).

Сплошными черными кружками обозначены центры кластеров.

Оценка количества кластеров с использованием метода сдвига среднего

Метод сдвига среднего (Mean Shift) — мощный алгоритм, используемый в обучении без учителя. Этот непараметрический алгоритм часто применяется для решения задач кластеризации. Он называется *непараметрическим*, поскольку в нем не используются какие-либо допущения относительно базового распределения данных.

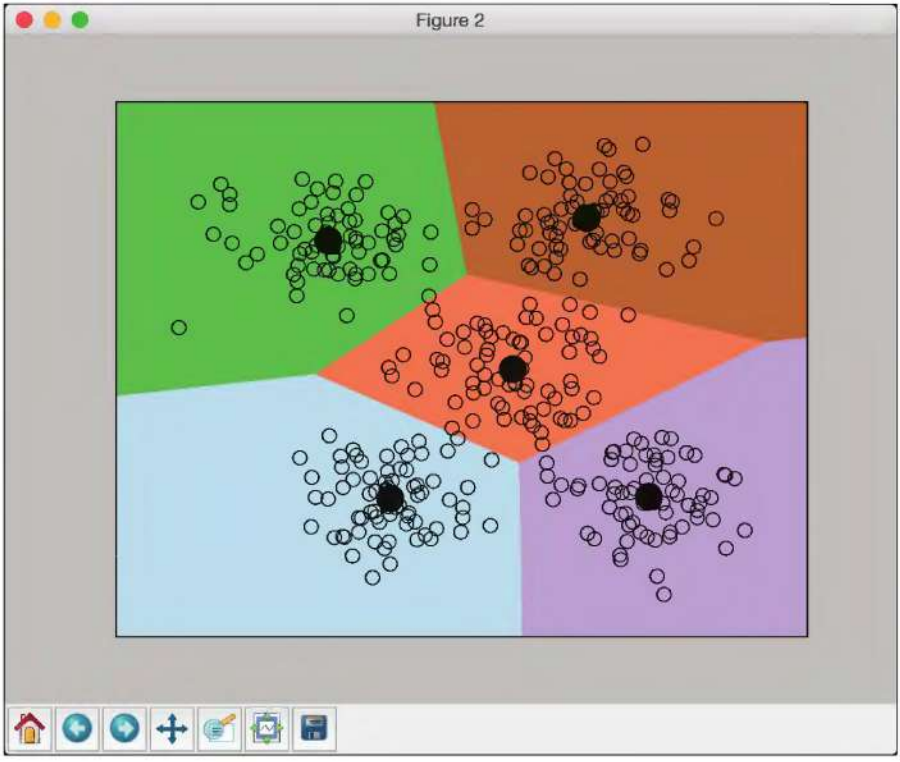

Рис. 4.2. (См. цветную вклейку; адрес указан во введении)

Этот метод контрастирует с параметрическими подходами, в которых предполагается, что базовые данные подчиняются стандартному распределению вероятностей. Метод сдвига среднего находит множество применений в таких областях, как отслеживание объектов и анализ данных в реальном времени.

В алгоритме сдвига среднего все пространство признаков рассматривается как функция распределения вероятности. Мы начинаем с тренировочного набора данных и предполагаем, что данная выборка соответствует функции распределения вероятности. В рамках такого подхода кластеры соответствуют максимумам базового распределения. Если существуют K кластеров, то в базовом распределении существуют K пиков, и метод сдвига среднего идентифицирует эти пики.

Целью метода сдвига среднего является идентификация позиций центров кластеров. Для каждой точки данных обучающего набора определяется окружающее ее окно. Затем для этого окна вычисляется центроид, и

положение окна обновляется так, чтобы оно соответствовало положению нового центроида. Далее процесс повторяется для нового центроида посредством определения окна вокруг него. По мере продолжения описанного процесса мы приближаемся к пику кластера. Каждая точка данных будет перемещаться в направлении кластера, которому она принадлежит. Это перемещение осуществляется в направлении области с более высокой плотностью вероятности.

Мы продолжаем процесс смещения центроидов, также называемых *средними*, к пикам каждого кластера. Поскольку средние при этом смещаются, метод и называется *сдвиг среднего*. Этот процесс продолжается до тех пор, пока алгоритм не сойдется, т.е. пока центроиды не перестанут смещаться.

Оценим максимальное количество кластеров в заданном наборе данных с помощью алгоритма сдвига среднего. Для анализа мы используем данные, содержащиеся в файле `data_clustering.txt`. Это тот же самый файл, который мы задействовали в разделе, посвященном методу k-средних.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from itertools import cycle
```

Загрузим входные данные.

```
# Загрузка данных из входного файла
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

Оценим ширину окна входных данных. *Ширина окна* (bandwidth) — это параметр базового процесса оценки плотности распределения ядра в алгоритме сдвига среднего. Ширина окна влияет на общую скорость сходимости алгоритма и результирующее количество кластеров. Следовательно, этот параметр играет очень важную роль. Выбор слишком малой ширины окна может привести к слишком большому количеству кластеров, тогда как завышенные значения этого параметра будут приводить к слиянию отдельных кластеров.

Параметр `quantile` влияет на ширину окна. Более высокие значения этого параметра увеличивают ширину окна, тем самым уменьшая количество кластеров.

```
# Оценка ширины окна для X
bandwidth_X = estimate_bandwidth(X, quantile=0.1, n_samples=len(X))
```

Обучим модель кластеризации на основе сдвига среднего, используя полученную оценку ширины окна.

```
# Кластеризация данных методом сдвига среднего
meanshift_model = MeanShift(bandwidth=bandwidth_X, bin_seeding=True)
meanshift_model.fit(X)
```

Извлечем центры всех кластеров.

```
# Извлечение центров кластеров
cluster_centers = meanshift_model.cluster_centers_
print('\nCenters of clusters:\n', cluster_centers)
```

Извлечем количество кластеров.

```
# Оценка количества кластеров
labels = meanshift_model.labels_
num_clusters = len(np.unique(labels))
print("\nNumber of clusters in input data =", num_clusters)
```

Визуализируем точки данных.

```
# Отображение на графике точек и центров кластеров
plt.figure()
markers = 'o*xvs'
for i, marker in zip(range(num_clusters), markers):
    # Отображение на графике точек, принадлежащих
    # текущему кластеру
    plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker,
                color='black')
```

Отобразим на графике центр текущего кластера.

```
    # Отображение на графике центра кластера
    cluster_center = cluster_centers[i]
    plt.plot(cluster_center[0], cluster_center[1], marker='o',
             markerfacecolor='black', markeredgecolor='black',
             markersize=15)

plt.title('Кластеры')
plt.show()
```

Полный код этого примера содержится в файле `mean_shift.py`. После выполнения этого кода на экране отобразится следующий график, представляющий кластеры и их центры (рис. 4.3).

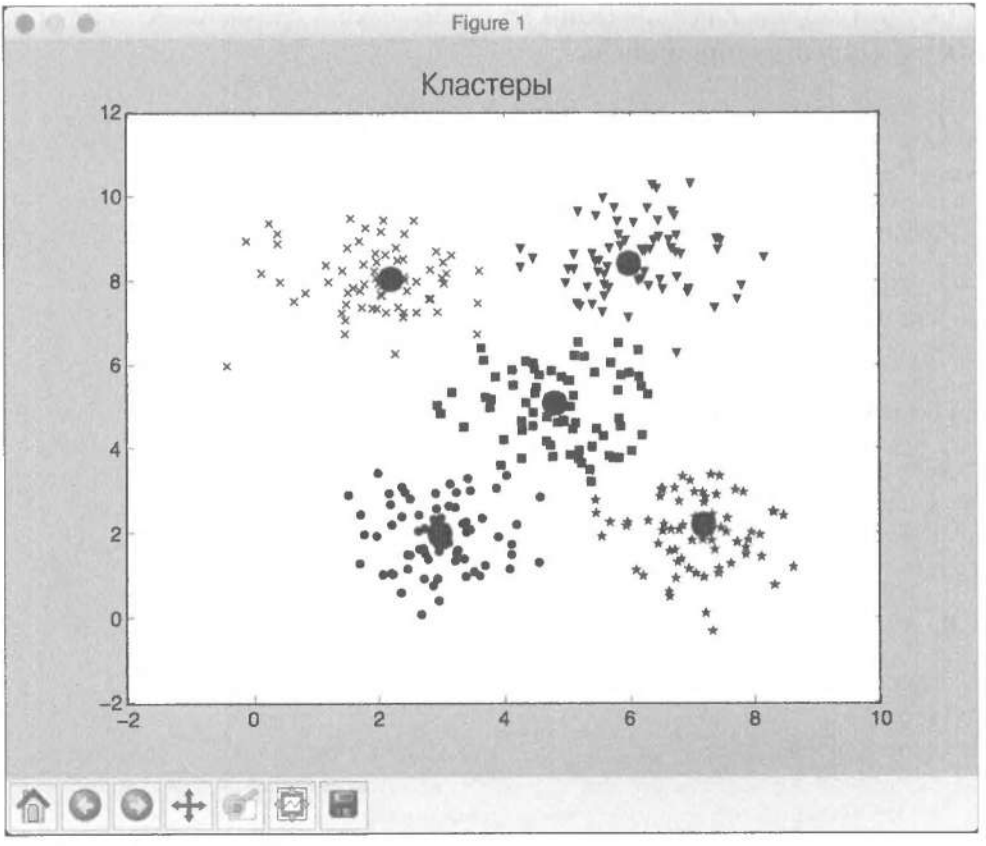

Рис. 4.3

Кроме того, в окне терминала отобразится следующая информация (рис. 4.4).

```
Centers of clusters:
 [[ 2.95568966  1.95775862]
 [ 7.17563636  2.18145455]
 [ 2.17603774  8.03283019]
 [ 5.97960784  8.39078431]
 [ 4.81044444  5.07111111]]

Number of clusters in input data = 5
```

Рис. 4.4

Оценка качества кластеризации с помощью силуэтных оценок

Если данные естественным образом организованы в виде некоторого количества различимых кластеров, то уже одно только их визуальное исследование позволяет сделать определенные заключения относительно их свойств. Однако на практике такое встречается редко. В реальных задачах мы имеем дело с огромным количеством неструктурированных данных. Поэтому мы нуждаемся в каких-то способах количественной оценки качества кластеризации.

Силуэтная мера — это интегральная характеристика связности и разделения кластеров данных. Она дает оценку того, насколько хорошо каждая точка данных вписывается в свой кластер. *Силуэтная оценка* (silhouette score) — это метрика, измеряющая степень сходства точки данных с собственным кластером по сравнению с другими кластерами. Силуэтная оценка работает с любой метрикой сходства.

Силуэтная оценка вычисляется для каждой точки данных по следующей формуле:

$$силуэтная\ оценка = (p - q) / max(p, q)$$

где p — среднее расстояние до точек ближайшего кластера, частью которого данная точка не является, а q — среднее расстояние до всех точек в кластере данной точки.

Значения силуэтной оценки находятся в пределах от –1 до 1. Значения, близкие к 1, указывают на тесное сходство точки данных с другими точками данного кластера, тогда как значения, близкие к –1, указывают на отсутствие такого сходства. Сказанное можно интерпретировать следующим образом: если вы получили слишком много точек с отрицательной силуэтной оценкой, то это может означать, что результирующее количество кластеров слишком мало или слишком велико. В подобных ситуациях следует вновь запустить алгоритм для нахождения оптимального количества кластеров.

Посмотрим, как можно оценить качество кластеризации, используя силуэтные оценки. Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.cluster import KMeans
```

Мы используем данные, содержащиеся в файле `data_quality.txt`. В этом файле каждая строка содержит два числа, разделенных запятой.

```
# Загрузка данных из входного файла
X = np.loadtxt('data_quality.txt', delimiter=',')
```

Инициализируем переменные. Массив values будет содержать список значений, итерируя по которым мы хотим найти оптимальное количество кластеров.

```
# Инициализация переменных
scores = []
values = np.arange(2, 10)
```

Выполним цикл по всем значениям, создавая модель k-средних на каждой итерации.

```
# Итерирование в определенном диапазоне значений
for num_clusters in values:
    # Обучение модели кластеризации KMeans
    kmeans = KMeans(init='k-means++', n_clusters=num_clusters,
                    n_init=10)
    kmeans.fit(X)
```

Получим силуэтную оценку для текущей модели кластеризации, используя евклидовы расстояния.

```
score = metrics.silhouette_score(X, kmeans.labels_,
metric='euclidean', sample_size=len(X))
```

Выведем силуэтную оценку для текущего значения.

```
print("\nNumber of clusters =", num_clusters)
print("Silhouette score =", score)
scores.append(score)
```

Визуализируем силуэтные оценки для различных значений.

```
# Отображение силуэтных оценок на графике
plt.figure()
plt.bar(values, scores, width=0.7, color='black', align='center')
plt.title('Зависимость силуэтной оценки от количества кластеров')
```

Извлечем наилучшую оценку и соответствующее значение для количества кластеров.

```
# Извлечение наилучшей оценки и оптимального
# количества кластеров
num_clusters = np.argmax(scores) + values[0]
print('\nOptimal number of clusters =', num_clusters)
```

Визуализируем входные данные.

```
# Отображение данных на графике
plt.figure()
plt.scatter(X[:,0], X[:,1], color='black', s=80, marker='o',
            facecolors='none')
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Входные данные')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

plt.show()
```

Полный код этого примера содержится в файле `clustering_quality.py`. В процессе выполнения этого кода на экране отобразятся два графика. На первом из них представлены входные данные (рис. 4.5).

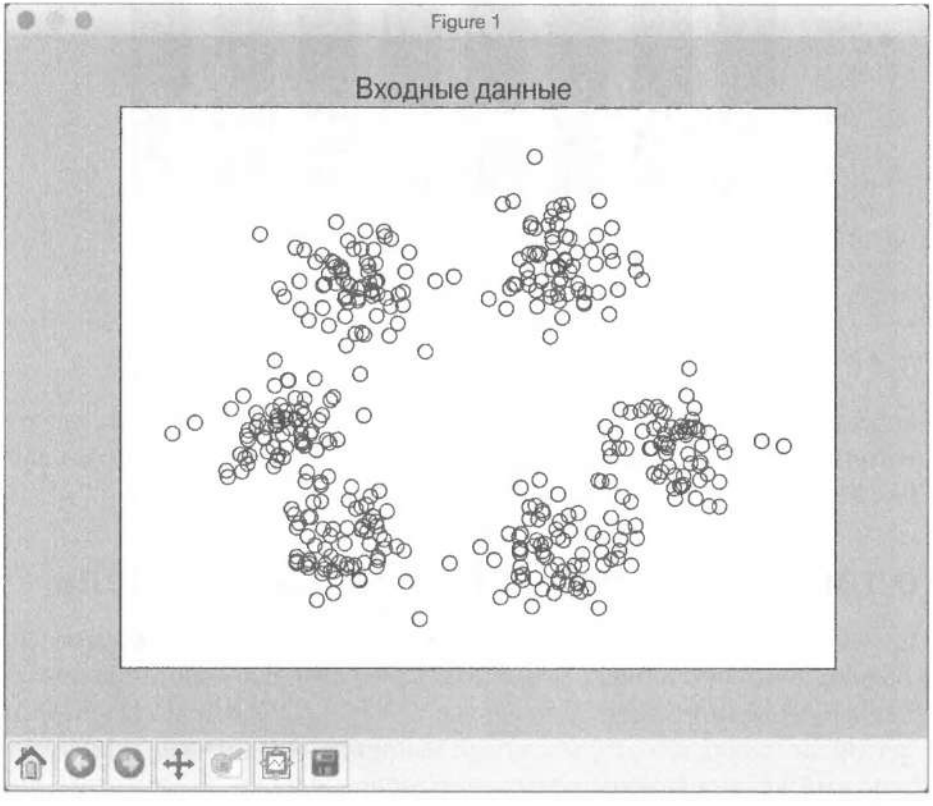

Рис. 4.5

Среди наших данных отчетливо выделяются шесть кластеров. На втором экранном снимке представлены оценки для различных значений количества кластеров (рис. 4.6).

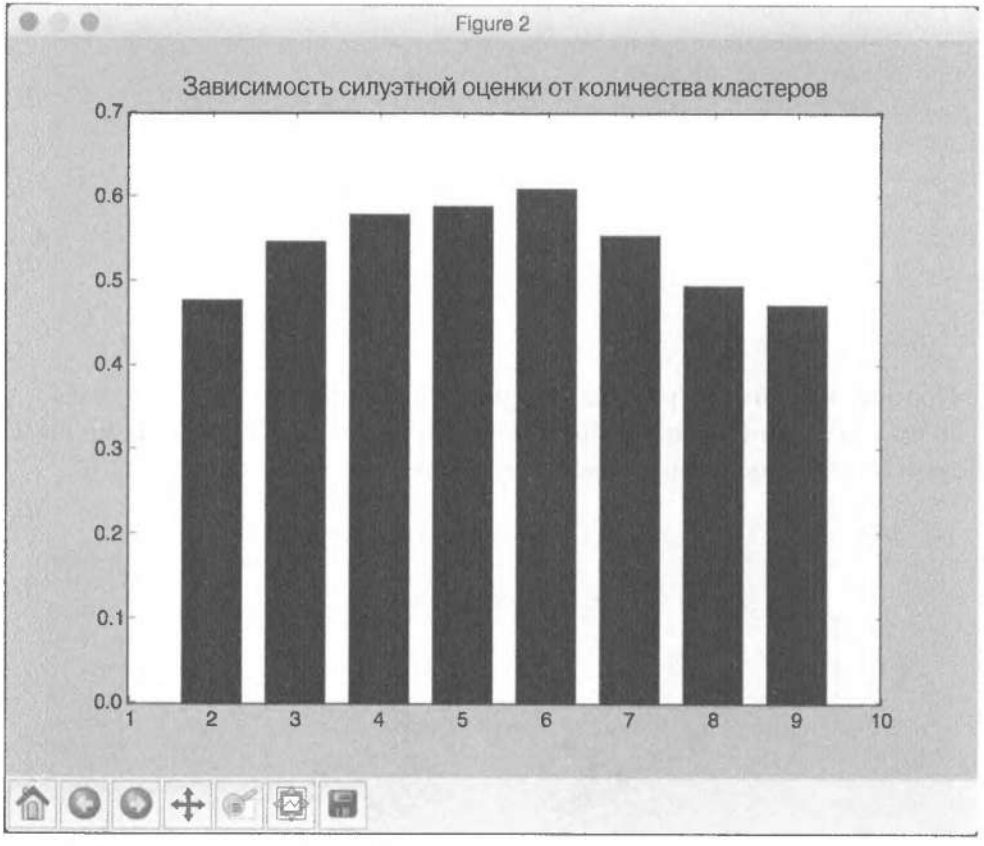

Рис. 4.6

Нетрудно убедиться в том, что силуэтная оценка имеет максимальное значение при количестве кластеров, равном 6, что согласуется с нашими данными. В окне терминала отобразится следующая информация (рис. 4.7).

Что такое смешанные гауссовские модели

Прежде чем приступить к обсуждению **смешанных гауссовских моделей** (Gaussian mixture models, *сокр.* GMM), обсудим, что собой представляют смешанные модели вообще. *Смешанная модель* — это тип моделей плотности распределения, в которых предполагается, что данные управляются несколькими компонентными законами распределения. Если эти распределения гауссовы, то модель называется гауссовской моделью. Компонентные

распределения комбинируются, предоставляя многомодальную функцию распределения, которая и становится смешанной моделью.

```
Number of clusters = 2
Silhouette score = 0.477626248705

Number of clusters = 3
Silhouette score = 0.547174241173

Number of clusters = 4
Silhouette score = 0.579480188969

Number of clusters = 5
Silhouette score = 0.589003263565

Number of clusters = 6
Silhouette score = 0.609690411895

Number of clusters = 7
Silhouette score = 0.554310234032

Number of clusters = 8
Silhouette score = 0.494433661954

Number of clusters = 9
Silhouette score = 0.471414689437

Optimal number of clusters = 6
```

Рис. 4.7

Рассмотрим пример, иллюстрирующий работу смешанных моделей. Мы хотим создать модель покупательских привычек населения Южной Америки. Для этого можно было бы моделировать континент в целом и построить единую модель на основе полных данных. Однако мы знаем, что люди в разных странах имеют разные покупательские привычки. Поэтому необходимо понять, в чем заключаются эти различия.

Если мы хотим получить качественную репрезентативную модель, то должны учесть все возможные вариации в пределах континента. В таком случае мы можем использовать отдельные модели для моделирования покупательских привычек в разных странах, а затем объединить их в смешанную модель. Благодаря этому мы не упустим нюансов базового поведения населения отдельных стран. За счет отказа от навязывания какой-либо одной базовой модели для всех стран мы получаем возможность построения более точной модели.

Стоит подчеркнуть, что подобные смешанные модели являются *полупараметрическими* в том смысле, что они частично зависят от набора

предопределенных функций. Такие модели обеспечивают получение более высокой точности и гибкости моделирования базовых распределений данных. Они также сглаживают пропуски, обусловленные наличием разреженных данных.

Если мы определяем функцию, то смешанная модель переходит из категории полупараметрических в категорию параметрических моделей. Следовательно, GMM — это параметрическая модель, представленная в виде взвешенной суммы компонентных гауссовских функций. Мы предполагаем, что данные генерируются объединенным набором гауссовских моделей. GMM — очень мощный инструмент, который применяется во многих областях. Параметры GMM оцениваются на основе обучающих данных с использованием таких алгоритмов, как **EM-алгоритм (Expectation-Maximization** — принцип максимума правдоподобия) или **MAP-алгоритм (Maximum A-Posteriori** — принцип максимума апостериорной вероятности). К числу популярных применений GMM относятся извлечение изображений из баз данных, моделирование флуктуаций фондового рынка, биометрическая идентификация личности и др.

Создание классификатора на основе гауссовской смешанной модели

Создадим классификатор на основе смешанной гауссовской модели. Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import patches

from sklearn import datasets
from sklearn.mixture import GMM
from sklearn.cross_validation import StratifiedKFold
```

Для анализа мы используем набор данных `iris`, доступный в библиотеке `scikit-learn`.

```
# Загрузка набора данных iris
iris = datasets.load_iris()
```

Разобьем данные на обучающий и тестовый наборы данных в пропорции 80/20. Параметр `n_folds` позволяет указать количество поднаборов, которые будут получены. Мы используем значение 5, т.е. набор данных будет разбит на пять частей. Четыре части будут использованы для обучения, а оставшаяся часть — для тестирования, что соответствует пропорции 80/20.

```
# Разбиение данных на обучающий и тестовый наборы
# (в пропорции 80/20)
indices = StratifiedKFold(iris.target, n_folds=5)
```

Извлечем обучающий набор данных.

```
# Используем первый набор
train_index, test_index = next(iter(indices))

# Извлечем обучающие данные и метки
X_train = iris.data[train_index]
y_train = iris.target[train_index]

# Извлечем тестовые данные и метки
X_test = iris.data[test_index]
y_test = iris.target[test_index]
```

Извлечем количество классов в обучающих данных.

```
# Извлечение количества классов
num_classes = len(np.unique(y_train))
```

Создадим классификатор на основе GMM, используя соответствующие параметры. Параметр `n_components` позволяет указать количество компонент в базовом распределении. В данном случае это будет количество различных классов в наших данных. Кроме того, мы должны задать используемый тип ковариации. В данном случае мы будем использовать полную ковариацию. Параметр `init_params` управляет параметрами, которые должны обновляться в процессе обучения. Для него мы используем значение `wc`, которому соответствует обновление параметров *весов* (weights) и *ковариации* (covariance). Параметр `n_iter` задает количество итераций EM-алгоритма, которые должны быть выполнены в процессе обучения.

```
# Создание GMM
classifier = GMM(n_components=num_classes, covariance_type='full',
                 init_params='wc', n_iter=20)
```

Инициализируем средние классификатора.

```
# Инициализация средних GMM
classifier.means_ = np.array([X_train[y_train == i].mean(axis=0)
                              for i in range(num_classes)])
```

Обучим классификатор на основе смешанной гауссовской модели, используя тренировочные данные.

```
# Обучение GMM-классификатора
classifier.fit(X_train)
```

Визуализируем границы классификатора. Для оценки эллиптических границ вокруг кластеров мы используем собственные векторы и собственные значения. Краткие сведения, касающиеся собственных векторов и собственных значений, можно найти по адресу `https://www.math.hmc.edu/calculus/tutorials/eigenstuff`. Построим соответствующий график.

```
# Вычерчивание границ
plt.figure()
colors = 'bgr'
for i, color in enumerate(colors):
    # Извлечение собственных значений и собственных векторов
    eigenvalues, eigenvectors = np.linalg.eigh(
            classifier._get_covars()[i][:2, :2])
```

Нормализуем первый собственный вектор.

```
    # Нормализация первого собственного вектора
    norm_vec = eigenvectors[0] / np.linalg.norm(eigenvectors[0])
```

Чтобы обеспечить точное отображение распределения, эллипсы следует повернуть. Оценим величину угла поворота.

```
    # Извлечение угла наклона
    angle = np.arctan2(norm_vec[1], norm_vec[0])
    angle = 180 * angle / np.pi
```

Увеличим размеры эллипсов для визуализации. Размерами эллипсов управляют собственные значения.

```
    # Масштабный множитель для увеличения эллипсов
    # (выбрано произвольное значение, которое нас удовлетворяет)
    scaling_factor = 8
    eigenvalues *= scaling_factor
```

Вычертим эллипсы.

```
    # Вычерчивание эллипсов
    ellipse = patches.Ellipse(classifier.means_[i, :2],
            eigenvalues[0], eigenvalues[1], 180 + angle,
            color=color)
    axis_handle = plt.subplot(1, 1, 1)
    ellipse.set_clip_box(axis_handle.bbox)
    ellipse.set_alpha(0.6)
    axis_handle.add_artist(ellipse)
```

Отложим входные данные на графике.

```
# Откладывание входных данных на графике
colors = 'bgr'
for i, color in enumerate(colors):
    cur_data = iris.data[iris.target == i]
    plt.scatter(cur_data[:,0], cur_data[:,1], marker='o',
                facecolors='none', edgecolors='black', s=40,
                label=iris.target_names[i])
```

Отобразим на графике тестовые данные.

```
    test_data = X_test[y_test == i]
    plt.scatter(test_data[:,0], test_data[:,1], marker='s',
                facecolors='black', edgecolors='black', s=40,
                label=iris.target_names[i])
```

Вычислим прогнозируемый результат для обучающих и тестовых данных.

```
# Вычисление прогнозных результатов
# для обучающих и тестовых данных

y_train_pred = classifier.predict(X_train)
accuracy_training = np.mean(y_train_pred.ravel() ==
                            y_train.ravel()) * 100
print('Accuracy on training data =', accuracy_training)

y_test_pred = classifier.predict(X_test)
accuracy_testing = np.mean(y_test_pred.ravel() ==
                           y_test.ravel()) * 100
print('Accuracy on testing data =', accuracy_testing)

plt.title('GMM-классификатор')
plt.xticks(())
plt.yticks(())
plt.show()
```

Полный код этого примера содержится в файле `gmm_classifier.py`. В процессе выполнения этого кода на экране отобразится следующий график (рис. 4.8).

Входные данные включают три распределения. Эти базовые распределения входных данных представлены тремя эллипсами различных размеров. В окне терминала отобразится следующая информация.

```
Accuracy on training data = 87.5
Accuracy on testing data = 86.6666666667
```

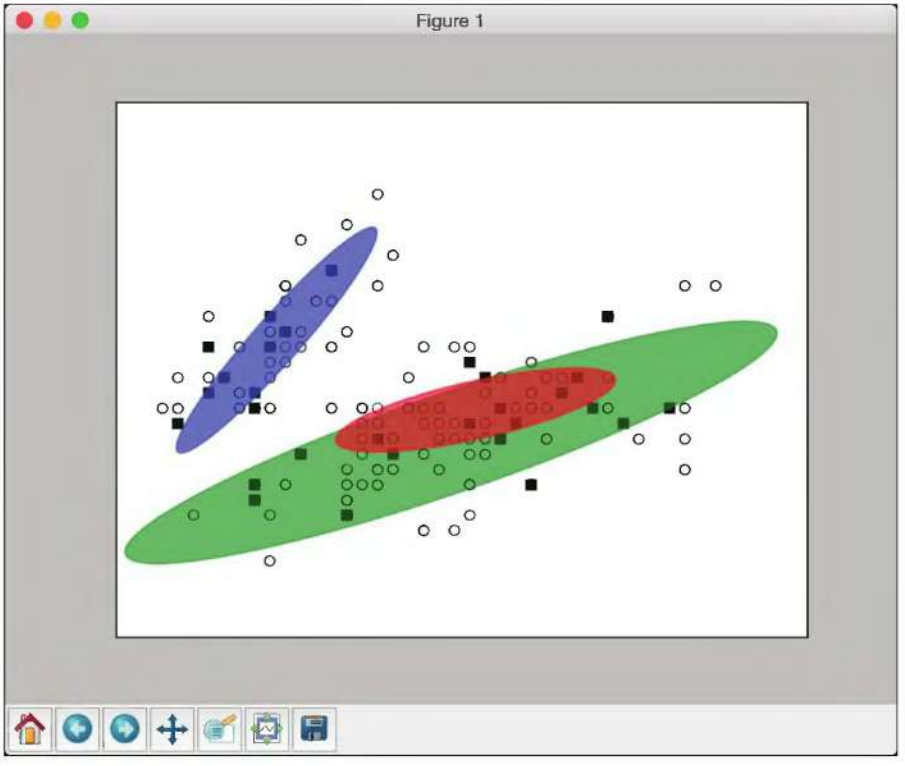

Рис. 4.8. (См. цветную вклейку; адрес указан во введении)

Нахождение подгрупп на фондовом рынке с использованием модели распространения сходства

Распространение сходства (affinity propagation) — это алгоритм кластеризации, выполнение которого не требует предварительного указания используемого количества кластеров. В силу своей общности и простоты реализации он широко применяется в различных областях. Этот алгоритм находит представительные элементы кластеров, так называемые *образцы* (exemplars), используя технику "обмена сообщениями" между точками данных. Мы начинаем с определения мер сходства, которые должен использовать алгоритм. Первоначально в качестве потенциальных образцов рассматриваются все обучающие точки данных. Далее точки данных "общаются" между собой до тех пор, пока не удастся определить оптимальный набор представительных образцов.

Элементы кластеров попарно обмениваются сообщениями двух категорий, содержащими информацию о **пригодности** (responsibility) и **доступности** (availability) элементов для роли образцов. Сообщения первой категории отсылаются элементами кластера потенциальным образцам и указывают на то, насколько хорошо точка данных подходила бы для того, чтобы быть элементом кластера данного образца. Сообщения второй категории отсылаются потенциальными образцами потенциальным элементам кластера и указывают на то, насколько хорошо они подошли бы для того, чтобы служить образцом. Этот процесс продолжается до тех пор, пока алгоритм не сойдется к оптимальному набору образцов.

Также имеется параметр `preference`, управляющий количеством образцов, которые должны быть найдены. Если вы выберете для него завышенное значение, то это приведет к тому, что алгоритм найдет слишком большое количество кластеров. Следствием заниженного значения этого параметра будет слишком малое количество кластеров.

Используем модель распространения сходства для нахождения подгрупп среди участников фондового рынка. В качестве управляющего признака будем использовать вариацию котировок между открытием и закрытием биржи.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import datetime
import json

import numpy as np
import matplotlib.pyplot as plt
from sklearn import covariance, cluster
from matplotlib.finance import quotes_historical_yahoo_ochl
as quotes_yahoo
```

Мы используем данные фондового рынка, доступные в библиотеке `matplotlib`. Привязки символических обозначений компаний к их полным названиям содержатся в файле `company_symbol_mapping.json`.

```
# Входной файл с символическими обозначениями компаний
input_file = 'company_symbol_mapping.json'
```

Загрузим из файла массив соответствия символов компаний их полным названиям.

```
# Загрузка привязок символов компаний к их полным названиям
with open(input_file, 'r') as f:
    company_symbols_map = json.loads(f.read())

symbols, names = np.array(list(company_symbols_map.items())).T
```

Загрузим данные котировок из библиотеки matplotlib.

```
# Загрузка архивных данных котировок
start_date = datetime.datetime(2003, 7, 3)
end_date = datetime.datetime(2007, 5, 4)
quotes = [quotes_yahoo(symbol, start_date, end_date,
          asobject=True) for symbol in symbols]
```

Вычислим разности между котировками при открытии и закрытии биржи.

```
# Извлечение котировок, соответствующих
# открытию и закрытию биржи
opening_quotes = np.array([quote.open for quote in
quotes]).astype(np.float)
closing_quotes = np.array([quote.close for quote in
quotes]).astype(np.float)

# Вычисление разности между двумя видами котировок
quotes_diff = closing_quotes - opening_quotes
```

Нормализуем данные.

```
# Нормализация данных
X = quotes_diff.copy().T
X /= X.std(axis=0)
```

Создадим модель графа.

```
# Создание модели графа
edge_model = covariance.GraphLassoCV()
```

Обучим модель.

```
# Обучение модели
with np.errstate(invalid='ignore'):
    edge_model.fit(X)
```

Создадим модель кластеризации на основе распространения сходства, используя только что обученную краевую модель.

```
# Создание модели кластеризации на основе
# распространения сходства
_, labels = cluster.affinity_propagation(edge_model.covariance_)
num_labels = labels.max()
```

Выведем результат.

```
# Вывод результата кластеризации
for i in range(num_labels + 1):
    print("Cluster", i+1, "==>", ', '.join(names[labels == i]))
```

Полный код этого примера содержится в файле `stocks.py`. После выполнения этого кода в окне терминала отобразится следующая информация (рис. 4.9).

```
Clustering of stocks based on difference in opening and closing quotes:

Cluster 1 ==> Kraft Foods
Cluster 2 ==> CVS, Walgreen
Cluster 3 ==> Amazon, Yahoo
Cluster 4 ==> Cablevision
Cluster 5 ==> Pfizer, Sanofi-Aventis, GlaxoSmithKline, Novartis
Cluster 6 ==> HP, General Electrics, 3M, Microsoft, Cisco, IBM, Texas instruments, Dell
Cluster 7 ==> Coca Cola, Kimberly-Clark, Pepsi, Procter Gamble, Kellogg, Colgate-Palmolive
Cluster 8 ==> Comcast, Wells Fargo, Xerox, Home Depot, Wal-Mart, Marriott, Navistar, DuPont de Nemours, A
merican express, Ryder, JPMorgan Chase, AIG, Time Warner, Bank of America, Goldman Sachs
Cluster 9 ==> Canon, Unilever, Mitsubishi, Apple, Mc Donalds, Boeing, Toyota, Caterpillar, Ford, Honda, S
AP, Sony
Cluster 10 ==> Valero Energy, Exxon, ConocoPhillips, Chevron, Total
Cluster 11 ==> Raytheon, General Dynamics, Lookheed Martin, Northrop Grumman
```

Рис. 4.9

Этот вывод представляет различные группы участников фондового рынка за исследованный период. Следует отметить, что порядок следования кластеров в выводе, который вы получите, может отличаться от приведенного.

Сегментирование рынка на основе моделей совершения покупок

В этом разделе в качестве примера применения методик машинного обучения без учителя мы рассмотрим сегментирование рынка на основе данных о покупательских привычках потребителей. В файле `sales.csv` содержатся нужные нам данные о количестве товара, проданного в различных магазинах одежды. Наша цель — определить стереотипы поведения покупателей и сегментировать рынок на основании информации об объемах продажи товаров в этих магазинах.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import csv

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
```

Загрузим данные из входного файла. Поскольку это *csv*-файл, мы можем использовать специальный объект Python, предназначенный для чтения данных такого типа, и преобразовать данные в массив `NumPy`.

```
# Загрузка данных из входного файла
input_file = 'sales.csv'
file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
```

```
X = []
for count, row in enumerate(file_reader):
    if not count:
        names = row[1:]
        continue

    X.append([float(x) for x in row[1:]])

# Преобразование данных в массив numpy
X = np.array(X)
```

Оценим ширину окна входных данных.

```
# Оценка ширины окна входных данных
bandwidth = estimate_bandwidth(X, quantile=0.8, n_samples=len(X))
```

Обучим модель сдвига среднего, основанную на оцененной ширине окна.

```
# Вычисление кластеризации методом сдвига среднего
meanshift_model = MeanShift(bandwidth=bandwidth,
                            bin_seeding=True)
meanshift_model.fit(X)
```

Извлечем маркеры и центры каждого кластера.

```
labels = meanshift_model.labels_
cluster_centers = meanshift_model.cluster_centers_
num_clusters = len(np.unique(labels))
```

Выведем количество кластеров и кластерные центры.

```
print("\nNumber of clusters in input data =", num_clusters)

print("\nCenters of clusters:")
print('\t'.join([name[:3] for name in names]))
for cluster_center in cluster_centers:
    print('\t'.join([str(int(x)) for x in cluster_center]))
```

Мы имеем дело с шестимерными данными. Давайте визуализируем двумерные данные, сформированные с использованием второго и третьего измерений.

```
# Извлечение двух признаков в целях визуализации
cluster_centers_2d = cluster_centers[:, 1:3]
```

Отобразим центры кластеров на графике.

```
# Отображение центров кластеров
plt.figure()
plt.scatter(cluster_centers_2d[:,0], cluster_centers_2d[:,1],
            s=120, edgecolors='black', facecolors='none')
```

```
offset = 0.25
plt.xlim(cluster_centers_2d[:,0].min() - offset *
         cluster_centers_2d[:,0].ptp(),
         cluster_centers_2d[:,0].max() + offset *
         cluster_centers_2d[:,0].ptp(),)
plt.ylim(cluster_centers_2d[:,1].min() - offset *
         cluster_centers_2d[:,1].ptp(),
         cluster_centers_2d[:,1].max() + offset *
         cluster_centers_2d[:,1].ptp())

plt.title('Центры 2D-кластеров')
plt.show()
```

Полный код этого примера содержится в файле `market_segmentation.py`. В процессе выполнения кода на экране отобразится следующий график (рис. 4.10).

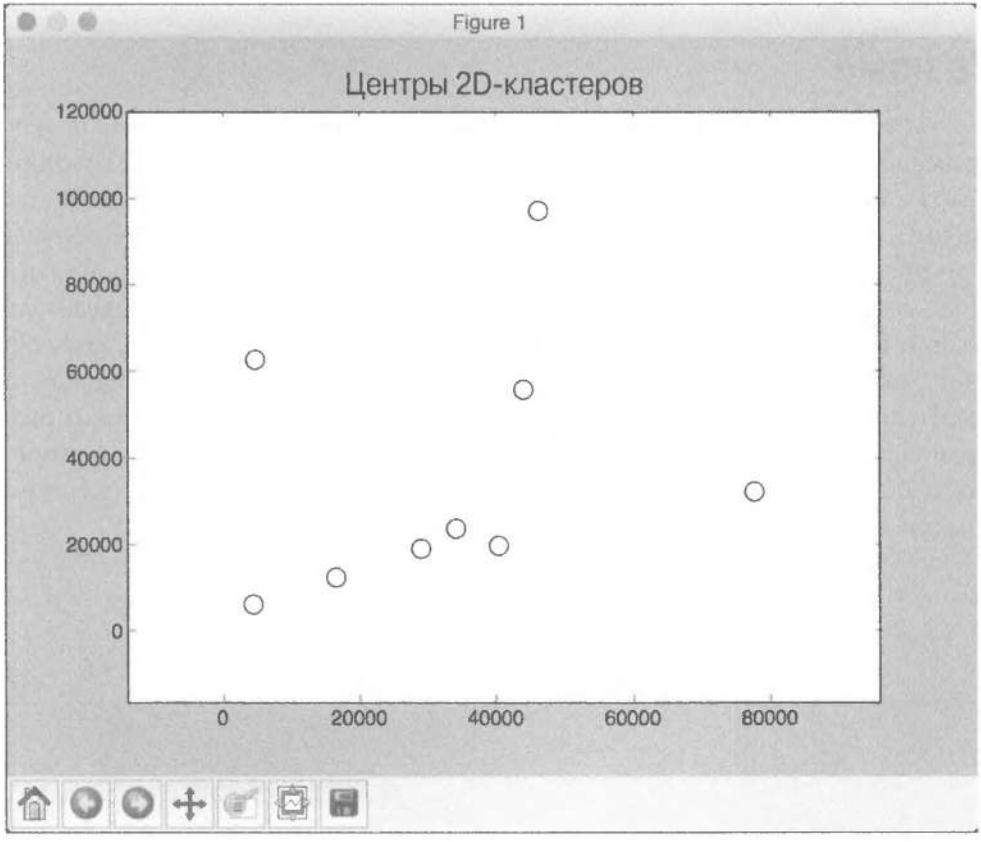

Рис. 4.10

В окне терминала отобразится следующая информация (рис. 4.11).

```
Number of clusters in input data = 9

Centers of clusters:
Tsh     Tan     Hal     Tur     Tub     Swe
9823    4637    6539    2607    2228    1239
38589   44199   56158   5030    24674   4125
7852    4939    63081   134     40066   1332
35314   16745   12775   66900   1298    5613
22617   77873   32543   1005    21035   837
104972  29186   19415   16016   5060    9372
38741   40539   20120   35059   255     50710
28333   34263   24065   5575    4229    18076
14987   46397   97393   1127    37315   3235
```

Рис. 4.11

Резюме

Эта глава началась с обсуждения машинного обучения без учителя и его применений. Затем вы узнали о том, что такое кластеризация и как кластеризовать данные с помощью метода k-средних. После этого было показано, как оценить количество кластеров с помощью алгоритма сдвига среднего. Мы обсудили силуэтные оценки и показали, как они используются для оценивания качества кластеризации. Вы узнали о том, что такое смешанные гауссовские модели и как построить классификатор на их основе. Кроме того, мы обсудили модель распространения сходства и использовали ее для нахождения подгрупп участников фондового рынка. Наконец, мы применили алгоритм сдвига среднего для сегментирования рынка на основе информации о стереотипах поведения покупателей. В следующей главе вы познакомитесь с движком для создания рекомендательных систем.

5
Создание рекомендательных систем

В этой главе будет показано, как создать собственную рекомендательную систему фильмов. Сначала мы создадим обучающий конвейер, для тренировки которого используются настраиваемые параметры. Далее вы узнаете о том, что собой представляют классификаторы на основе ближайших соседей и как их можно реализовать. Это послужит основой для обсуждения коллаборативной фильтрации, которое завершится построением рекомендательной системы.

К концу главы вы освоите следующие темы:

- создание обучающего конвейера;
- извлечение ближайших соседей;
- создание классификатора методом K ближайших соседей;
- вычисление оценок сходства;
- использование коллаборативной фильтрации для поиска пользователей с похожими предпочтениями;
- создание рекомендательной системы фильмов.

Создание обучающего конвейера

Обычно системы машинного обучения строятся на модульной основе. Конкретная конечная цель достигается за счет формирования подходящих комбинаций отдельных модулей. В библиотеке `scikit-learn` содержатся функции, позволяющие объединять различные модули в единые конвейерные цепочки. Нам остается лишь указать нужные модули вместе с соответствующими параметрами. Далее на основе этих модулей создается конвейер, который обрабатывает данные и тренирует систему.

Конвейер может формироваться из модулей, выполняющих самые разные функции, такие как отбор признаков, предварительная обработка данных, построение случайных лесов, кластеризация и т.п. В этом разделе мы покажем, как создать конвейер, предназначенный для выбора наиболее важных K признаков из входных данных и их последующей классификации с использованием классификатора на основе предельно случайного леса.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from sklearn.datasets import samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
```

Сгенерируем маркированные выборочные данные для процессов обучения и тестирования. Пакет `scikit-learn` включает встроенную функцию, которая справляется с этой задачей. В приведенной ниже строке кода создаются 150 точек данных, каждая из которых представляет собой 25-мерный вектор. Числовые значения в каждом векторе признаков будут генерироваться с использованием генератора случайных выборок. Каждая точка данных включает шесть информативных признаков и не содержит ни одного избыточного. Используем следующий код.

```
# Генерирование данных
X, y = samples_generator.make_classification(n_samples=150,
        n_features=25, n_classes=3, n_informative=6,
        n_redundant=0, random_state=7)
```

Первым блоком этого конвейера является селектор признаков. Этот блок отбирает k "наилучших" признаков. Установим для k значение 9.

```
# Выбор k наиболее важных признаков
k_best_selector = SelectKBest(f_regression, k=9)
```

Следующий блок конвейера — классификатор на основе предельно случайного леса с 60 деревьями и максимальной глубиной, равной четырем. Используем следующий код.

```
# Инициализация классификатора на основе
# предельно случайного леса
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
```

Создадим конвейер посредством объединения описанных блоков. Мы можем присвоить имя каждому блоку, чтобы их было легче отслеживать.

```
# Создание конвейера
processor_pipeline = Pipeline([('selector', k_best_selector),
                               ('erf', classifier)])
```

Параметры отдельных блоков можно изменять. Давайте присвоим значение 7 параметру k в первом блоке и значение 30 количеству деревьев (n_estimators) во втором блоке. Для определения областей видимости переменных мы используем имена, ранее присвоенные блокам.

```
# Установка параметров
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
```

Обучим конвейер, используя сгенерированные перед этим выборочные данные.

```
# Обучение конвейера
processor_pipeline.fit(X, y)
```

Спрогнозируем результаты для всех входных значений и выведем их.

```
# Прогнозирование результатов для входных данных
output = processor_pipeline.predict(X)
print("\nPredicted output:\n", output)
```

Вычислим оценку, используя маркированные тренировочные данные.

```
# Вывод оценки
print("\nScore:", processor_pipeline.score(X, y))
```

Извлечем признаки, отобранные блоком селектора. Мы указали, что хотим выбрать 7 таких признаков из их общего количества 25. Используем следующий код.

```
# Вывод признаков, отобранных селектором конвейера
status = processor_pipeline.named_steps['selector']
                           .get_support()

# Извлечение и вывод индексов выбранных признаков
selected = [i for i, x in enumerate(status) if x]
print("\nIndices of selected features:",
      ', '.join([str(x) for x in selected]))
```

Полный код этого примера содержится в файле pipeline_trainer.py. После выполнения этого кода в окне терминала отобразится следующая информация (рис. 5.1).

```
Predicted output:
 [1 2 2 0 2 2 0 2 0 1 2 0 2 1 0 0 2 2 2 1 0 2 0 1 2 1 1 1 0 0 1 2 1 0 0 0 2
 1 1 0 2 0 0 0 1 2 0 2 1 0 1 0 0 0 2 1 1 1 1 1 0 1 2 2 2 0 2 0 2 2 0 1 2 0
 2 0 2 0 1 0 2 2 1 1 1 2 0 0 0 0 2 2 0 2 1 1 2 0 1 1 2 1 1 0 1 0 2 2 2 0 0
 1 2 1 1 0 2 0 0 0 0 0 2 2 1 1 1 2 0 2 2 1 0 2 0 0 0 1 1 2 2 2 2 2 2 1 1 0
 2 0]

Score: 0.893333333333

Indices of selected features: 13, 15, 18, 19, 21, 23, 24
```

Рис. 5.1

Первый из отображенных на рис. 5.1 списков представляет результирующие метки, спрогнозированные с помощью процессора данных. Значение `Score` характеризует эффективность процессора данных. В последней строке выведены индексы отобранных свойств.

Извлечение ближайших соседей

Для формирования эффективных рекомендаций в рекомендательных системах используется понятие *ближайших соседей* (nearest neighbours), суть которого заключается в нахождении тех точек заданного набора, которые расположены на ближайших расстояниях от указанной. Такой подход часто применяется для создания систем, классифицирующих точку данных на основании ее близости к различным классам. Обратимся к примеру нахождения ближайших соседей заданной точки данных.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
```

Определим выборку двумерных точек данных.

```
# Входные данные
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4],
        [3.8, 0.9], [7.3, 2.1], [4.2, 6.5], [3.8, 3.7],
        [2.5, 4.1], [3.4, 1.9], [5.7, 3.5], [6.1, 4.3],
        [5.1, 2.2], [6.2, 1.1]])
```

Определим количество ближайших соседей, которые хотим извлечь.

```
# Количество ближайших соседей
k = 5
```

Определим тестовую точку данных, для которой будем извлекать k ближайших соседей.

```
# Тестовая точка данных
test_datapoint = [4.3, 2.7]
```

Отобразим на графике входные данные, используя в качестве маркеров черные кружки.

```
# Отображение входных данных на графике
plt.figure()
plt.title('Входные данные')
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')
```

Создадим и обучим модель на основе метода K ближайших соседей, используя входные данные. Применим эту модель для извлечения ближайших соседей нашей тестовой точки данных.

```
# Построение модели на основе метода K ближайших соседей
knn_model = NearestNeighbors(n_neighbors=k,
                             algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors(test_datapoint)
```

Выведем извлеченные из модели точки данных, являющиеся ближайшими соседями.

```
# Выведем 'k' ближайших соседей
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])
```

Визуализируем ближайших соседей.

```
# Визуализация ближайших соседей вместе с
# тестовой точкой данных
plt.figure()
plt.title('Ближайшие соседи')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],
        marker='o', s=250, color='k', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
        marker='x', s=75, color='k')

plt.show()
```

Полный код этого примера содержится в файле `k_nearest_neighbors.py`. В процессе выполнения этого кода на экране отобразятся два графика. Первый экранный снимок представляет входные данные (рис. 5.2).

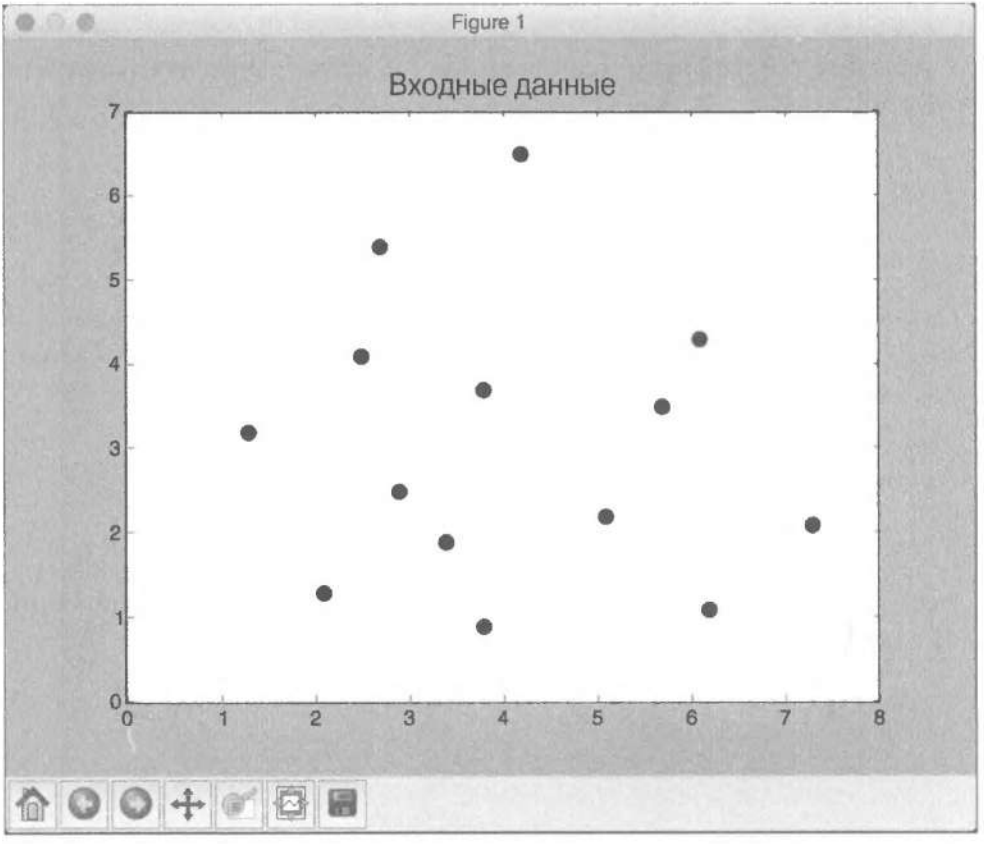

Рис. 5.2

Второй экранный снимок представляет пять ближайших соседей. Тестовая точка данных обозначена крестиком, а ближайшие к ней точки данных обведены окружностями (рис. 5.3).

В окне терминала отобразится следующая информация (рис. 5.4).

На рис. 5.4 представлены пять точек, являющихся ближайшими соседями тестовой точки данных.

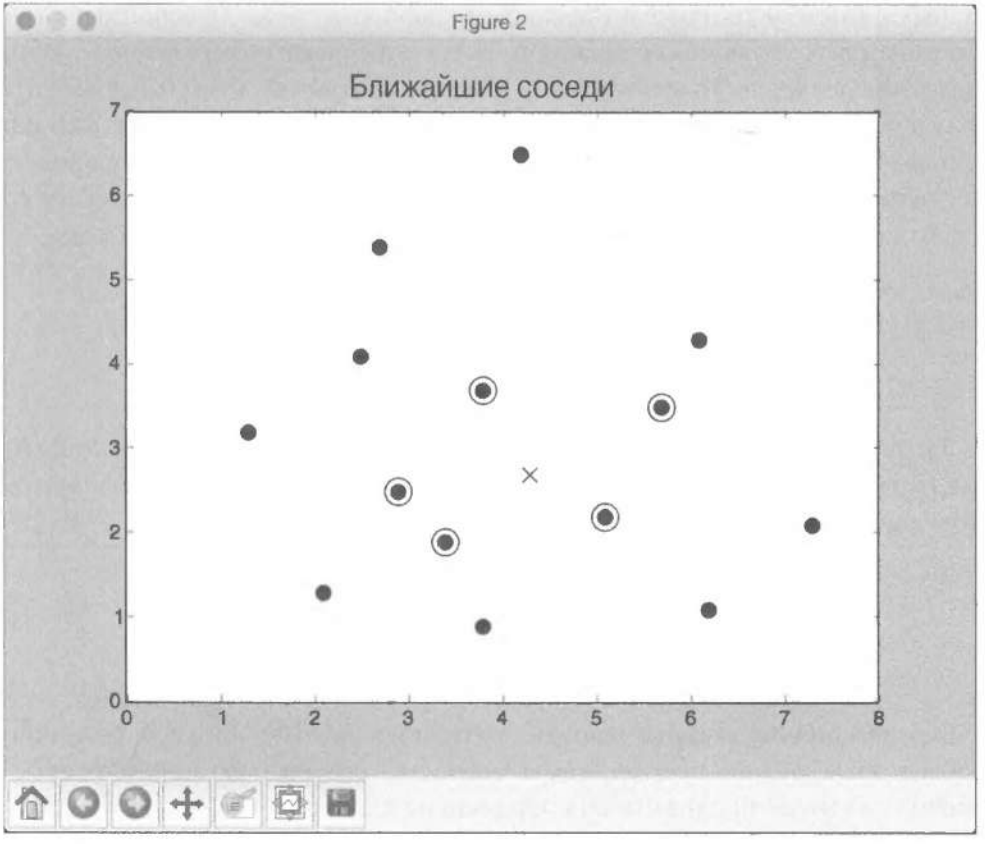

Рис. 5.3

```
K Nearest Neighbors:
1 ==> [ 5.1  2.2]
2 ==> [ 3.8  3.7]
3 ==> [ 3.4  1.9]
4 ==> [ 2.9  2.5]
5 ==> [ 5.7  3.5]
```

Рис. 5.4

Создание классификатора методом K ближайших соседей

Классификатор на основе K ближайших соседей — это модель классификации, в которой заданная точка классифицируется с использованием алгоритма ближайших соседей. Для определения категории входной точки

данных алгоритм находит в обучающем наборе K точек, являющихся ближайшими по отношению к заданной. После этого назначаемый точке данных класс определяется "голосованием". Мы просматриваем классы K элементов полученного списка и выбираем из них тот класс, которому соответствует наибольшее количество "голосов". Рассмотрим пример создания классификатора с использованием этой модели. Значение K зависит от конкретной задачи.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets
```

Загрузим входные данные из файла `data.txt`. Каждая строка этого файла содержит значения, разделенные запятой, причем данные представляют четыре класса.

```
# Загрузка входных данных
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(np.int)
```

Визуализируем входные данные, используя четыре маркера различной формы. Нам нужно преобразовать метки в соответствующие маркеры, и именно для этого предназначена переменная `mapper`.

```
# Отображение входных данных на графике
plt.figure()
plt.title('Входные данные')
marker_shapes = 'v^os'
mapper = [marker_shapes[i] for i in y]
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
```

Определим количество ближайших соседей, которое мы хотим использовать.

```
# Количество ближайших соседей
num_neighbors = 12
```

Определим шаг сетки, которую будем использовать для визуализации границ классификатора.

```
# Шаг сетки визуализации
step_size = 0.01
```

Создадим модель классификатора методом K ближайших соседей.

```
# Создание классификатора на основе метода K ближайших соседей
classifier = neighbors.KNeighborsClassifier(num_neighbors,
weights='distance')
```

Обучим модель, используя тренировочные данные.

```
# Обучение модели на основе метода K ближайших соседей
classifier.fit(X, y)
```

Создадим деления, которые будем использовать для визуализации сетки.

```
# Создание сетки для отображения границ на графике
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_values, y_values = np.meshgrid(np.arange(x_min, x_max,
        step_size), np.arange(y_min, y_max, step_size))
```

Выполним классификатор на всех точках сетки.

```
# Выполнение классификатора на всех точках сетки
output = classifier.predict(np.c_[x_values.ravel(),
        y_values.ravel()])
```

Создадим сетку с цветовым выделением областей для визуализации результата.

```
# Визуализация предсказанного результата
output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)
```

Отобразим обучающие данные поверх цветовой карты для визуализации расположения точек относительно границ.

```
# Наложение обучающих точек на карту
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=50, edgecolors='black', facecolors='none')
```

Зададим предельные значения для осей X и Y и укажем заголовок.

```
plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
plt.title('Границы модели классификатора на основе K
          ближайших соседей')
```

Чтобы оценить эффективность классификатора, определим тестовую точку данных. Отобразим на графике обучающие точки данных вместе с тестовой точкой для визуальной оценки их взаимного расположения.

```
# Тестирование входной точки данных
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Тестовая точка данных')
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')
```

Извлечем K ближайших соседей тестовой точки данных, используя модель классификатора.

```
# Извлечение K ближайших соседей
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(np.int)[0]
```

Отобразим на графике K ближайших соседей, полученных на предыдущем шаге.

```
# Отображение K ближайших соседей на графике
plt.figure()
plt.title('K ближайших соседей')
for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
            linewidth=3, s=100, facecolors='black')
```

Отобразим на том же графике тестовую точку.

```
plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')
```

Отобразим на том же графике входные данные.

```
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
```

Выведем предсказанный результат.

```
print("Predicted output:",
      classifier.predict([test_datapoint])[0])

plt.show()
```

Полный код этого примера содержится в файле `nearest_neighbors_classifier.py`. В процессе выполнения этого кода на экране отобразятся четыре графика. Первый экранный снимок представляет входные данные (рис. 5.5).

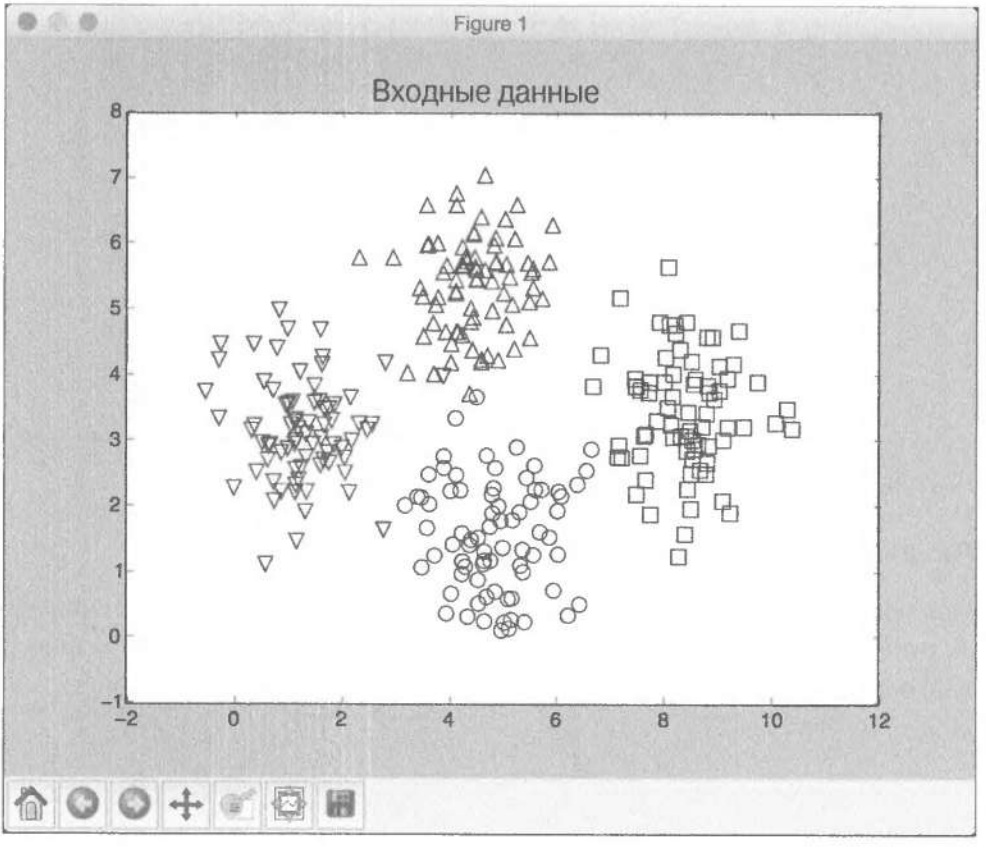

Рис. 5.5

Второй экранный снимок представляет границы классификатора (рис. 5.6).

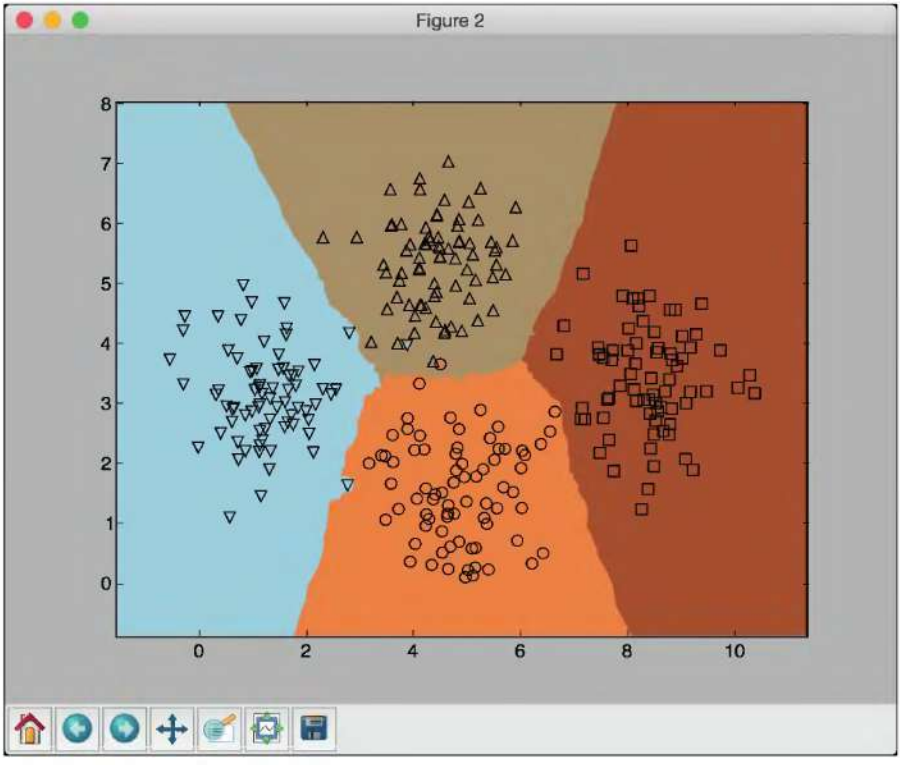

Рис. 5.6. (См. цветную вклейку; адрес указан во введении)

На третьем экранном снимке показано расположение тестовой точки данных относительно точек входного набора. Тестовая точка данных обозначена крестиком (рис. 5.7).

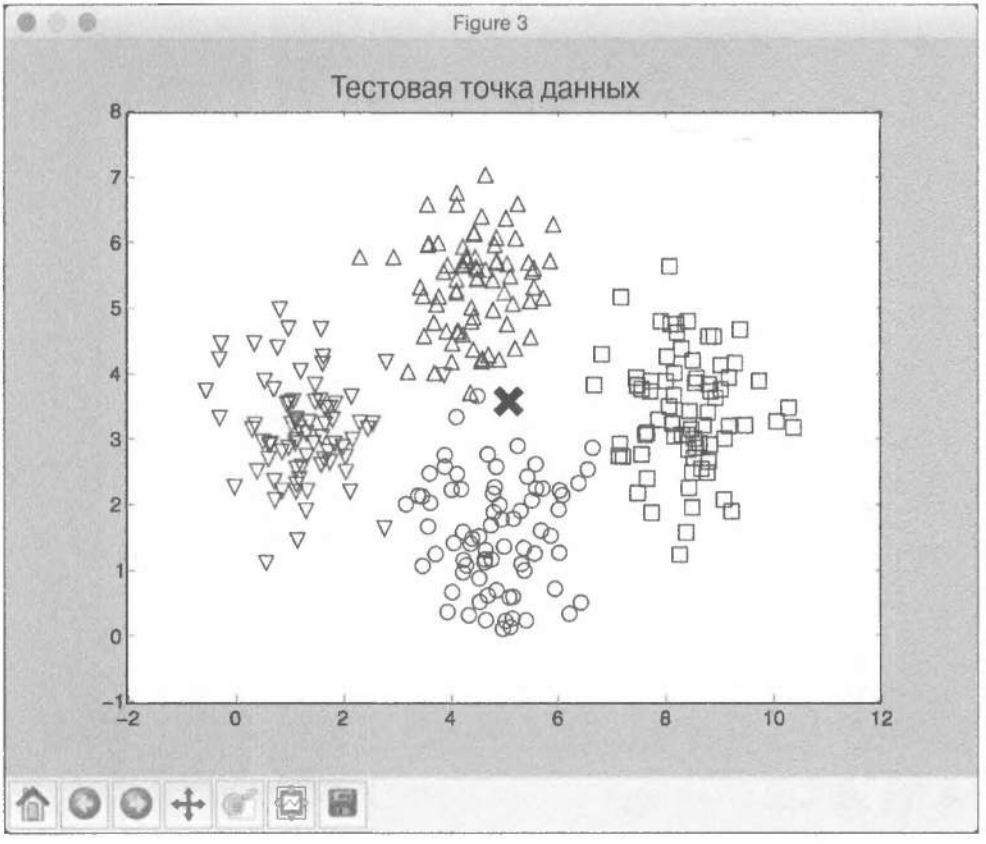

Рис. 5.7

На четвертом экранном снимке сплошными черными маркерами выделены 12 ближайших соседей тестовой точки данных (рис. 5.8).

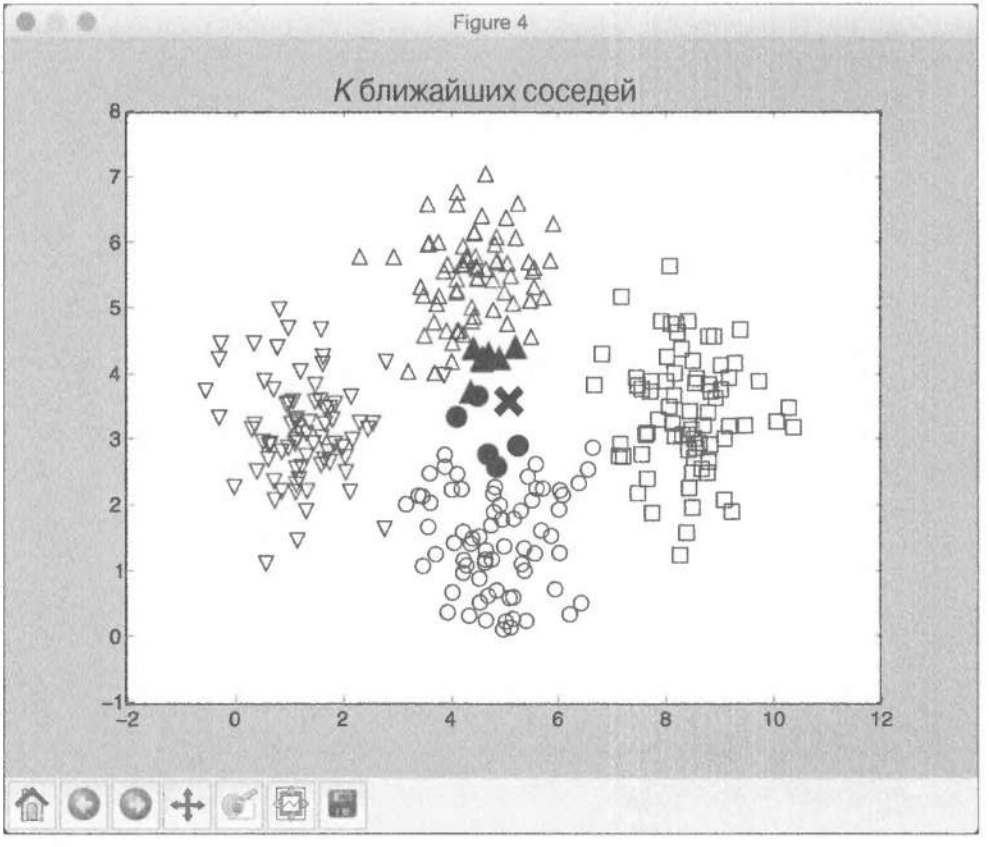

Рис. 5.8

В окне терминала отобразится строка, информирующая о том, что тестовая точка данных относится к классу 1.

```
Predicted output: 1
```

Вычисление оценок сходства

При построении рекомендательных систем очень важную роль играет выбор способа сравнения различных объектов, входящих в набор данных. Предположим, что наш набор данных включает информацию о пользователях и их предпочтениях. Для того чтобы что-то рекомендовать, мы должны понимать, как сравнивать вкусы различных людей. И здесь на первый план выходят *оценки сходства* (similarity scores). Оценка сходства дает представление о том, в какой степени два объекта могут считаться аналогичными друг другу.

Для этой цели часто используют оценки двух типов: евклидовы и по Пирсону. В основу евклидовой оценки положено евклидово расстояние между

двумя точками данных. Если вам необходимо освежить знания относительно того, что такое *евклидово расстояние*, то загляните в Википедию (`https://ru.wikipedia.org/wiki/Евклидова_метрика`). Евклидово расстояние не ограничено по величине. Поэтому мы берем соответствующее значение и преобразуем его таким образом, чтобы новое значение находилось в диапазоне от 0 до 1. Если евклидово расстояние между двумя объектами велико, то соответствующая евклидова оценка должна иметь небольшую величину, поскольку низкая оценка указывает на малую степень сходства между объектами. Следовательно, евклидово расстояние обратно пропорционально евклидовой оценке.

Оценка сходства по Пирсону (Pearson score) — это математическая мера корреляции двух объектов. Для ее вычисления используют *ковариацию* (covariance) двух объектов и их индивидуальные *стандартные отклонения* (standard deviations). Значения этой оценки могут изменяться в пределах от –1 до +1. Оценка +1 указывает на высокую степень сходства объектов, тогда как оценка –1 — на большие различия между ними. Оценка 0 свидетельствует об отсутствии корреляции между двумя объектами. Приступим к вычислению этих оценок.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse
import json
import numpy as np
```

Создадим парсер для обработки входных аргументов. Ими будут служить имена двух пользователей и тип оценки, которая будет использоваться для вычисления степени сходства.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Compute
similarity score')
    parser.add_argument('--user1', dest='user1', required=True,
            help='First user')
    parser.add_argument('--user2', dest='user2', required=True,
            help='Second user')
    parser.add_argument("--score-type", dest="score_type",
            required=True, choices=['Euclidean', 'Pearson'],
            help='Similarity metric to be used')
    return parser
```

Определим функцию, вычисляющую евклидову оценку для двух заданных пользователей. Если информация о пользователях отсутствует в наборе данных, генерируется исключение.

```
# Вычисление оценки евклидова расстояния между
# пользователями user1 и user2
def euclidean_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')
    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')
```

Определим переменную, которая будет использоваться для отслеживания фильмов, получивших рейтинговую оценку от обоих пользователей.

```
    # Фильмы, оцененные обоими пользователями, user1 и user2
    common_movies = {}
```

Извлечем фильмы, получившие рейтинговую оценку от обоих пользователей.

```
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1
```

В случае отсутствия фильмов, оцененных обоими пользователями, вычисление оценки сходства становится невозможным.

```
    # В отсутствие фильмов, оцененных обоими пользователями,
    # оценка принимается равной 0
    if len(common_movies) == 0:
        return 0
```

Вычислим квадрат разности между рейтинговыми оценками и используем его для получения евклидовой оценки.

```
    squared_diff = []

    for item in dataset[user1]:
        if item in dataset[user2]:
            squared_diff.append(np.square(dataset[user1][item] -
                                dataset[user2][item]))
    return 1 / (1 + np.sqrt(np.sum(squared_diff)))
```

Определим функцию, вычисляющую оценку сходства по Пирсону для двух заданных пользователей из числа включенных в набор данных. В случае отсутствия информации об этих пользователях генерируется исключение.

```
# Вычислим коэффициент корреляции Пирсона для user1 и user2
def pearson_score(dataset, user1, user2):
    if user1 not in dataset:
```

```
        raise TypeError('Cannot find ' + user1 + ' in the dataset')
    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')
```

Определим переменную, которая будет использоваться для отслеживания фильмов, получивших рейтинговую оценку от обоих пользователей.

```
# Фильмы, оцененные обоими пользователями, user1 и user2
common_movies = {}
```

Извлечем фильмы, получившие рейтинговую оценку от обоих пользователей.

```
for item in dataset[user1]:
    if item in dataset[user2]:
        common_movies[item] = 1
```

В случае отсутствия фильмов, оцененных обоими пользователями, вычисление оценки сходства становится невозможным.

```
num_ratings = len(common_movies)

# В отсутствие фильмов, оцененных обоими пользователями,
# оценка принимается равной 0
if num_ratings == 0:
    return 0
```

Вычислим сумму рейтинговых оценок всех фильмов, оцененных обоими пользователями.

```
# Вычисление суммы рейтинговых оценок всех фильмов,
# оцененных обоими пользователями
user1_sum = np.sum([dataset[user1][item] for item in
                    common_movies])
user2_sum = np.sum([dataset[user2][item] for item in
                    common_movies])
```

Вычислим сумму квадратов рейтинговых оценок всех фильмов, оцененных обоими пользователями.

```
# Вычисление суммы квадратов рейтинговых оценок всех
# фильмов, оцененных обоими пользователями
user1_squared_sum = np.sum([np.square(dataset[user1][item])
                            for item in common_movies])
user2_squared_sum = np.sum([np.square(dataset[user2][item])
                            for item in common_movies])
```

Вычислим сумму произведений рейтинговых оценок всех фильмов, оцененных обоими пользователями.

```
    # Вычисление суммы произведений рейтинговых оценок всех
    # фильмов, оцененных обоими пользователями
    sum_of_products = np.sum([dataset[user1][item] *
            dataset[user2][item] for item in common_movies])
```

Вычислим различные параметры, требуемые для вычисления оценки сходства по Пирсону с использованием результатов предыдущих вычислений.

```
    # Вычисление коэффициента корреляции Пирсона
    Sxy = sum_of_products - (user1_sum * user2_sum /
          num_ratings)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
    Syy = user2_squared_sum - np.square(user2_sum) / num_ratings
```

В отсутствие отклонения оценка равна 0.

```
    if Sxx * Syy == 0:
        return 0
```

Возвращаем значение оценки по Пирсону.

```
    return Sxy / np.sqrt(Sxx * Syy)
```

Определим основную функцию и разберем входные аргументы.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
    user2 = args.user2
    score_type = args.score_type
```

Загрузим рейтинговые оценки из файла `ratings.json` в словарь.

```
    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())
```

Вычислим оценку сходства на основании входных аргументов.

```
    if score_type == 'Euclidean':
        print("\nEuclidean score:")
        print(euclidean_score(data, user1, user2))
    else:
```

```
print("\nPearson score:")
print(pearson_score(data, user1, user2))
```

Полный код этого примера содержится в файле `compute_scores.py`. Выполним этот код для некоторых комбинаций входных данных. Предположим, мы хотим вычислить евклидову оценку сходства пользователей `David Smith` и `Bill Duffy`.

```
$ python3 compute_scores.py --user1 "David Smith" --user2
"Bill Duffy" --score-type Euclidean
```

После выполнения приведенной выше команды в окне терминала отобразится следующая информация.

```
Euclidean score:
0.585786437627
```

Чтобы вычислить оценку сходства по Пирсону для той же пары пользователей, выполните следующую команду.

```
$ python3 compute_scores.py --user1 "David Smith" --user2
"Bill Duffy" --score-type Pearson
```

В окне терминала отобразится следующая информация.

```
Pearson score:
0.99099243041
```

Можете поэкспериментировать, выполняя аналогичные команды с использованием других сочетаний параметров.

Поиск пользователей с похожими предпочтениями методом коллаборативной фильтрации

Термин *коллаборативная фильтрация* (collaborative filtering) относится к процессу идентификации шаблонов поведения объектов набора данных с целью принятия решений относительно нового объекта. В контексте рекомендательных систем метод коллаборативной фильтрации используют для прогнозирования предпочтений нового пользователя на основании имеющейся информации о предпочтениях других пользователей с аналогичными вкусами.

Составляя прогнозы для предпочтений индивидуальных пользователей, мы используем имеющуюся совместную информацию о предпочтениях других пользователей. Именно поэтому данный метод фильтрации называется *коллаборативным*.

В данном случае основное допущение заключается в том, что если два человека дают одинаковые рейтинговые оценки некоторому набору фильмов, то их оценки фильмов из неизвестного набора также будут примерно одинаковыми. Находя общие оценочные суждения в отношении одних фильмов, мы можем прогнозировать оценки в отношении других фильмов. Из предыдущего раздела вы узнали о том, как сравнивать между собой различных пользователей в пределах одного набора данных. Мы задействуем эти методики оценки сходства для поиска пользователей с похожими предпочтениями в нашем наборе данных. Как правило, коллаборативную фильтрацию применяют, когда имеют дело с наборами данных большого размера. Методы такого типа можно использовать в самых разных областях, включая финансовый анализ, онлайн-покупки, маркетинговые исследования, изучение покупательских привычек и т.п.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse
import json
import numpy as np

from compute_scores import pearson_score
```

Определим функцию для парсинга входных аргументов. В данном случае единственным входным аргументом является имя пользователя.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description=
            'Find users who are similar to the input user ')
    parser.add_argument('--user', dest='user', required=True,
            help='Input user')
    return parser
```

Определим функцию, которая будет находить в наборе данных пользователей, аналогичных указанному. Если информация об указанном пользователе отсутствует, генерируется исключение.

```
# Поиск в наборе данных пользователей, аналогичных указанному
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')
```

Мы уже импортировали функцию, необходимую для вычисления оценки сходства по Пирсону. Вычислим с ее помощью оценку сходства по Пирсону между указанным пользователем и всеми остальными пользователями в наборе данных.

```
    # Вычисление оценки сходства по Пирсону между
    # указанным пользователем и всеми остальными
    # пользователями в наборе данных
    scores = np.array([[x, pearson_score(dataset, user,
            x)] for x in dataset if x != user])
```

Отсортируем оценки по убыванию.

```
    # Сортировка оценок по убыванию
    scores_sorted = np.argsort(scores[:, 1])[::-1]
```

Извлечем первых `num_users` пользователей и вернем массив.

```
    # Извлечение оценок первых 'num_users' пользователей
    top_users = scores_sorted[:num_users]
    return scores[top_users]
```

Определим основную функцию и разберем входные аргументы, чтобы извлечь имя пользователя.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user = args.user
```

Загрузим данные из файла `ratings.json`, в котором содержатся имена пользователей и рейтинговые оценки фильмов.

```
    ratings_file = 'ratings.json'

with open(ratings_file, 'r') as f:
    data = json.loads(f.read())
```

Найдем первых трех пользователей, аналогичных пользователю, указанному с помощью входного аргумента. Если хотите, можете указать в качестве аргумента другое количество пользователей. Выведем имена пользователей вместе с оценками сходства.

```
print('\nUsers similar to ' + user + ':\n')
similar_users = find_similar_users(data, user, 3)
print('User\t\t\tSimilarity score')
print('-'*41)
for item in similar_users:
    print(item[0], '\t\t', round(float(item[1]), 2))
```

Полный код этого примера содержится в файле `collaborative_filtering.py`. Выполним этот код и найдем пользователей, аналогичных пользователю `Bill Duffy`.

```
$ python3 collaborative_filtering.py --user "Bill Duffy"
```

В окне терминала отобразится следующая информация (рис. 5.9).

```
Users similar to Bill Duffy:

User                    Similarity score
-----------------------------------------
David Smith              0.99
Samuel Miller            0.88
Adam Cohen               0.86
```

Рис. 5.9

Выполним тот же код и найдем пользователей, аналогичных пользователю `Clarissa Jackson`.

```
$ python3 collaborative_filtering.py --user "Clarissa Jackson"
```

На этот раз в окне терминала отобразится следующая информация (рис. 5.10).

```
Users similar to Clarissa Jackson:

User                    Similarity score
-----------------------------------------
Chris Duncan             1.0
Bill Duffy               0.83
Samuel Miller            0.73
```

Рис. 5.10

Создание рекомендательной системы фильмов

Теперь, когда мы располагаем всеми необходимыми строительными кирпичиками, можно перейти к созданию рекомендательной системы фильмов. Со всеми концепциями, которые при этом будут использоваться, вы уже знакомы. В этом разделе мы создадим рекомендательную систему фильмов на

основании данных, предоставленных в файле `ratings.json`. В этом файле содержится информация о пользователях и оценках, данных ими различным фильмам. Чтобы рекомендовать фильмы конкретному пользователю, мы должны найти аналогичных ему пользователей в имеющемся наборе данных и использовать информацию об их предпочтениях для формирования соответствующей рекомендации.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse
import json
import numpy as np

from compute_scores import pearson_score
from collaborative_filtering import find_similar_users
```

Определим функцию для парсинга входных аргументов. В нашем случае единственным входным аргументом является имя пользователя.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find the movie
             recommendations for the given user')
    parser.add_argument('--user', dest='user', required=True,
           help='Input user')
    return parser
```

Определим функцию, которая будет получать рекомендации для указанного пользователя. Если информация об указанном пользователе отсутствует в наборе данных, генерируется исключение.

```
# Получить рекомендации относительно фильмов
# для указанного пользователя
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')
```

Определим переменные для отслеживания оценок.

```
    overall_scores = {}
    similarity_scores = {}
```

Вычислим оценку сходства между указанным пользователем и всеми остальными пользователями в наборе данных.

```
    for user in [x for x in dataset if x != input_user]:
        similarity_score = pearson_score(dataset, input_user, user)
```

Если оценка сходства меньше 0, переходим к следующему пользователю.

```
        if similarity_score <= 0:
            continue
```

Извлечем список фильмов, уже получивших рейтинговую оценку от текущего пользователя, но еще не оцененных указанным пользователем.

```
    filtered_list = [x for x in dataset[user] if x not in \
        dataset[input_user] or dataset[input_user][x] == 0]
```

Отследим взвешенную рейтинговую оценку для каждого элемента отфильтрованного списка, исходя из оценок сходства. Также отследим оценки сходства.

```
        for item in filtered_list:
            overall_scores.update({item: dataset[user][item] *
                                   similarity_score})
            similarity_scores.update({item: similarity_score})
```

В случае отсутствия подходящих фильмов мы не можем предоставить никаких рекомендаций.

```
    if len(overall_scores) == 0:
        return ['No recommendations possible']
```

Нормализуем оценки на основании взвешенных оценок.

```
    # Генерация рейтингов фильмов посредством их нормализации
    movie_scores = np.array([[score/similarity_scores[item],
            item] for item, score in overall_scores.items()])
```

Выполним сортировку оценок и извлечем рекомендации фильмов.

```
    # Сортировка по убыванию
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]
    # Извлечение рекомендаций фильмов
    movie_recommendations = [movie for _, movie in movie_scores]

    return movie_recommendations
```

Определим основную функцию и проанализируем входные аргументы, чтобы извлечь имя указанного пользователя.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user = args.user
```

Загрузим данные о рейтинге фильмов из файла `ratings.json`.

```
ratings_file = 'ratings.json'

with open(ratings_file, 'r') as f:
    data = json.loads(f.read())
```

Извлечем рекомендации фильмов и выведем результаты.

```
print("\nMovie recommendations for " + user + ":")
movies = get_recommendations(data, user)
    for i, movie in enumerate(movies):
        print(str(i+1) + '. ' + movie)
```

Полный код этого примера содержится в файле `movie_recommender.py`. Найдем рекомендации фильмов для пользователя `Chris Duncan`.

```
$ python3 movie_recommender.py --user "Chris Duncan"
```

В окне терминала отобразится следующая информация (рис. 5.11).

```
Movie recommendations for Chris Duncan:
1. Vertigo
2. Goodfellas
3. Scarface
4. Roman Holiday
```

Рис. 5.11

А теперь найдем рекомендации фильмов для пользователя `Julie Hammel`.

```
$ python3 movie_recommender.py --user "Julie Hammel"
```

На этот раз в окне терминала отобразится следующая информация (рис. 5.12).

```
Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
```

Рис. 5.12

Резюме

В этой главе рассказывалось о том, как создать конвейерный процессор данных, который можно задействовать для тренировки системы машинного обучения. Вы также узнали, как извлечь K ближайших соседей любой точки из заданного набора. Затем мы использовали эти сведения для создания классификатора на основе метода K ближайших соседей. Мы также обсудили такие способы вычисления оценок сходства, как евклидова оценка и оценка по Пирсону. Кроме того, вы узнали о том, как использовать коллаборативную фильтрацию для нахождения пользователей с одинаковыми предпочтениями в заданном наборе данных и создать рекомендательную систему фильмов на основе этой информации.

В следующей главе будет рассказано о том, что такое логическое программирование и как создать систему логического вывода, способную решать реальные задачи.

6
Логическое программирование

В этой главе вы научитесь писать программы, применяя методы логического программирования. Мы обсудим различные парадигмы программирования и покажем, как создавать программы, используя теорию и аппарат математической логики. Мы рассмотрим конструкции логического программирования и способы решения задач в этой области, а также напишем программы на языке Python, реализующие решатели для ряда задач.

К концу главы вы освоите следующие темы:

- что такое логическое программирование;
- конструкции логического программирования;
- решение задач с помощью логического программирования;
- установка пакетов Python;
- сопоставление математических выражений;
- проверка простых чисел;
- анализ генеалогических деревьев;
- анализ географических данных;
- создание решателя головоломок.

Что такое логическое программирование

Логическое программирование — это парадигма, т.е. особый подход к программированию. Прежде чем обсуждать, что собой представляет логическое программирование и какое отношение оно имеет к искусственному интеллекту, следует сказать несколько слов о парадигмах программирования.

Своим возникновением понятие *парадигма программирования* обязано потребности классифицировать языки программирования. Оно характеризует подход, который используется компьютерными программами для решения

задач посредством программного кода. В одних парадигмах главную роль играют условные связки (импликации) и последовательности операций, используемых для достижения результата. В других центральное место отводится способам организации кода.

Ниже перечислены некоторые наиболее популярные парадигмы программирования.

- **Императивное программирование.** Выполнение инструкций приводит к изменению состояния программы, что может сопровождаться побочными эффектами.
- **Функциональное программирование.** Вычислительный процесс трактуется как вычисление значения математических функций, причем изменяемость состояний программы или данных не допускается.
- **Декларативное программирование.** Процесс написания программ сводится к описанию желаемых действий и способов их выполнения. Базовая логика выполнения программы выражается без явного описания потока управления.
- **Объектно-ориентированное программирование.** Код программы группируется таким образом, чтобы каждый объект самостоятельно отвечал за свое поведение. Объекты содержат данные и методы, определяющие способ изменения данных.
- **Процедурное программирование.** Код группируется в функции, и каждая функция самостоятельно отвечает за выполнение конкретной последовательности шагов.
- **Символическое программирование.** Используется особый стиль синтаксиса и грамматики, посредством которого программа может изменять собственные компоненты, трактуя их как простые данные.
- **Логическое программирование.** Вычисления рассматриваются как автоматизированный процесс получения логических выводов с использованием базы знаний, состоящей из фактов и правил.

Овладение методами логического программирования требует понимания как *вычислительных приемов* (computation), так и *приемов логического вывода* (deduction). Чтобы что-то вычислить, мы исходим из выражения и набора правил. По сути, этот набор правил и представляет программу.

Выражения и правила используются для получения результата. Предположим, мы хотим вычислить сумму чисел 23, 12 и 49 (рис. 6.1).

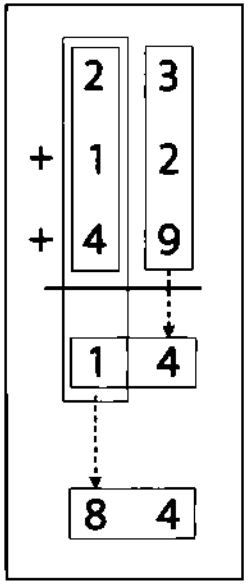

Рис. 6.1

Соответствующая процедура может выглядеть так:

23 + 12 + 49 => (2 + 1 + 4 + 1) 4 => 84

С другой стороны, если мы хотим вывести умозаключение, то должны исходить из некоторой гипотезы. Затем мы должны построить доказательство в соответствии с заданным набором правил. В сущности, вычисления — это механический процесс, тогда как процесс логического вывода носит более творческий характер.

Создавая программу в рамках парадигмы логического программирования, мы определяем набор утверждений, основанных на фактах и правилах, относящихся к предметной области задачи, а решатель задач находит решение, используя эту информацию.

Конструкции логического программирования

В рамках парадигм объектно-ориентированного или императивного программирования мы всегда должны указывать, как определяется переменная. В логическом программировании все работает немного иначе. Мы можем передать функции неинстанциализированные аргументы, и интерпретатор самостоятельно инстанциализирует их, просматривая факты, определенные пользователем. Это весьма мощный способ решения проблемы, связанной с установлением соответствия переменных. Процесс сопоставления переменных различным элементам называется *унификацией*. Это один из тех

моментов, которые ставят логическое программирование несколько обособленно от других парадигм. В логическом программировании мы обязаны специфицировать *отношения*. Эти отношения определяются посредством предложений, называемых *фактами* и *правилами*.

Факты — это всего лишь истинные утверждения о нашей программе и данных, которыми она оперирует. Соответствующий синтаксис довольно прост. Например, утверждение "Дональд — сын Алана" может быть фактом, тогда как вопрос "Кто является сыном Алана?" таковым быть не может. Любая логическая программа нуждается в фактах, с которыми она могла бы работать и используя которые могла бы достигнуть цели.

Правила — это положения, касающиеся того, как мы выражаем различные факты и запрашиваем их. Они представляют собой ограничения, с которыми мы должны работать, и позволяют нам делать заключения в рамках предметной области задачи. Предположим, вы работаете над созданием программы для игры в шахматы. Вы должны установить все правила, определяющие возможные перемещения каждой фигуры на шахматной доске. По сути, окончательное заключение является верным, лишь если все отношения являются истинными.

Решение задач с помощью логического программирования

В логическом программировании решения ищут с помощью фактов и правил. Для каждой программы должна быть определена цель. В тех случаях, когда программа и цель не содержат никаких переменных, в игру вступает решатель задач с деревом, которое образует пространство поиска для решения задачи и достижения цели.

В логическом программировании одним из наиболее важных факторов является способ обработки правил. Правила могут рассматриваться как логические утверждения. Рассмотрим следующее правило:

Катя любит шоколад => Александр любит Катю

Это правило можно прочитать как логический вывод, который гласит: *если Катя любит шоколад, то Александр любит Катю*. То же самое правило можно толковать так: *из того факта, что Катя любит шоколад, следует, что Александр любит Катю*. Аналогичным образом рассмотрим следующее правило:

Криминальные фильмы, английский => Мартин Скорсезе

Его можно прочитать как следующий логический вывод: *если вы любите фильмы криминальной тематики на английском языке, то вам понравятся фильмы Мартина Скорсезе*.

Такая конструкция используется в различных формах во всем логическом программировании для решения различных типов задач. Давайте рассмотрим, как решать подобные задачи с помощью языка Python.

Установка пакетов Python

Прежде чем приступить к логическому программированию на языке Python, установим некоторые пакеты. В Python возможности логического программирования предоставляет пакет `logpy`. Для тех же целей мы будем использовать также пакет `sympy`. Поэтому установим пакеты `logpy` и `sympy` с помощью программы `pip`.

```
$ pip3 install logpy
$ pip3 install sympy
```

Если в процессе установки пакета `logpy` вы получите сообщение об ошибке, можете выполнить установку, используя адрес `https://github.com/logpy/logpy` в качестве источника. Успешно установив оба пакета, переходите к чтению следующего раздела.

Сопоставление математических выражений

С математическими операциями мы сталкиваемся на каждом шагу. Логическое программирование представляет собой весьма эффективный способ сравнения выражений и нахождения неизвестных значений. Рассмотрим, как это делается.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from logpy import run, var, fact
import logpy.assoccomm as la
```

Определим пару математических операций.

```
# Определение математических операций
add = 'addition'
mul = 'multiplication'
```

И сложение, и умножение — коммутативные операции. Укажем это.

```
# Объявление того, что операции коммутативные,
# посредством использования системы фактов
fact(la.commutative, mul)
fact(la.commutative, add)
```

```
fact(la.associative, mul)
fact(la.associative, add)
```

Определим некоторые переменные.

```
# Определение переменных
a, b, c = var('a'), var('b'), var('c')
```

Теперь рассмотрим следующее выражение:

```
expression_orig = 3 x (-2) + (1 + 2 x 3) x (-1)
```

Давайте сгенерируем это выражение с помощью *маскированных переменных* (masked variables). Первое выражение могло бы быть таким:

- *expression1 = (1 + 2 x a) x b + 3 x c*

Второе выражение могло бы быть таким:

- *expression2 = c x 3 + b x (2 x a + 1)*

Третье выражение могло бы быть таким:

- *expression3 = (((2 x a) x b) + b) + 3 x c*

Внимательно присмотревшись, можно заметить, что все три выражения представляют одно и то же базовое выражение. Наша цель состоит в том, чтобы увязать эти выражения с первоначальным выражением для извлечения неизвестных значений.

```
# Генерация выражений
expression_orig = (add, (mul, 3, -2), (mul, (add, 1, (mul, 2, 3)), -1))
expression1 = (add, (mul, (add, 1, (mul, 2, a)), b), (mul, 3, c))
expression2 = (add, (mul, c, 3), (mul, b, (add, (mul, 2, a), 1)))
expression3 = (add, (add, (mul, (mul, 2, a), b), b), (mul, 3, c))
```

Сравним эти выражения с исходным. В пакете `logpy` для этой цели обычно используется метод `run`. Этот метод получает входные аргументы и выполняет выражение. Первый аргумент — количество значений, второй — переменная, третий — функция.

```
# Сравнение выражений
print(run(0, (a, b, c), la.eq_assoccomm(expression1, expression_orig)))
print(run(0, (a, b, c), la.eq_assoccomm(expression2, expression_orig)))
print(run(0, (a, b, c), la.eq_assoccomm(expression3, expression_orig)))
```

Полный код этого примера содержится в файле `expression_matcher.py`. После выполнения этого кода в окне терминала должен отобразиться следующий результат.

```
((3, -1, -2),)
((3, -1, -2),)
()
```

Три числа в первых двух строках представляют значения a, b и c. Первые два выражения соответствуют первоначальному выражению, тогда как третье ничего не возвращает. Это произошло потому, что даже если с точки зрения математики третье выражение аналогично первым двум, оно структурно отличается от них. Шаблон сравнения работает, сравнивая структуры выражений.

Проверка простых чисел

Рассмотрим, как использовать логическое программирование для того, чтобы проверить, что число является простым. Для определения того, какие числа в заданном списке являются простыми, а также для выяснения того, является ли данное число простым, мы используем конструкции, доступные в пакете `logpy`.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import itertools as it
import logpy.core as lc
from sympy.ntheory.generate import prime, isprime
```

Определим функцию, которая проверяет, является ли данный элемент простым числом, в зависимости от типа данных. Если этот элемент — число, то ответ находится непосредственно. Если же данный элемент является переменной, то мы должны выполнить операции над последовательностью. Метод `conde`, являющийся конструктором целей, предоставляет логические операции И (AND) и ИЛИ (OR). Мы воспользуемся методом `condeseq`, который аналогичен методу `conde`, но поддерживает универсальный *итератор целей* (iterator of goals).

```
# Проверка того, являются ли элементы x простыми числами
def check_prime(x):
    if lc.isvar(x):
        return lc.condeseq([(lc.eq, x, p)] for p in map(prime,
                           it.count(1)))
    else:
        return lc.success if isprime(x) else lc.fail
```

Объявим переменную x, которую будем использовать.

```
# Объявление переменной
x = lc.var()
```

Определим набор чисел и проверим, какие из них являются простыми. Метод membero проверяет, является ли данное число элементом списка чисел, указанного во входном аргументе.

```
# Проверка того, является ли элемент списка простым числом
list_nums = (23, 4, 27, 17, 13, 10, 21, 29, 3, 32, 11, 19)
print('\nList of primes in the list:')
print(set(lc.run(0, x, (lc.membero, x, list_nums),
      (check_prime, x))))
```

А теперь используем эту функцию немного иначе, выведя первые 7 простых чисел.

```
# Вывести первые 7 простых чисел
print('\nList of first 7 prime numbers:')
print(lc.run(7, x, check_prime(x)))
```

Полный код этого примера содержится в файле prime.py. После выполнения этого кода в окне терминала отобразится следующий результат.

```
List of primes in the list:
{3, 11, 13, 17, 19, 23, 29}
List of first 7 prime numbers:
(2, 3, 5, 7, 11, 13, 17)
```

Вы можете убедиться в том, что результирующие значения корректны.

Парсинг генеалогического дерева

Теперь, когда вы уже знаете о принципах логического программирования, мы можем применить его для решения одной интересной задачи. Рассмотрим следующее генеалогическое дерево (рис. 6.2).

У Джона (John) и Меган (Megan) три сына: Уильям (William), Дэвид (David) и Адам (Adam). Их жен зовут Эмма (Emma), Оливия (Olivia) и Лили (Lily) соответственно. У Уильяма и Эммы двое детей: Крис (Chris) и Стефани (Stephanie). У Дэвида и Оливии пять детей, которых зовут Уэйн (Wayne), Тиффани (Tiffany), Джули (Julie), Нил (Neil) и Питер (Peter). У Адама и Лили одна дочка, которую зовут София (Sophia). Исходя из этих фактов мы можем написать программу, которая сообщит нам имя дедушки Уэйна или имена дядюшек и тетушек Софии. Несмотря на то что мы не указали, кто кому является дедушкой (бабушкой) или дядей (тетей), логическое программирование позволяет сделать соответствующие заключения.

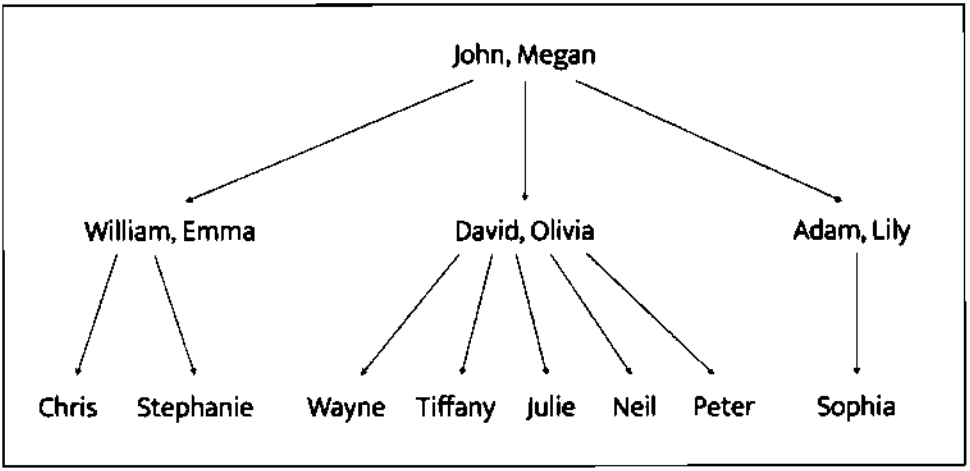

Рис. 6.2

Соответствующие отношения родства определены в файле `relationships.json`. Его содержимое имеет следующий вид.

```
{
     "father":
     [
          {"John": "William"},
          {"John": "David"},
          {"John": "Adam"},
          {"William": "Chris"},
          {"William": "Stephanie"},
          {"David": "Wayne"},
          {"David": "Tiffany"},
          {"David": "Julie"},
          {"David": "Neil"},
          {"David": "Peter"},
          {"Adam": "Sophia"}
     ],
     "mother":
     [
          {"Megan": "William"},
          {"Megan": "David"},
          {"Megan": "Adam"},
          {"Emma": "Stephanie"},
          {"Emma": "Chris"},
          {"Olivia": "Tiffany"},
          {"Olivia": "Julie"},
          {"Olivia": "Neil"},
```

```
            {"Olivia": "Peter"},
            {"Lily": "Sophia"}
        ]
}
```

Это простой файл в формате *json*, в котором указаны лишь отцовские и материнские родственные связи. Заметьте, что в файле отсутствует информация о том, кто кому является мужем, женой, дедушкой, бабушкой, дядей или тетей.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import json
from logpy import Relation, facts, run, conde, var, eq
```

Определим функцию, которая проверяет, является ли объект x родительским по отношению к объекту y. Мы будем использовать следующую логику: если x — родитель y, то x является либо отцом, либо матерью. "Отцы" и "матери" уже определены в нашей базе фактов.

```
# Проверка того, является ли 'x' родителем 'y'
def parent(x, y):
    return conde([father(x, y)], [mother(x, y)])
```

Определим функцию, которая проверяет, является ли x дедушкой (бабушкой) y. Мы будем использовать следующую логику: если x — дедушка (бабушка) y, то непосредственный потомок x является родителем y.

```
# Проверка того, является ли 'x' дедушкой (бабушкой) 'y'
def grandparent(x, y):
    temp = var()
    return conde((parent(x, temp), parent(temp, y)))
```

Определим функцию, которая проверяет, является ли x братом (сестрой) y. Мы будем использовать следующую логику: если x — брат или сестра y, то x и y имеют одних и тех же родителей. Заметьте, что в данном случае требуется незначительное видоизменение кода, поскольку при выводе списка всех родственников того же уровня, что и x, сам x также будет входить в этот список, поскольку он удовлетворяет данным условиям. Поэтому при выводе результата мы должны исключить x из списка. Мы еще вернемся к этому моменту, когда будем обсуждать основную функцию.

```
# Проверка братских (сестринских) отношений между 'a' и 'b'
def sibling(x, y):
    temp = var()
    return conde((parent(temp, x), parent(temp, y)))
```

Определим функцию, которая проверяет, является ли x дядей (тетей) y. Мы будем использовать следующую логику: если x — дядя (тетя) y, то родители x являются дедушкой и бабушкой y. Заметьте, что в данном случае требуется незначительное видоизменение кода, поскольку при выводе списка всех дядюшек (тетушек) y отец y также будет входить в этот список, потому что он удовлетворяет данным условиям. Поэтому при выводе результата мы должны исключить отца y из списка. Мы еще вернемся к этому моменту, когда будем обсуждать основную функцию.

```
# Проверка того, является ли x дядей y
def uncle(x, y):
    temp = var()
    return conde((father(temp, x), grandparent(temp, y)))
```

Определим основную функцию и инициализируем родственные отношения "отец" (father) и "мать" (mother).

```
if __name__=='__main__':
    father = Relation()
    mother = Relation()
```

Загрузим данные из файла relationships.json.

```
    with open('relationships.json') as f:
        d = json.loads(f.read())
```

Прочитаем данные и добавим их в базу фактов.

```
    for item in d['father']:
        facts(father, (list(item.keys())[0],
              list(item.values())[0]))
    for item in d['mother']:
        facts(mother, (list(item.keys())[0],
              list(item.values())[0]))
```

Определим переменную x.

```
    x = var()
```

Теперь мы готовы к тому, чтобы задать системе ряд вопросов и проверить, удается ли нашему решателю задач дать на них правильные ответы. Давайте спросим, как зовут детей Джона.

```
    # Дети Джона
    name = 'John'
    output = run(0, x, father(name, x))
```

```
print("\nList of " + name + "'s children:")
for item in output:
    print(item)
```

Кто мать Уильяма?

```
# Мать Уильяма
name = 'William'
output = run(0, x, mother(x, name))[0]
print("\n" + name + "'s mother:\n" + output)
```

Кто родители Адама?

```
# Родители Адама
name = 'Adam'
output = run(0, x, parent(x, name))
print("\nList of " + name + "'s parents:")
for item in output:
    print(item)
```

Кто дедушка и бабушка Уэйна?

```
# Дедушка и бабушка Уэйна
name = 'Wayne'
output = run(0, x, grandparent(x, name))
print("\nList of " + name + "'s grandparents:")
for item in output:
    print(item)
```

Кто внуки и внучки Меган?

```
# Внуки и внучки Меган
name = 'Megan'
output = run(0, x, grandparent(name, x))
print("\nList of " + name + "'s grandchildren:")
for item in output:
    print(item)
```

Кто братья и сестры Дэвида?

```
# Братья и сестры Дэвида
name = 'David'
output = run(0, x, sibling(x, name))
siblings = [x for x in output if x != name]
print("\nList of " + name + "'s siblings:")
for item in siblings:
    print(item)
```

Кто дяди Тиффани?

```
# Дяди Тиффани
name = 'Tiffany'
name_father = run(0, x, father(x, name))[0]
output = run(0, x, uncle(x, name))
output = [x for x in output if x != name_father]
print("\nList of " + name + "'s uncles:")
for item in output:
    print(item)
```

Выведем список всех супругов в данном семействе.

```
# Все супруги
a, b, c = var(), var(), var()
output = run(0, (a, b), (father, a, c), (mother, b, c))
print("\nList of all spouses:")
for item in output:
    print('Husband:', item[0], '<==> Wife:', item[1])
```

Полный код этого примера содержится в файле `family.py`. После выполнения этого кода в окне терминала отобразится большой объем информации. Ее первая половина будет выглядеть так (рис. 6.3).

```
List of John's children:
David
William
Adam

William's mother:
Megan

List of Adam's parents:
John
Megan

List of Wayne's grandparents:
John
Megan
```

Рис. 6.3

Вторая половина будет иметь следующий вид (рис. 6.4).

```
List of Megan's grandchildren:
Chris
Sophia
Peter
Stephanie
Julie
Tiffany
Neil
Wayne

List of David's siblings:
William
Adam

List of Tiffany's uncles:
William
Adam

List of all spouses:
Husband: Adam <==> Wife: Lily
Husband: David <==> Wife: Olivia
Husband: John <==> Wife: Megan
Husband: William <==> Wife: Emma
```

Рис. 6.4

Вы можете сравнить этот результат с генеалогическим деревом и убедиться в том, что все ответы являются корректными.

Анализ географических данных

Давайте применим методы логического программирования для создания решателя задач, предназначенного для анализа географических данных. В этой задаче мы будем указывать информацию о географическом положении различных штатов США, а затем задавать программе различные вопросы, ответы на которые будут базироваться на этих фактах и правилах. На рис. 6.5 представлена карта США.

Вам предоставляются два текстовых файла: `adjacent_states.txt` и `coastal_states.txt`. В них содержатся подробные данные о том, какие штаты являются смежными по отношению друг к другу, а какие — прибрежными. На основании этих данных мы можем получить интересную информацию.

Например, мы можем выяснить, какие штаты соседствуют одновременно со штатами Оклахома и Техас или какие прибрежные штаты соседствуют как со штатом Нью-Мексико, так и со штатом Луизиана.

Рис. 6.5

Создайте новый файл Python и импортируйте следующие пакеты.

```
from logpy import run, fact, eq, Relation, var
```

Инициализируем отношения.

```
adjacent = Relation()
coastal = Relation()
```

Определим входные файлы, из которых будем загружать данные.

```
file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'
```

Загрузим данные.

```
# Чтение файла, содержащего данные о прибрежных штатах
with open(file_coastal, 'r') as f:
    line = f.read()
    coastal_states = line.split(',')
```

Добавим эту информацию в базу фактов.

```
# Добавление данных в базу фактов
for state in coastal_states:
    fact(coastal, state)
```

Прочитаем данные о смежности штатов.

```
# Чтение файла, содержащего данные о смежности штатов
with open(file_adjacent, 'r') as f:
    adjlist = [line.strip().split(',') for line in f if line and
               line[0].isalpha()]
```

Добавим информацию о смежности штатов в базу фактов.

```
# Добавление информации в базу данных
for L in adjlist:
    head, tail = L[0], L[1:]
    for state in tail:
        fact(adjacent, head, state)
```

Инициализируем переменные x и y.

```
# Инициализация переменных
x = var()
y = var()
```

Теперь мы готовы к тому, чтобы задать системе некоторые вопросы. Проверим, соседствует ли Невада с Луизианой.

```
# Проверка того, соседствует ли Невада с Луизианой
output = run(0, x, adjacent('Nevada', 'Louisiana'))
print('\nIs Nevada adjacent to Louisiana?:')
print('Yes' if len(output) else 'No')
```

Выведем все штаты, соседствующие со штатом Орегон.

```
# Штаты, соседствующие со штатом Орегон
output = run(0, x, adjacent('Oregon', x))
print('\nList of states adjacent to Oregon:')
for item in output:
    print(item)
```

Выведем список всех прибрежных штатов, соседствующих со штатом Миссисипи.

```
# Штаты, которые соседствуют с Миссисипи и являются прибрежными
output = run(0, x, adjacent('Mississippi', x), coastal(x))
print('\nList of coastal states adjacent to Mississippi:')
for item in output:
    print(item)
```

Выведем список семи штатов, граничащих с прибрежным штатом.

```
# Вывод списка 'n' штатов, граничащих с прибрежным штатом
n = 7
output = run(n, x, coastal(y), adjacent(x, y))
print('\nList of ' + str(n) + ' states that border a coastal state:')
for item in output:
    print(item)
```

Выведем список штатов, соседствующих как с Арканзасом, так и с Кентукки.

```
# Список штатов, соседствующих в двумя заданными штатами
output = run(0, x, adjacent('Arkansas', x), adjacent('Kentucky', x))
print('\nList of states that are adjacent to Arkansas and Kentucky:')
for item in output:
    print(item)
```

Полный код этого примера содержится в файле `states.py`. После выполнения этого кода в окне терминала отобразится следующая информация (рис. 6.6).

```
Is Nevada adjacent to Louisiana?:
No

List of states adjacent to Oregon:
Washington
California
Nevada
Idaho

List of coastal states adjacent to Mississippi:
Alabama
Louisiana

List of 7 states that border a coastal state:
Georgia
Pennsylvania
Massachusetts
Wisconsin
Maine
Oregon
Ohio

List of states that are adjacent to Arkansas and Kentucky:
Missouri
Tennessee
```

Рис. 6.6

Можете сверить эти результаты с картой США, чтобы убедиться в том, что все ответы являются правильными. Кроме того, вы можете задать программе дополнительные вопросы, чтобы выяснить, сможет ли она ответить на них.

Создание решателя головоломок

Другая интересная область применения логического программирования — решение головоломок. Мы можем задать условия головоломки, и программа ее решит. В этом разделе мы будем указывать различные элементы информации о четырех людях и задавать вопросы относительно недостающих частей информации.

В логической программе мы сформулируем головоломку следующим образом.

- У Стива есть автомобиль синего цвета.
- Человек, у которого есть кот, живет в Канаде.
- Мэтью живет в США.
- Человек, у которого есть автомобиль черного цвета, живет в Австралии.
- У Джека есть кот.
- Альфред живет в Австралии.
- Человек, у которого есть собака, живет во Франции.
- У кого есть кролик?

Цель головоломки состоит в том, чтобы определить человека, у которого есть кролик. На рис. 6.7 приведена подробная информация обо всех четырех людях.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from logpy import *
from logpy.core import lall
```

Объявим переменную people.

```
# Объявление переменной
people = var()
```

Определим все правила, используя функцию lall. Первое правило гласит, что число людей равно 4.

```
# Определение правил
rules = lall(
    # Число людей - 4
    (eq, (var(), var(), var(), var()), people),
```

Name	Pet	Car color	Country
Steve	dog	blue	France
Jack	cat	green	Canada
Matthew	rabbit	yellow	USA
Alfred	parrot	black	Australia

Рис. 6.7

У Стива есть автомобиль синего цвета.

```
# У Стива есть автомобиль синего цвета
(membero, ('Steve', var(), 'blue', var()), people),
```

Человек, у которого есть кот, живет в Канаде.

```
# Человек, у которого есть кот, живет в Канаде
(membero, (var(), 'cat', var(), 'Canada'), people),
```

Мэтью живет в США.

```
# Мэтью живет в США
(membero, ('Matthew', var(), var(), 'USA'), people),
```

Человек, у которого есть автомобиль черного цвета, живет в Австралии.

```
# Человек, у которого есть автомобиль черного цвета,
# живет в Австралии
(membero, (var(), var(), 'black', 'Australia'), people),
```

У Джека есть кот.

```
# У Джека есть кот
(membero, ('Jack', 'cat', var(), var()), people),
```

Альфред живет в Австралии.

```
    # Альфред живет в Австралии
    (membero, ('Alfred', var(), var(), 'Australia'), people),
```

Человек, у которого есть собака, живет во Франции.

```
    # Человек, у которого есть собака, живет во Франции
    (membero, (var(), 'dog', var(), 'France'), people),
```

У одного из людей этой группы есть кролик. Кто этот человек?

```
    # У кого есть кролик?
    (membero, (var(), 'rabbit', var(), var()), people)
)
```

Выполним решатель с использованием только что сформулированных условий.

```
# Выполнение решателя
solutions = run(0, people, rules)
```

Извлечем результат из решения.

```
# Извлечение результата
output = [house for house in solutions[0] if 'rabbit' in house][0][0]
```

Выведем полную матрицу, полученную с помощью решателя.

```
# Вывод результата
print('\n' + output + ' is the owner of the rabbit')
print('\nHere are all the details:')
attribs = ['Name', 'Pet', 'Color', 'Country']
print('\n' + '\t\t'.join(attribs))
print('=' * 57)
for item in solutions[0]:
    print('')
    print('\t\t'.join([str(x) for x in item]))
```

Полный код этого примера содержится в файле `puzzle.py`. После выполнения этого кода в окне терминала отобразится следующая информация (рис. 6.8).

```
Matthew is the owner of the rabbit

Here are all the details:

Name            Pet             Color           Country
=========================================================

Steve           dog             blue            France

Jack            cat             ~_9             Canada

Matthew         rabbit          ~_11            USA

Alfred          ~_13            black           Australia
```

Рис. 6.8

На рис. 6.8 приведены все значения, полученные с помощью решателя. Некоторые из них все еще остаются неизвестными, на что указывают нумерованные имена переменных. Несмотря на то что исходная информация была неполной, решателю удалось ответить на наш вопрос. Но для того, чтобы программа могла ответить на любой вопрос такого рода, нам могут понадобиться дополнительные правила. Данная программы была представлена вам исключительно для того, чтобы продемонстрировать, как решать головоломки в условиях, когда информация задана в неполном объеме. Можете самостоятельно поэкспериментировать с созданием решателей головоломок для различных сценариев.

Резюме

Из этой главы вы узнали о том, как писать программы на Python в рамках парадигмы логического программирования. Мы обсудили способы создания программ с использованием этих парадигм, подробно рассмотрев логическое программирование.

Мы реализовали несколько программ на языке Python, предназначенных для решения некоторых задач и головоломок. В следующей главе речь пойдет о методах эвристического поиска и будет продемонстрировано использование соответствующих алгоритмов для решения реальных задач.

7
Методы эвристического поиска

Данная глава посвящена методам эвристического поиска. Целью этих методов является сокращение объема вычислений по перебору вариантов при проведении поиска в пространстве решений. Этот поиск выполняется с использованием эвристик, управляющих поисковым алгоритмом. Эвристики позволяют алгоритму ускорить процесс получения решения, на что в противном случае ушло бы длительное время.

В этой главе вы ознакомитесь со следующими темами:

- что такое эвристический поиск;
- неинформированный и информированный виды поиска;
- задачи с ограничениями;
- методы локального поиска;
- алгоритм имитации отжига;
- конструирование строк с использованием жадного поиска;
- решение задачи с ограничениями;
- решение задачи раскрашивания областей;
- создание программы для поиска решений в игре "8";
- создание программы для прохождения лабиринтов.

Что такое эвристический поиск

Поиск и организация данных — важная тема в области искусственного интеллекта. Существует целый ряд задач, в которых требуется найти правильный ответ среди множества возможных вариантов решения. Эффективная организация данных обеспечивает быстрый и результативный поиск решений.

Часто число возможных вариантов решения задачи настолько велико, что разработка алгоритма, способного найти единственно правильное решение, становится невозможной. Алгоритм полного перебора всех потенциальных способов решения задачи также может оказаться фактически нереализуемым ввиду большого количества возможных вариантов. В подобных случаях мы полагаемся на приближенные методы, исключая заведомо неверные варианты. Такой подход называют *эвристическим*. Методы, предполагающие использование эвристик для организации поиска, называют **эвристическими методами поиска**.

Эвристические методы удобны тем, что позволяют ускорить процесс поиска. Даже если эвристический подход не в состоянии полностью исключить некоторые из вариантов решений, он обеспечивает их упорядочение, позволяя приблизиться к лучшему решению.

Неинформированный и информированный виды поиска

Если вы изучали компьютерные науки, то, должно быть, знаете о существовании таких разновидностей поиска, как **поиск в глубину** (Depth First Search — DFS), **поиск в ширину** (Breadth First Search — BFS) и **поиск с равномерной стоимостью** (Uniform Cost Search — UCS). Эти виды поиска, обычно применяемые для поиска решений при работе с графами, служат примером **неинформированного поиска** (uninformed search). Подобные методы не используют никакой предварительной информации или правил для исключения некоторых путей поиска решений. Они проверяют все вероятные пути и выбирают оптимальный.

С другой стороны, эвристический поиск называют **информированным** (informed search), поскольку он использует априорную информацию или правила для исключения излишних путей. В неинформированных методах поиска цель принимается в расчет. Этим методам действительно ничего не известно о том, в каком именно направлении следует продвигаться, если только в процессе поиска им не удается найти целевое решение.

Мы можем использовать эвристики для управления поиском решений в задачах с графами. Например, мы можем определить в каждом узле эвристическую функцию, возвращающую оценку стоимости пути от текущего узла до цели. Определяя эту эвристическую функцию, мы информируем поисковый метод о правильном направлении продвижения к цели, что позволяет алгоритму идентифицировать соседний узел, через который должен пролегать путь.

Следует отметить, что эвристический поиск не всегда приводит к наиболее оптимальному решению. Это может происходить потому, что мы не

исследуем каждую отдельную возможность и полагаемся на эвристику. Но такой подход гарантирует нахождение приемлемого решения за разумное время, что делает его полезным с практической точки зрения. Эвристические методы эффективны в быстром получении разумного варианта решения. Их используют в тех случаях, когда получение решения задачи иным способом невозможно или заняло бы слишком много времени.

Задачи с ограничениями

Существует много задач, которые должны решаться с учетом определенных ограничений. Этими ограничениями в основном являются условия, которые не могут быть нарушены в процессе решения задачи. Такие задачи называют **задачами с ограничениями** (Constraint Satisfaction Problems — CSP).

Задачи с ограничениями — это, по сути, математические задачи, определенные в виде набора переменных, которые должны удовлетворять ряду ограничений. Если мы приходим к окончательному решению, то состояния переменных должны подчиняться всем ограничениям. Такой подход предполагает представление сущностей конкретной задачи в виде коллекции фиксированного числа ограничений, налагаемых на переменные. В данном случае для нахождения решений необходимо использовать методы, обеспечивающие соблюдение указанных ограничений.

В задачах подобного рода требуется, чтобы сочетание эвристических и других методов поиска обеспечивало получение подходящего, пусть даже не оптимального, решения за короткое время. В нашем случае методы, обеспечивающие соблюдение ограничений, будут использоваться для решения задач в конечных областях. Конечная область состоит из конечного числа элементов, поэтому для нахождения решения можно воспользоваться поисковыми методами.

Методы локального поиска

Локальный поиск — это частный способ решения задач с ограничениями. В данном случае значения переменных постоянно обновляются до тех пор, пока не достигнут конечного состояния. Алгоритмы этой группы изменяют значения на каждом шаге процесса, который приближает нас к цели. Обновленные значения находятся в пространстве решений ближе к цели, чем предыдущие. Именно поэтому данный процесс и назвали *локальным* поиском.

Алгоритмы локального поиска являются эвристическими. Эти алгоритмы используют функцию, которая рассчитывает качество каждого обновления. Например, такая функция может подсчитывать количество ограничений, нарушаемых текущим обновлением, или проверять, каким образом обновления

влияют на расстояние до цели. Такой подход называют *стоимостным анализом*. Общей целью локального поиска является поиск обновлений, минимизирующих стоимость на каждом шаге.

Одним из популярных методов локального поиска является метод *крутого восхождения* (hill climbing). В нем используется эвристическая функция, измеряющая расстояние между текущим состоянием и целью. В самом начале процесса выполняется проверка того, не является ли состояние конечной целью. Если это действительно так, то процесс прекращается. В противном случае выбирается обновление и генерируется новое состояние. Если новое состояние оказывается ближе к цели, чем текущее, оно становится текущим состоянием. В противном случае обновление игнорируется, и процесс продолжается до тех пор, пока не будут проверены все возможные обновления. В общих чертах этот процесс напоминает восхождение к вершине холма, когда оптимальный маршрут подъема по склонам неизвестен.

Алгоритм имитации отжига

Алгоритм **имитации отжига** (simulated annealing) — это тип локального поиска, который одновременно можно отнести и к методам стохастического поиска. Методы стохастического поиска интенсивно используются в таких областях, как робототехника, химия, машиностроение, медицина, экономика и т.п. В частности, они позволяют оптимизировать конструкцию робота, определить стратегию выполнения последовательности операций в автоматических системах управления производством или осуществлять планирование дорожного движения. Стохастические алгоритмы применяются для решения многих реальных задач.

Алгоритм имитации отжига — это разновидность алгоритма крутого восхождения. Одной из основных проблем последней методики является возможность попадания на ложные вершины, т.е. локальные максимумы. Поэтому перед принятием каких-либо решений лучше всего исследовать все пространство для получения предварительного представления о его структуре. Это позволяет избежать попадания в локальные максимумы или на плато, откуда невозможно выбраться.

Используя алгоритм имитации отжига, мы переформулируем задачу и решаем ее, обеспечивая достижение минимума некоторой величины, а не максимума. Поэтому в данном случае аналогией служит не восхождение к вершине холма, а спуск в долину. Однако, несмотря на сходство осуществляемых операций, они выполняются немного по-разному. Выполнением поиска управляет *целевая функция*, которая служит нашей эвристикой.

Название данного алгоритма обусловлено тем, что он имитирует один из реальных металлургических процессов. Сначала мы нагреваем металл, расплавляя его, а затем позволяем ему охлаждаться до тех пор, пока он не достигнет оптимального энергетического состояния.

Скорость, с которой мы охлаждаем систему, называется **графиком отжига** (annealing schedule). Скорость охлаждения играет важную роль в силу того, что она оказывает самое непосредственное влияние на результат. В реальном физическом процессе, происходящем в металлах, если охлаждение происходит слишком быстро, то состояние металла стабилизируется в локальном максимуме. Например, в такое неоптимальное состояние с локальным максимумом перейдет расплавленный металл, если его опустить в холодную воду.

Если же скорость охлаждения мала и контролируется, то тем самым мы предоставляем металлу возможность перейти в глобально оптимальное состояние. При таких условиях вероятность совершения больших шагов в направлении любого отдельного холма снижается. В силу невысокой скорости охлаждения система получает шанс выбрать наилучшее состояние. Нечто подобное происходит и с данными.

Сначала мы вычисляем текущее состояние и проверяем, не является ли оно целевым. Если это так, то процесс прекращается. В противном случае мы присваиваем переменной, обозначающей наилучшее состояние, текущее значение состояния. После этого мы определяем график отжига, который управляет тем, насколько быстрым должен быть спуск в долину. Затем мы вычисляем разницу между текущим и новым состояниями. Если новое состояние не лучше текущего, мы делаем его текущим с некоторой предварительно установленной долей вероятности. Это достигается за счет использования генератора случайных чисел и принятия решения на основании некоего порогового значения. В случае превышения порога мы устанавливаем данное состояние в качестве наилучшего. Исходя из этого, мы обновляем график отжига в зависимости от количества узлов. Этот процесс продолжается вплоть до достижения целевого состояния.

Конструирование строк с использованием жадного поиска

Жадный поиск — это алгоритмическая парадигма, в соответствии с которой для нахождения глобального оптимума на каждом шаге делается локально оптимальный выбор. Но во многих задачах жадные алгоритмы поиска не приводят к глобально оптимальным решениям. Преимуществом жадных алгоритмов является то, что они позволяют получить приближенное решение

за достаточно короткий промежуток времени. При этом предполагается, что результат аппроксимации оказывается достаточно близким к оптимальному глобальному решению.

Жадные алгоритмы не уточняют свои решения на основании новой информации в процессе поиска. Предположим, вы планируете поездку на автомобиле и хотите выбрать наилучший возможный маршрут. Если вы используете для этого жадный алгоритм, то он предложит вам маршруты, которые, возможно, будут самыми короткими, но при этом займут больше времени. Он также может проложить маршрут, который в ближайшей перспективе покажется самым коротким, но впоследствии заведет вас в дорожную пробку. Подобное может происходить по той причине, что жадные алгоритмы продумывают лишь следующий шаг, а не ищут оптимальное глобальное решение.

Обратимся к примеру решения задачи с помощью алгоритма жадного поиска. В этой задаче мы попытаемся воссоздать входную строку, выбирая буквы из алфавита. Мы поручим алгоритму выполнить поиск в пространстве решений и сконструировать путь к решению.

В данной главе мы будем использовать пакет `simpleai`. Он содержит различные программы, которые могут быть использованы для построения решений с использованием метода эвристического поиска. Этот пакет доступен по адресу `https://github.com/simpleai-team/simpleai`. Нам потребуется внести в него некоторые изменения, чтобы он работал в Python 3. Среди файлов примеров, прилагаемых к книге, вы найдете файл `simpleai.zip`. Распакуйте его в папку `simpleai`. В ней будут содержаться все необходимые изменения, которые должны быть внесены в оригинальную библиотеку, чтобы обеспечить ее работу с Python 3. Поместите папку `simpleai` в ту же папку, в которой находится ваш код, что обеспечит его корректное выполнение.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse
import simpleai.search as ss
```

Определим функцию для анализа входных аргументов.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Creates the /
            input string using the greedy algorithm')
    parser.add_argument("--input-string", dest="input_string",
                        required=True, help="Input string")
    parser.add_argument("--initial-state", dest="initial_state",
                        required=False, default='',
                        help="Starting point for the search")
    return parser
```

Создадим класс, содержащий методы, необходимые для решения задачи. Этот класс наследует библиотечный класс `SearchProblem`. Нам придется всего лишь перекрыть несколько методов. Сначала мы определим пользовательский метод `set_target`, задающий целевую строку.

```
class CustomProblem(ss.SearchProblem):
    def set_target(self, target_string):
        self.target_string = target_string
```

Метод `actions` определен в классе `SearchProblem`, и мы должны переопределить его. Этот метод отвечает за выбор подходящих шагов, ведущих к цели. Если длина текущей строки меньше длины целевой строки, то он возвращает алфавит. В противном случае возвращается пустая строка.

```
    # Проверка текущего состояния и выбор подходящего действия
    def actions(self, cur_state):
        if len(cur_state) < len(self.target_string):
            alphabets = 'abcdefghijklmnopqrstuvwxyz'
            return list(alphabets + ' ' + alphabets.upper())
        else:
            return []
```

Нам также потребуется метод, конкатенирующий текущую строку с действием. Этот метод определен в классе `SearchProblem`, и мы должны перекрыть его.

```
    # Конкатенация состояния и действия для получения результата
    def result(self, cur_state, action):
        return cur_state + action
```

Метод `is_goal` является частью класса `SearchProblem` и позволяет проверить, не достигнута ли цель.

```
    # Проверка достижения цели
    def is_goal(self, cur_state):
        return cur_state == self.target_string
```

Метод `heuristic` также является частью класса `SearchProblem`, и его тоже следует переопределить. Определим собственную эвристику, которую используем для решения задачи. Мы вычислим, насколько далеко находимся от цели, и применим полученный результат в качестве эвристики, управляющей работой алгоритма.

```
    # Определение эвристики, которую будем использовать
    def heuristic(self, cur_state):
        # Сравнение текущей строки с целевой строкой
```

```
        dist = sum([1 if cur_state[i] != self.target_string[i] else 0
                    for i in range(len(cur_state))])

        # Разность длин
        diff = len(self.target_string) - len(cur_state)

        return dist + diff
```

Извлечем аргументы командной строки.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
```

Инициализируем объект `CustomProblem`.

```
# Инициализация объекта
problem = CustomProblem()
```

Зададим начальную точку, а также цель, которой хотим достигнуть.

```
# Зададим целевую строку и начальное состояние
problem.set_target(args.input_string)
problem.initial_state = args.initial_state
```

Запустим решатель задачи.

```
# Решение задачи
output = ss.greedy(problem)
```

Отобразим путь к решению.

```
print('\nTarget string:', args.input_string)
print('\nPath to the solution:')
for item in output.path():
    print(item)
```

Полный код этого примера содержится в файле `greedy_search.py`. Выполнив его с пустым начальным состоянием, вы получите следующий вывод (рис. 7.1).

```
    $ python3 greedy_search.py --input-string 'Artificial
Intelligence' -- initial-state ''
```

```
Path to the solution:
(None, '')
('A', 'A')
('r', 'Ar')
('t', 'Art')
('i', 'Arti')
('f', 'Artif')
('i', 'Artifi')
('c', 'Artific')
('i', 'Artifici')
('a', 'Artificia')
('l', 'Artificial')
(' ', 'Artificial ')
('I', 'Artificial I')
('n', 'Artificial In')
('t', 'Artificial Int')
('e', 'Artificial Inte')
('l', 'Artificial Intel')
('l', 'Artificial Intell')
('i', 'Artificial Intelli')
('g', 'Artificial Intellig')
('e', 'Artificial Intellige')
('n', 'Artificial Intelligen')
('c', 'Artificial Intelligenc')
('e', 'Artificial Intelligence')
```

Рис. 7.1

Выполнив этот код с непустой начальной строкой, вы получите другой вывод (рис. 7.2).

```
$ python3 greedy_search.py --input-string 'Artificial
Intelligence with Python' --initial-state 'Artificial Inte'
```

Решение задачи с ограничениями

Мы уже обсуждали вопрос о том, как формулируются задачи с ограничениями. Применим эти знания к реальной задаче. В ней мы будем иметь дело со списком имен, каждое из которых может принимать лишь фиксированный набор значений. Кроме того, в условии задачи задан набор ограничений, которым должно подчиняться решение. Приступим к решению задачи.

```
Path to the solution:
(None, 'Artificial Inte')
('l', 'Artificial Intel')
('l', 'Artificial Intell')
('i', 'Artificial Intelli')
('g', 'Artificial Intellig')
('e', 'Artificial Intellige')
('n', 'Artificial Intelligen')
('c', 'Artificial Intelligenc')
('e', 'Artificial Intelligence')
(' ', 'Artificial Intelligence ')
('w', 'Artificial Intelligence w')
('i', 'Artificial Intelligence wi')
('t', 'Artificial Intelligence wit')
('h', 'Artificial Intelligence with')
(' ', 'Artificial Intelligence with ')
('P', 'Artificial Intelligence with P')
('y', 'Artificial Intelligence with Py')
('t', 'Artificial Intelligence with Pyt')
('h', 'Artificial Intelligence with Pyth')
('o', 'Artificial Intelligence with Pytho')
('n', 'Artificial Intelligence with Python')
```

Рис. 7.2

Создайте новый файл Python и импортируйте следующие пакеты.

```
from simpleai.search import CspProblem, backtrack, \
    min_conflicts, MOST_CONSTRAINED_VARIABLE, \
    HIGHEST_DEGREE_VARIABLE, LEAST_CONSTRAINING_VALUE
```

Определим ограничение, в соответствии с которым все переменные во входном списке должны иметь уникальные значения.

```
# Возможные значения каждой переменной должны быть разными
def constraint_unique(variables, values):
    # Проверка уникальности значений
    return len(values) == len(set(values))
```

Определим ограничение, в соответствии с которым первая переменная должна быть больше второй.

```
# Ограничение, в соответствии с которым одна переменная
# должны быть больше, чем вторая
```

```
def constraint_bigger(variables, values):
    return values[0] > values[1]
```

Определим ограничение, в соответствии с которым, если первая переменная является четной, то другая переменная должна быть нечетной, и наоборот.

```
# Ограничение, заключающееся в том, что из двух переменных
# одна должна быть четной, а другая - нечетной
def constraint_odd_even(variables, values):
    # Если первая переменная четная, то вторая должна быть
    # нечетной, и наоборот
    if values[0] % 2 == 0:
        return values[1] % 2 == 1
    else:
        return values[1] % 2 == 0
```

Определим основную функцию и переменные.

```
if __name__=='__main__':
    variables = ('John', 'Anna', 'Tom', 'Patricia')
```

Определим список значений, которые может иметь каждая из переменных.

```
    domains = {
        'John': [1, 2, 3],
        'Anna': [1, 3],
        'Tom': [2, 4],
        'Patricia': [2, 3, 4],
    }
```

Определим ограничения для различных сценариев. В данном случае мы определим следующие три ограничения:

- переменные John, Anna и Tom должны иметь разные значения;
- значение переменной Tom должно быть больше значения переменной Anna;
- если значение переменной John — нечетное, то значение переменной Patricia должно быть четным, и наоборот.

Используем следующий код.

```
    constraints = [
        (('John', 'Anna', 'Tom'), constraint_unique),
        (('Tom', 'Anna'), constraint_bigger),
        (('John', 'Patricia'), constraint_odd_even),
    ]
```

Используем предыдущие переменные и ограничения для инициализации объекта CspProblem.

```
problem = CspProblem(variables, domains, constraints)
```

Вычислим решение и выведем его.

```
print('\nSolutions:\n\nNormal:', backtrack(problem))
```

Вычислим решение с помощью эвристики MOST_CONSTRAINED_VARIABLE.

```
print('\nMost constrained variable:', backtrack(problem,
        variable_heuristic=MOST_CONSTRAINED_VARIABLE))
```

Вычислим решение с помощью эвристики HIGHEST_DEGREE_VARIABLE.

```
print('\nHighest degree variable:', backtrack(problem,
        variable_heuristic=HIGHEST_DEGREE_VARIABLE))
```

Вычислим решение с помощью эвристики LEAST_CONSTRAINING_VALUE.

```
print('\nLeast constraining value:', backtrack(problem,
        value_heuristic=LEAST_CONSTRAINING_VALUE))
```

Вычислим решение с помощью эвристики переменной MOST_CONSTRAINED_VARIABLE и эвристики значения LEAST_CONSTRAINING_VALUE.

```
print('\nMost constrained variable and least constraining \
        value:', backtrack(problem,
        variable_heuristic=MOST_CONSTRAINED_VARIABLE,
        value_heuristic=LEAST_CONSTRAINING_VALUE))
```

Вычислим решение с помощью эвристики переменной HIGHEST_DEGREE_VARIABLE и эвристики значения LEAST_CONSTRAINING_VALUE.

```
print('\nHighest degree and least constraining value:',
        backtrack(problem, variable_heuristic=
        HIGHEST_DEGREE_VARIABLE, value_heuristic=
        LEAST_CONSTRAINING_VALUE))
```

Вычислим решение с помощью эвристики минимального количества конфликтов.

```
print('\nMinimum conflicts:', min_conflicts(problem))
```

Полный код этого примера содержится в файле constrained_problem.py. Выполнив этот код, вы должны получить следующий вывод (рис. 7.3).

```
Solutions:
Normal: {'Patricia': 2, 'John': 1, 'Anna': 3, 'Tom': 4}
Most constrained variable: {'Patricia': 2, 'John': 3, 'Anna': 1, 'Tom': 2}
Highest degree variable: {'Patricia': 2, 'John': 1, 'Anna': 3, 'Tom': 4}
Least constraining value: {'Patricia': 2, 'John': 1, 'Anna': 3, 'Tom': 4}
Most constrained variable and least constraining value: {'Patricia': 2, 'John': 3, 'Anna': 1, 'Tom': 2}
Highest degree and least constraining value: {'Patricia': 2, 'John': 1, 'Anna': 3, 'Tom': 4}
Minimum conflicts: {'Patricia': 4, 'John': 1, 'Anna': 3, 'Tom': 4}
```

Рис. 7.3

Сверившись с условиями задачи, вы убедитесь в том, что решения удовлетворяют всем заданным ограничениям.

Решение задачи о раскраске областей

Применим рассмотренный подход для решения задачи о раскраске областей. Взгляните на приведенный ниже снимок экрана (рис. 7.4).

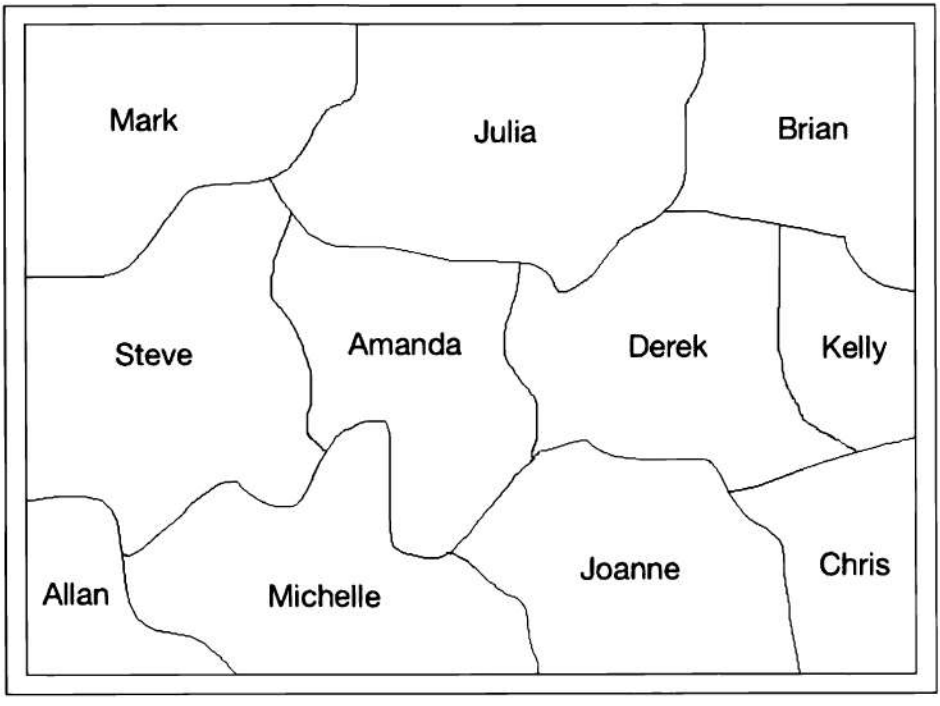

Рис. 7.4

На нем изображены несколько областей, помеченных именами. Мы должны так раскрасить эти области, используя четыре краски, чтобы цвет любой области не совпадал с цветом любой из смежных с ней областей.

Создайте новый файл Python и импортируйте следующие пакеты:

```
from simpleai.search import CspProblem, backtrack
```

Определим ограничение, в соответствии с которым значения должны быть разными.

```
# Определение функции, которая налагает ограничение,
# заключающееся в том, что соседние области должны
# иметь разные цвета
def constraint_func(names, values):
    return values[0] != values[1]
```

Определим основную функцию и список имен.

```
if __name__=='__main__':
    # Определение переменных
    names = ('Mark', 'Julia', 'Steve', 'Amanda', 'Brian',
             'Joanne', 'Derek', 'Allan', 'Michelle', 'Kelly')
```

Определим список возможных цветов.

```
    # Определение возможных цветов
    colors = dict((name, ['red', 'green', 'blue', 'gray']) for name in
                  names)
```

Мы должны привести эту информацию к виду, понятному для алгоритма. Определим ограничения, указав список людей, которые соседствуют друг с другом.

```
    # Определение ограничений
    constraints = [
      (('Mark', 'Julia'), constraint_func),
      (('Mark', 'Steve'), constraint_func),
      (('Julia', 'Steve'), constraint_func),
      (('Julia', 'Amanda'), constraint_func),
      (('Julia', 'Derek'), constraint_func),
      (('Julia', 'Brian'), constraint_func),
      (('Steve', 'Amanda'), constraint_func),
      (('Steve', 'Allan'), constraint_func),
      (('Steve', 'Michelle'), constraint_func),
      (('Amanda', 'Michelle'), constraint_func),
      (('Amanda', 'Joanne'), constraint_func),
```

```
    (('Amanda', 'Derek'), constraint_func),
    (('Brian', 'Derek'), constraint_func),
    (('Brian', 'Kelly'), constraint_func),
    (('Joanne', 'Michelle'), constraint_func),
    (('Joanne', 'Amanda'), constraint_func),
    (('Joanne', 'Derek'), constraint_func),
    (('Joanne', 'Kelly'), constraint_func),
    (('Derek', 'Kelly'), constraint_func),
]
```

Используем переменные и ограничения для инициализации объекта.

```
# Решение задачи
problem = CspProblem(names, colors, constraints)
```

Решим задачу и выведем решение.

```
# Вывод решения
output = backtrack(problem)
print('\nColor mapping:\n')
for k, v in output.items():
    print(k, '==>', v)
```

Полный код этого примера содержится в файле `coloring.py`. После его выполнения в окне терминала отобразится следующая информация (рис. 7.5).

```
Color mapping:

Derek ==> blue
Michelle ==> gray
Allan ==> red
Steve ==> blue
Julia ==> green
Amanda ==> red
Joanne ==> green
Mark ==> red
Kelly ==> gray
Brian ==> red
```

Рис. 7.5

Если вы раскрасите области на основании этой информации, то увидите следующий результат (рис. 7.6).

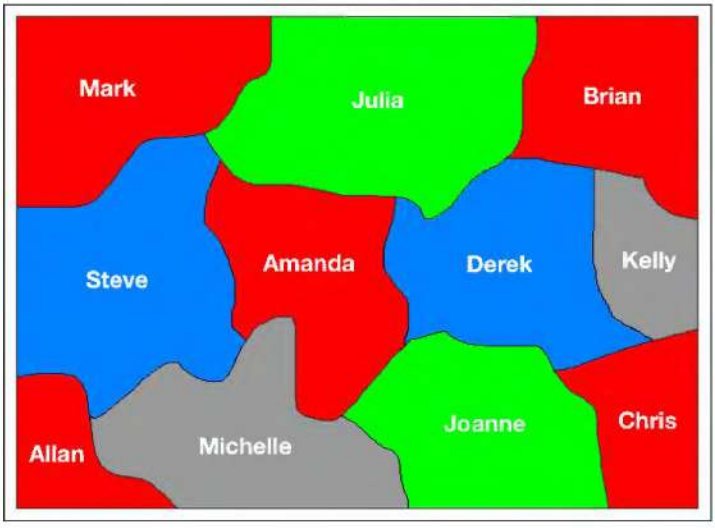

Рис. 7.6. (См. цветную вклейку; адрес указан во введении)

Нетрудно убедиться в том, что на этой карте отсутствуют смежные области, окрашенные в один и тот же цвет.

Создание головоломки "8"

Игра "8" — это разновидность игры "15" (`https://ru.wikipedia.org/wiki/Игра_в_15`). Игровое поле представляет собой квадратную сетку, где случайным образом расставлены числа от 1 до 8, которые необходимо расположить в порядке возрастания, перемещая соответствующие поля. Вы сможете поиграть в эту игру на сайте `http://mypuzzle.org/sliding`.

Для решения этой задачи мы используем **алгоритм A***. Этот алгоритм применяется для поиска путей в графе. Он представляет собой сочетание **алгоритма Дейкстры** и жадного поиска. Вместо того чтобы угадывать вслепую следующий шаг, алгоритм A* выбирает тот шаг, который представляется наиболее перспективным. В каждом узле мы генерируем список всех возможных шагов и выбираем из них шаг с минимальной стоимостью, который требуется выполнить для достижения цели.

Перейдем к определению функции стоимости. Нам нужно вычислить стоимость в каждом узле. Она представлена двумя слагаемыми, первым из которых является стоимость достижения текущего узла, а вторым — стоимость достижения цели из текущего узла.

Мы будем использовать это суммирование в качестве эвристики. Следует отметить, что оценка, выраженная вторым слагаемым стоимости, не вполне идеальна. Если бы она была идеальной, то алгоритм A* сходился бы быстро. Но обычно это не так. Для нахождения наилучшего пути к решению требуется некоторое время. Тем не менее алгоритм A* весьма эффективен в нахождении оптимальных путей, благодаря чему пользуется наибольшей популярностью.

Используем алгоритм A* для создания решателя головоломки "8". Мы рассмотрим вариант решения, предложенный в библиотеке `simpleai`. Создайте новый файл Python и импортируйте следующие пакеты:

```
from simpleai.search import astar, SearchProblem
```

Определим класс, который содержит методы, необходимые для решения головоломки "8".

```
# Класс, содержащий методы для решения головоломки
class PuzzleSolver(SearchProblem):
```

Переопределим метод `actions` с учетом специфики нашей задачи.

```
    # Метод для получения списка чисел, которые могут быть
    # перемещены в пустую клетку
    def actions(self, cur_state):
        rows = string_to_list(cur_state)
        row_empty, col_empty = get_location(rows, 'e')
```

Проверим позицию пустой клетки и создадим новое действие.

```
        actions = []
        if row_empty > 0:
            actions.append(rows[row_empty - 1][col_empty])
        if row_empty < 2:
            actions.append(rows[row_empty + 1][col_empty])
        if col_empty > 0:
            actions.append(rows[row_empty][col_empty - 1])
        if col_empty < 2:
            actions.append(rows[row_empty][col_empty + 1])

        return actions
```

Переопределим метод `result`. Преобразуем строку в список и извлечем позицию пустой клетки. Сгенерируем результат, обновив позиции чисел.

```
    # Вернем результирующее состояние после перемещения
    # числа в пустую клетку
```

```
def result(self, state, action):
    rows = string_to_list(state)
    row_empty, col_empty = get_location(rows, 'e')
    row_new, col_new = get_location(rows, action)

    rows[row_empty][col_empty], rows[row_new][col_new] = \
    rows[row_new][col_new], rows[row_empty][col_empty]

    return list_to_string(rows)
```

Проверим, достигнута ли цель.

```
# Возвращает значение true, если состояние целевое
def is_goal(self, state):
    return state == GOAL
```

Определим метод `heuristic`. Мы используем эвристику, которая вычисляет Манхэттенское расстояние[1] между текущим и целевым состояниями.

```
# Вычисление оценки удаленности состояния от цели
# с использованием Манхэттенского расстояния
def heuristic(self, state):
    rows = string_to_list(state)

    distance = 0
```

Вычислим расстояние.

```
    for number in '12345678e':
        row_new, col_new = get_location(rows, number)
        row_new_goal, col_new_goal = goal_positions[number]

        distance += abs(row_new - row_new_goal) + abs(col_new -
                    col_new_goal)
    return distance
```

Определим функцию для преобразования списка в строку.

```
# Преобразование списка в строку
def list_to_string(input_list):
    return '\n'.join([''.join(x) for x in input_list])
```

Определим функцию для преобразования строки в список.

[1] См. в Википедии по адресу `https://ru.wikipedia.org/wiki/Манхэттенское_расстояние`. — *Примеч. ред.*

```
# Преобразование строки в список
def string_to_list(input_string):
    return [x.split('-') for x in
        input_string.split('\n')]
```

Определим функцию для получения позиции заданного элемента в сетке.

```
# Определение 2D-положения входного элемента
def get_location(rows, input_element):
    for i, row in enumerate(rows):
      for j, item in enumerate(row):
        if item == input_element:
          return i, j
```

Определим начальное состояние и цель, которую хотим достигнуть.

```
# Конечный результат, которого мы хотим достигнуть
GOAL = '''1-2-3
4-5-6
7-8-e'''

# Начальная точка
INITIAL = '''1-e-2
6-3-4
7-5-8'''
```

Отследим позицию цели для каждого элемента путем создания переменной.

```
# Создание кеша для позиции цели каждого элемента
goal_positions = {}
rows_goal = string_to_list(GOAL)
for number in '12345678e':
    goal_positions[number] = get_location(rows_goal, number)
```

Создадим объект решателя A*, используя определенное ранее начальное состояние, и извлечем результат.

```
# Создание объекта решателя
result = astar(PuzzleSolver(INITIAL))
```

Выведем решение.

```
# Вывод результатов
for i, (action, state) in enumerate(result.path()):
    print()
    if action == None:
```

```
        print('Initial configuration')
    elif i == len(result.path()) - 1:
        print('After moving', action, 'into the empty space.
              Goal achieved!')
    else:
        print('After moving', action, 'into the empty space')

    print(state)
```

Полный код этого примера содержится в файле `puzzle.py`. Выполнив этот код, вы увидите в окне своего терминала длинный вывод, начало которого выглядит так (рис. 7.7).

```
Initial configuration
1-e-2
6-3-4
7-5-8

After moving 2 into the empty space
1-2-e
6-3-4
7-5-8

After moving 4 into the empty space
1-2-4
6-3-e
7-5-8

After moving 3 into the empty space
1-2-4
6-e-3
7-5-8

After moving 6 into the empty space
1-2-4
e-6-3
7-5-8
```

Рис. 7.7

Прокручивая эту информацию, вы сможете проследить за шагами, которые привели к данному решению. Завершающий фрагмент этого вывода будет таким (рис. 7.8).

```
After moving 2 into the empty space
e-2-3
1-4-6
7-5-8

After moving 1 into the empty space
1-2-3
e-4-6
7-5-8

After moving 4 into the empty space
1-2-3
4-e-6
7-5-8

After moving 5 into the empty space
1-2-3
4-5-6
7-e-8

After moving 8 into the empty space. Goal achieved!
1-2-3
4-5-6
7-8-e
```

Рис. 7.8

Создание решателя для прохождения лабиринта

Используем алгоритм A* для прохождения лабиринта. Обратимся к рис. 7.9.

Символами # обозначены препятствия. Символ o представляет начальную точку, а символ x — цель. Наша задача заключается в том, чтобы найти кратчайший путь от начальной до конечной точки. Рассмотрим, как это можно сделать в Python. Представленное ниже решение является вариантом решения, предложенным в библиотеке `simpleai`. Создайте новый файл Python и импортируйте следующие пакеты.

```
import math
from simpleai.search import SearchProblem, astar
```

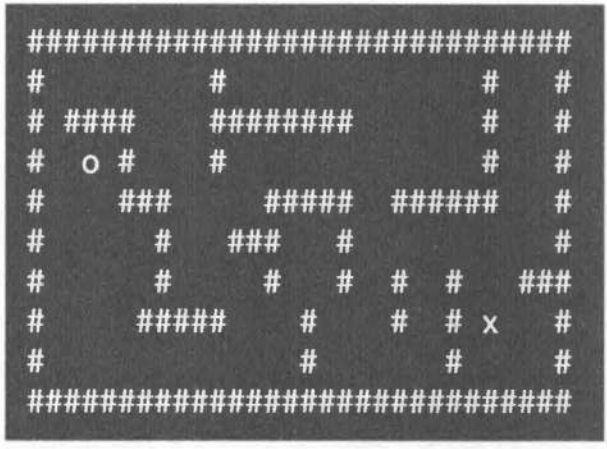

Рис. 7.9

Создадим класс, который содержит методы, необходимые для решения задачи.

```
# Класс, содержащий методы для прохождения лабиринта
class MazeSolver(SearchProblem):
```

Определим метод-инициализатор.

```
# Инициализация класса
    def __init__(self, board):
        self.board = board
        self.goal = (0, 0)
```

Извлечем начальную и конечную позиции.

```
        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "o":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "x":
                    self.goal = (x, y)

        super(MazeSolver, self).__init__(initial_state=self.initial)
```

Переопределим метод `actions`. В каждой позиции мы должны проверять стоимость перехода в соседние ячейки, а затем присоединять все возможные действия. Если соседняя ячейка блокирована, соответствующее действие не рассматривается.

```
    # Определение метода, предпринимающего действия, которые
    # приводят к достижению цели
    def actions(self, state):
        actions = []
        for action in COSTS.keys():
            newx, newy = self.result(state, action)
            if self.board[newy][newx] != "#":
                actions.append(action)

        return actions
```

Переопределим метод `result`. В зависимости от текущего состояния и входного действия обновляем координаты x и y.

```
    # Обновление состояния на основании действия
    def result(self, state, action):
        x, y = state

        if action.count("up"):
            y -= 1
        if action.count("down"):
            y += 1
        if action.count("left"):
            x -= 1
        if action.count("right"):
            x += 1

        new_state = (x, y)

        return new_state
```

Проверим, достигнута ли конечная точка.

```
    # Проверка достижения цели
    def is_goal(self, state):
        return state == self.goal
```

Нам нужно определить функцию стоимости. Эта стоимость относится к перемещению в соседнюю ячейку и различна для вертикальных, горизонтальных и диагональных перемещений. Мы определим их позднее.

```
    # Вычисление стоимости действия
    def cost(self, state, action, state2):
        return COSTS[action]
```

Определим эвристику, которую будем использовать. В данном случае мы будем задействовать евклидово расстояние.

```
    # Эвристика, которую мы будем использовать
    # для получения решения
    def heuristic(self, state):
        x, y = state
        gx, gy = self.goal

        return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)
```

Определим основную функцию и лабиринт, который перед этим обсуждался.

```
if __name__ == "__main__":
    # Определение лабиринта
    MAP = """
    ################################
    #          #               #   #
    # ####     ########        #   #
    #  o #     #               #   #
    #    ###      #####  ######   #
    #       #   ###   #           #
    #       #     #   #  #  #   ###
    #     #####     #    #  # x   #
    #               #       #     #
    ################################
    """
```

Преобразуем информацию о лабиринте в список.

```
    # Преобразование карты в список
    print(MAP)
    MAP = [list(x) for x in MAP.split("\n") if x]
```

Определим стоимость перемещения по карте. Стоимость перемещения по диагонали больше, чем по горизонтали или вертикали.

```
    # Определение стоимости перемещения по лабиринту
    cost_regular = 1.0
    cost_diagonal = 1.7
```

Назначим значения стоимости соответствующим перемещениям.

```
# Создание словаря стоимости
COSTS = {
    "up": cost_regular,
    "down": cost_regular,
    "left": cost_regular,
    "right": cost_regular,
    "up left": cost_diagonal,
    "up right": cost_diagonal,
    "down left": cost_diagonal,
    "down right": cost_diagonal,
}
```

Создадим объект решателя, используя определенный перед этим класс.

```
# Создание объекта решателя лабиринта
problem = MazeSolver(MAP)
```

Запустим решатель для лабиринта и извлечем результат.

```
# Запуск решателя
result = astar(problem, graph_search=True)
```

Извлечем путь из результата.

```
# Извлечение результата
path = [x[1] for x in result.path()]
```

Выведем результат.

```
# Вывод результата
print()
for y in range(len(MAP)):
    for x in range(len(MAP[y])):
        if (x, y) == problem.initial:
            print('o', end='')
        elif (x, y) == problem.goal:
            print('x', end='')
        elif (x, y) in path:
            print(' ', end='')
        else:
            print(MAP[y][x], end='')
    print()
```

Полный код этого примера содержится в файле `maze.py`. Выполнив этот код, вы получите следующий вывод (рис. 7.10).

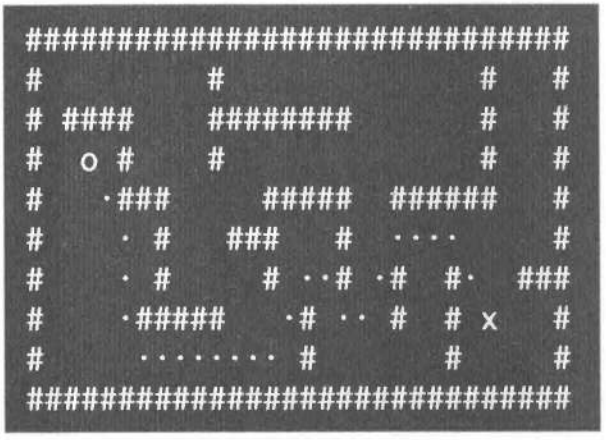

Рис. 7.10

Резюме

Из этой главы вы узнали о том, как работает эвристический поиск. Мы обсудили различие между неинформированным и информированным видами поиска. Было рассказано о том, что такое задачи с ограничениями и как решать задачи в рамках этой парадигмы. Мы обсудили локальные методы поиска и объяснили причины использования алгоритма имитации отжига. Жадный поиск был применен для решения задачи, связанной с обработкой строк, в рамках парадигмы задач с ограничениями.

Такой же подход бы применен для решения задачи о раскраске карты. Затем мы обсудили алгоритм A* и показали, как он может применяться для нахождения оптимальных путей решения задачи. Мы использовали этот алгоритм для решения головоломки "8" и прохождения лабиринта. В следующей главе мы обсудим генетические алгоритмы и продемонстрируем их применение на примере реальных задач.

8
Генетические алгоритмы

В этой главе вы изучите генетические алгоритмы. Мы обсудим понятия эволюционных алгоритмов и генетического программирования и покажем, как они соотносятся с генетическими алгоритмами. Вы узнаете о базовых строительных блоках генетических алгоритмов, включая кроссовер, мутацию и функцию приспособленности. Затем эти понятия будут использованы для создания различных систем.

К концу главы вы освоите следующие темы:

- эволюционные и генетические алгоритмы;
- фундаментальные понятия генетических алгоритмов;
- генерация битовых образов с предопределенными параметрами;
- визуализация хода эволюции;
- решение задачи символической регрессии;
- создание контроллера интеллектуального робота.

Эволюционные и генетические алгоритмы

Генетический алгоритм — это разновидность эволюционного алгоритма. Поэтому сначала мы должны обсудить, что собой представляют эволюционные алгоритмы. Эволюционный алгоритм — это метаалгоритм эвристической оптимизации, использующий принципы эволюции для решения задач. В данном контексте под *эволюцией* подразумевается процесс, аналогичный тому, который наблюдается в природе. Обычно получение решений основывается на непосредственном использовании функций и переменных, связанных с задачей. Однако в генетическом алгоритме любая задача кодируется в виде битовых образов, которыми манипулирует алгоритм.

Суть основной идеи эволюционных алгоритмов заключается в том, что мы берем популяцию агентов (индивидуумов) и применяем к ней правила естественного отбора. Мы начинаем с набора случайно выбранных индивидуумов, а затем идентифицируем наиболее жизнестойкие из них. Жизнестойкость каждого индивидуума устанавливается с использованием заранее определенной *функции приспособленности* (fitness function). Образно говоря, используется принцип **выживания наиболее приспособленных**.

Новое поколение индивидуумов создается посредством *рекомбинации* и *мутации* индивидуумов, выживших в процессе отбора. Обсуждению рекомбинации и мутации посвящен следующий раздел. А пока что вам достаточно знать, что эти методы реализуют механизм создания следующего поколения путем использования отобранных индивидуумов в качестве родителей.

В результате рекомбинации и мутации мы получаем новый набор индивидуумов, которые будут конкурировать с прежними за место в следующем поколении. Отбрасывая слабые индивидуумы и заменяя их более сильными потомками, мы повышаем общий уровень приспособленности популяции. Описанный процесс итеративно продолжается до тех пор, пока не будет достигнут желаемый уровень приспособленности.

Генетический алгоритм — это эволюционный алгоритм, в котором для нахождения строки битов, воплощающей решение задачи, используется эвристика. Это достигается за счет генерации новых поколений, состоящих из более стойких индивидуумов. Для генерации следующего поколения индивидуумов используются вероятностные операторы, такие как *отбор*, *кроссовер* и *мутация*. В сущности, индивидуумы представляются строками, каждая из которых является кодированной версией потенциального решения.

Используемая при этом функция приспособленности вычисляет степень приспособленности каждой строки, которая сообщает нам, насколько хорошо данная строка подходит для решения задачи. Функцию приспособленности также называют *оценочной функцией* (evaluation function). Генетические алгоритмы используют операторы, идею которых подсказала сама природа. Именно поэтому терминология в данной области тесно связана с терминами, используемыми в биологии.

Фундаментальные понятия генетических алгоритмов

Создание генетических алгоритмов требует понимания ряда ключевых понятий и владения соответствующей терминологией. Эти понятия широко используются при обсуждении генетических алгоритмов, предназначенных для решения различных задач. Одним из наиболее важных аспектов генетических

алгоритмов является фактор *случайности*. Описанный перед этим итеративный процесс основан на случайном отборе индивидуумов. Это означает, что данный процесс не является детерминированным. Следовательно, выполнив один и тот же алгоритм несколько раз, вы можете получить различные решения.

Что такое популяция? *Популяция* — это набор индивидуумов, являющихся потенциальными кандидатами на роль решений. В случае генетического алгоритма мы не поддерживается единственное наилучшее решение на любой стадии итеративного процесса. Вместо этого поддерживается набор потенциальных решений, одно из которых является наилучшим. Однако в процессе поиска другие решения также играют важную роль. Поскольку мы имеем популяцию решений, вероятность "застревания" в локальном оптимуме уменьшается. Застревание в *локальном оптимуме* — классическая проблема, с которой приходится сталкиваться при использовании различных методов оптимизации.

Теперь, когда вы узнали о популяциях и стохастической природе генетических алгоритмов, мы можем перейти к обсуждению операторов. Прежде чем создавать следующее поколение индивидуумов, мы должны убедиться в том, что они будут происходить от наиболее стойких индивидуумов текущего поколения. Одним из способов добиться этого является мутация. Генетический алгоритм случайным образом изменяет один или несколько индивидуумов текущего поколения и на основании этого генерирует новое решение-кандидат. Это изменение, которое называется *мутацией*, может сделать данный индивидуум как хуже, так и лучше существующих индивидуумов.

Следующим важным понятием является *рекомбинация*, или *кроссовер* (скрещивание). Это понятие непосредственно связано с воспроизводством популяции в процессе эволюции. Генетический алгоритм пытается комбинировать индивидуумы из текущего поколения для создания нового решения. Он комбинирует некоторые из свойств каждого родительского индивидуума для создания потомка. Этот процесс и получил название "кроссовер". Его целью является замена более слабых индивидуумов текущего поколения потомками, происходящими от более стойких индивидуумов популяции.

Для применения кроссовера и мутации необходимо располагать критериями отбора. Понятие *отбора* заимствовано из теории естественного отбора. Генетический алгоритм осуществляет отбор на каждой итерации. В ходе такого отбора сохраняются лишь наиболее стойкие индивидуумы, тогда как более слабые прекращают существование. Именно здесь вступает в игру идея выживания лишь наиболее приспособленных индивидуумов. Процесс отбора реализуется посредством функции приспособленности, которая вычисляет стойкость каждого индивидуума.

Генерация битовых образов с предопределенными параметрами

Теперь, когда вы представляете, как работают генетические алгоритмы, мы можем рассмотреть примеры их применения для решения некоторых задач. Мы будем использовать пакет Python deap. Его подробное описание вы найдете по адресу http://deap.readthedocs.io/en/master. Установите его, выполнив следующую команду в окне терминала:

```
$ pip3 install deap
```

Установив пакет, протестируйте его. Запустите интерпретатор Python, выполнив такую команду:

```
$ python3
```

После этого выполните следующую команду:

```
>>> import deap
```

Отсутствие сообщения об ошибке будет свидетельствовать о нормальной установке пакета.

В этом разделе мы будем решать вариант задачи One Max. В задаче One Max речь идет о том, чтобы сгенерировать битовую строку, которая содержит максимальное количество единиц. Это очень простая задача, но она позволит вам ознакомиться с библиотекой и понять, как реализовать решения с помощью эволюционного алгоритма. В данном случае мы попытаемся сгенерировать битовую строку, содержащую заданное количество единиц. Базовая структура и часть нашего кода будут аналогичны структуре и коду, которые используются в примере, приведенном в библиотеке DEAP.

Создайте новый файл Python и импортируйте следующие модули.

```
import random

from deap import base, creator, tools
```

Предположим, мы хотим сгенерировать битовый образ длиной 75 и хотим, чтобы он содержал 45 единиц. Мы должны определить оценочную функцию, которая будет использоваться для оценки степени продвижения к этой цели.

```
# Оценочная функция
def eval_func(individual):
    target_sum = 45
    return len(individual) - abs(sum(individual) - target_sum)
```

Если вы взглянете на формулу, которая используется для вычисления значения, возвращаемого этой функцией, то увидите, что оно достигает максимума тогда, когда количество единиц равно 45. Длина каждого индивидуума равна 75. Если количество единиц равно 45, то возвращаемое значение будет равно 75.

Теперь нам нужно определить функцию для создания инструментария. Определим объект `creator` для функции приспособленности, который будет отслеживать индивидуумы. Используемый далее класс `Fitness` — абстрактный и требует определения атрибута `weights`. Мы создаем максимизирующую приспособленность, используя положительные веса.

```
# Создание инструментария с подходящими параметрами
def create_toolbox(num_bits):
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)
```

Первая строка создает объект приспособленности `FitnessMax`, максимизирующий одиночную цель. Вторая строка связана с созданием индивидуума. В данном процессе первым создаваемым индивидуумом является список чисел с плавающей точкой. Для создания этого индивидуума мы должны создать класс `Individual`, используя объект `creator`. Атрибут `fitness` будет использовать объект `FitnessMax`, определенный ранее.

Объект `toolbox` обычно используется в библиотеке DEAP для хранения различных функций вместе с их аргументами. Создадим такой объект.

```
    # Инициализация инструментария
    toolbox = base.Toolbox()
```

Наши дальнейшие действия заключаются в регистрации функций в этом объекте. Начнем с генератора случайных чисел, который генерирует случайное число в диапазоне от 0 до 1. По сути, это делается для генерации битовых строк.

```
    # Генерирование атрибутов
    toolbox.register("attr_bool", random.randint, 0, 1)
```

Зарегистрируем функцию `individual`. Метод `initRepeat` имеет три аргумента: контейнерный класс для индивидуума, функция, заполняющая контейнер, и количество повторных вызовов данной функции.

```
    # Инициализация структур
    toolbox.register("individual", tools.initRepeat,
                     creator.Individual, toolbox.attr_bool, num_bits)
```

Теперь нам нужно зарегистрировать функцию population. Мы хотим, чтобы популяция была представлена списком индивидуумов.

```
# Определение популяции в виде списка индивидуумов
toolbox.register("population", tools.initRepeat, list,
                 toolbox.individual)
```

Нам также нужно зарегистрировать генетические операторы. Зарегистрируем определенную ранее оценочную функцию, которая будет выступать в качестве нашей функции приспособленности. Мы хотим, чтобы индивидуум, который является битовым образом, имел 45 единиц.

```
# Регистрация оператора оценки
toolbox.register("evaluate", eval_func)
```

Зарегистрируем оператор кроссовера mate, используя метод cxTwoPoint.

```
# Регистрация оператора кроссовера
toolbox.register("mate", tools.cxTwoPoint)
```

Зарегистрируем оператор мутации mutate, используя метод mutFlipBit. При этом мы должны задать вероятность мутации каждого атрибута с помощью аргумента indpb.

```
# Регистрация оператора мутации
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
```

Зарегистрируем оператор отбора selection, используя метод sel Tournament. Он задает, какие индивидуумы будут отобраны для размножения.

```
# Оператор выбора индивидуумов для размножения
toolbox.register("select", tools.selTournament, tournsize=3)
return toolbox
```

Вышеизложенное в основном является реализацией всех концепций, которые мы обсудили в предыдущем разделе. Функция, генерирующая набор инструментов, широко применяется в DEAP, и мы будем использовать ее на протяжении всей главы. Поэтому очень важно понимать, как генерируется этот инструментарий.

Определим основную функцию, начав с длины битового образа.

```
if __name__ == "__main__":
    # Определение количества битов
    num_bits = 75
```

Создадим набор инструментов, используя ранее определенную функцию.

```
# Создание набора инструментов с использованием параметра,
# определенного выше
toolbox = create_toolbox(num_bits)
```

Установим затравочное значение для генератора случайных чисел, чтобы обеспечить получение воспроизводимых результатов.

```
# Затравочное значение для генератора случайных чисел
random.seed(7)
```

Создадим начальную популяцию, состоящую, скажем, из 500 индивидуумов, используя метод, доступный в объекте `toolbox`. Вы сможете провести самостоятельные эксперименты, задавая различное количество индивидуумов в популяции.

```
# Создание начальной популяции из 500 индивидуумов
population = toolbox.population(n=500)
```

Определим вероятности скрещивания и мутации. Опять-таки, эти параметры определяются пользователем. Поэтому вы можете изменить данные значения по своему усмотрению, чтобы увидеть, как это повлияет на результат.

```
# Определение вероятностей скрещивания и мутации
probab_crossing, probab_mutating = 0.5, 0.2
```

Определим количество поколений, по которым будем итерировать до завершения процесса. Увеличив значение этого параметра, вы тем самым предоставите больше возможностей для повышения жизнеспособности популяции.

```
# Определение числа поколений
num_generations = 60
```

Проведем вычисления для всех индивидуумов в популяции, используя функции приспособленности.

```
print('\nStarting the evolution process')
# Проведение вычислений для всей популяции
fitnesses = list(map(toolbox.evaluate, population))
for ind, fit in zip(population, fitnesses):
    ind.fitness.values = fit
```

Начнем процесс итерирования по поколениям.

```
print('\nEvaluated', len(population), 'individuals')
# Итерации по поколениям
for g in range(num_generations):
    print("\n===== Generation", g)
```

Выберем в каждом поколении индивидуумов, переходящих в следующее поколение, используя оператор отбора, который мы до этого зарегистрировали в наборе инструментов.

```
# Выбор индивидуумов для перехода в следующее поколение
offspring = toolbox.select(population, len(population))
```

Клонируем выбранных индивидуумов.

```
# Клонирование отобранных индивидуумов
offspring = list(map(toolbox.clone, offspring))
```

Применим кроссовер и мутацию к индивидуумам следующего поколения, используя значения вероятности, определенные ранее. Сделав это, мы должны сбросить параметры приспособленности.

```
# Применение кроссовера и мутации к потомкам
for child1, child2 in zip(offspring[::2],
                          offspring[1::2]):
    # Скрестить двух индивидуумов
    if random.random() < probab_crossing:
        toolbox.mate(child1, child2)

        # "Забыть" параметры приспособленности детей
        del child1.fitness.values
        del child2.fitness.values
```

Применим мутацию к индивидуумам следующего поколения, используя значения вероятности, определенные ранее. Сделав это, мы должны сбросить параметр приспособленности.

```
# Применение мутации
for mutant in offspring:
    # Мутация индивидуума
    if random.random() < probab_mutating:
        toolbox.mutate(mutant)
        del mutant.fitness.values
```

Определим индивидуумов с недопустимыми значениями параметров приспособленности.

```
    # Определение индивидуумов с недопустимыми значениями
    # параметров приспособленности
    invalid_ind = [ind for ind in offspring if not
                   ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit
    print('Evaluated', len(invalid_ind), 'individuals')
```

Заменим популяцию индивидуумами следующего поколения.

```
    # Популяция полностью заменяется потомками
    population[:] = offspring
```

Выведем статистики для текущего поколения, чтобы проследить за ходом процесса.

```
    # Сбор всех значений приспособленности
    # в один список и вывод статистик
    fits = [ind.fitness.values[0] for ind in population]

    length = len(population)
    mean = sum(fits) / length
    sum2 = sum(x*x for x in fits)
    std = abs(sum2 / length - mean**2)**0.5

    print('Min =', min(fits), ', Max =', max(fits))
    print('Average =', round(mean, 2), ',
          Standard deviation =', round(std, 2))
print("\n==== End of evolution")
```

Вывод окончательного результата.

```
best_ind = tools.selBest(population, 1)[0]
print('\nBest individual:\n', best_ind)
print('\nNumber of ones:', sum(best_ind))
```

Полный код данного примера содержится в файле `bit_counter.py`. В процессе выполнения кода в окне терминала будут отображаться результаты, получаемые на каждой итерации. Начальный фрагмент выведенной информации будет выглядеть так (рис. 8.1).

Конечный фрагмент выведенной информации будет иметь примерно следующий вид, указывающий на завершение эволюции (рис. 8.2).

```
Starting the evolution process

Evaluated 500 individuals

===== Generation 0
Evaluated 297 individuals
Min = 58.0 , Max = 75.0
Average = 70.43 , Standard deviation = 2.91

===== Generation 1
Evaluated 303 individuals
Min = 63.0 , Max = 75.0
Average = 72.44 , Standard deviation = 2.16

===== Generation 2
Evaluated 310 individuals
Min = 65.0 , Max = 75.0
Average = 73.31 , Standard deviation = 1.6

===== Generation 3
Evaluated 273 individuals
Min = 67.0 , Max = 75.0
Average = 73.76 , Standard deviation = 1.41
```

Рис. 8.1

```
===== Generation 57
Evaluated 306 individuals
Min = 68.0 , Max = 75.0
Average = 74.02 , Standard deviation = 1.27

===== Generation 58
Evaluated 276 individuals
Min = 69.0 , Max = 75.0
Average = 74.15 , Standard deviation = 1.18

===== Generation 59
Evaluated 288 individuals
Min = 69.0 , Max = 75.0
Average = 74.12 , Standard deviation = 1.24

==== End of evolution

Best individual:
 [1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0
, 1, 0, 0, 0, 1]

Number of ones: 45
```

Рис. 8.2

В соответствии с рис. 8.2 процесс эволюции завершился через 60 поколений (отсчет поколений ведется с нуля). В последней строке вывода отображена информация о наилучшем индивидууме. Он содержит 45 единиц, что является для нас дополнительным подтверждением правильности полученного результата, поскольку именно значение 45 было установлено в качестве целевого для нашей оценочной функции.

Визуализация хода эволюции

Давайте посмотрим, как можно визуализировать эволюционный процесс. Для этой цели авторы библиотеки DEAP использовали *стратегию эволюции посредством адаптации ковариационной матрицы* (Covariance Matrix Adaptation Evolution Strategy — CMA-ES). Это эволюционный алгоритм, который используется для решения нелинейных задач в непрерывной области. Метод CMA-ES робастен, хорошо изучен и считается последним достижением в мире эволюционных алгоритмов. Рассмотрим, как он работает, обратившись к исходному коду. Приведенный ниже код — это незначительно измененный вариант кода, представленного в библиотеке DEAP.

Создайте новый файл Python и выполните следующие операции импорта.

```
import numpy as np
import matplotlib.pyplot as plt
from deap import algorithms, base, benchmarks, \ cma, creator, tools
```

Определим функцию для создания набора инструментов. Мы определим функцию `FitnessMin`, используя отрицательные веса.

```
# Функция для создания набора инструментов


def create_toolbox(strategy):
    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
    creator.create("Individual", list, \ fitness=creator.FitnessMin)
```

Создадим набор инструментов и зарегистрируем оценочную функцию.

```
    toolbox = base.Toolbox()
    toolbox.register("evaluate", benchmarks.rastrigin)

    # Затравочное значение для генератора случайных чисел
    np.random.seed(7)
```

Зарегистрируем методы `generate` и `update`. Они необходимы для работы в рамках парадигмы "генерация — обновление", в соответствии с которой мы

генерируем популяцию, исходя из стратегии, и эта стратегия обновляется на основании популяции.

```
    toolbox.register("generate", strategy.generate,
                     creator.Individual)
    toolbox.register("update", strategy.update)

    return toolbox
```

Определим основную функцию. Начнем с определения количеств индивидуумов и поколений.

```
if __name__ == "__main__":
    # Размеры задачи
    num_individuals = 10
    num_generations = 125
```

Прежде чем начать процесс эволюции, мы должны определить стратегию.

```
    # Создание стратегии с использованием алгоритма CMA-ES
    strategy = cma.Strategy(centroid=[5.0]*num_individuals,
            sigma=5.0, lambda_=20*num_individuals)
```

Создадим набор инструментов на базе стратегии.

```
    # Создание набора инструментов на базе
    # определенной выше стратегии
    toolbox = create_toolbox(strategy)
```

Создадим объект `HallOfFame`, который содержит наилучшие из индивидуумов за все время существования популяции. Этот объект в любой момент времени поддерживается в отсортированном состоянии. Благодаря этому его первым элементом всегда является индивидуум с наилучшим значением параметра приспособленности на протяжении всего эволюционного процесса.

```
    # Создание объекта HallOfFame
    hall_of_fame = tools.HallOfFame(1)
```

Зарегистрируем статистики, используя метод `Statistics`.

```
    # Регистрация соответствующих статистик
    stats = tools.Statistics(lambda x: x.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)
```

Определим журнал, в который будет записываться ход эволюции. В основном он представляет собой список словарей.

```
logbook = tools.Logbook()
logbook.header = "gen", "evals", "std", "min", "avg", "max"
```

Определим объекты для сбора всех данных.

```
# Объекты, предназначенные для сбора данных
sigma = np.ndarray((num_generations, 1))
axis_ratio = np.ndarray((num_generations, 1))
diagD = np.ndarray((num_generations, num_individuals))
fbest = np.ndarray((num_generations,1))
best = np.ndarray((num_generations, num_individuals))
std = np.ndarray((num_generations, num_individuals))
```

Выполним итерации по поколениям.

```
for gen in range(num_generations):
# Генерирование новой популяции
    population = toolbox.generate()
```

Проведем вычисления для индивидуумов, используя функцию приспособленности.

```
    # Выполним итерации по индивидуумам
    fitnesses = toolbox.map(toolbox.evaluate, population)
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit
```

Обновим стратегию, исходя из популяции.

```
    # Обновим стратегию на основании
    # информации об индивидуумах
    toolbox.update(population)
```

Обновим объект `HallOfFame` и статистики для текущего поколения индивидуумов.

```
    # Обновим объект HallOfFame и статистики для
    # вычисляемой в настоящий момент популяции
    hall_of_fame.update(population)
    record = stats.compile(population)
    logbook.record(evals=len(population), gen=gen, **record)
    print(logbook.stream)
```

Сохраним данные для построения графика.

```
# Сохранение данных для построения графика
sigma[gen] = strategy.sigma
axis_ratio[gen] = max(strategy.diagD)**2/min(strategy.diagD)**2
diagD[gen, :num_individuals] = strategy.diagD**2
fbest[gen] = hall_of_fame[0].fitness.values
best[gen, :num_individuals] = hall_of_fame[0]
std[gen, :num_individuals] = np.std(population, axis=0)
```

Определим ось x и отобразим статистики на графике.

```
# По оси X откладывается число оценок
x = list(range(0, strategy.lambda_ * num_generations,
        strategy.lambda_))
avg, max_, min_ = logbook.select("avg", "max", "min")
plt.figure()
plt.semilogy(x, avg, "--b")
plt.semilogy(x, max_, "--b")
plt.semilogy(x, min_, "-b")
plt.semilogy(x, fbest, "-c")
plt.semilogy(x, sigma, "-g")
plt.semilogy(x, axis_ratio, "-r")
plt.grid(True)
plt.title("Синий: f-значения, зеленый: sigma, красный: axis_ratio")
```

Построим график хода процесса.

```
plt.figure()
plt.plot(x, best)
plt.grid(True)
plt.title("Объектные переменные")

plt.figure()
plt.semilogy(x, diagD)
plt.grid(True)
plt.title("Масштабирование (все основные оси)")

plt.figure()
plt.semilogy(x, std)
plt.grid(True)
plt.title("Стандартные отклонения (все координаты)")
plt.show()
```

Полный код этого примера содержится в файле `visualization.py`. В процессе выполнения этого кода на экране отобразятся четыре графика. На первом экранном снимке представлены различные параметры (рис. 8.3).

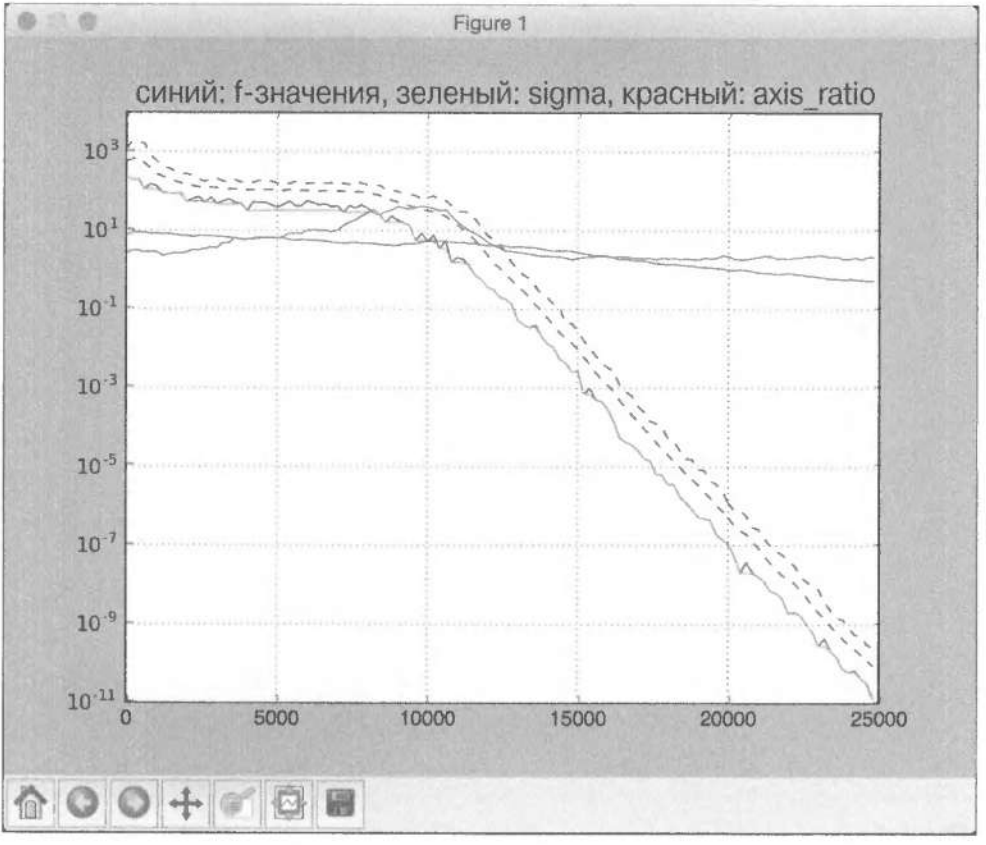

Рис. 8.3

На втором экранном снимке представлены объектные переменные (рис. 8.4).

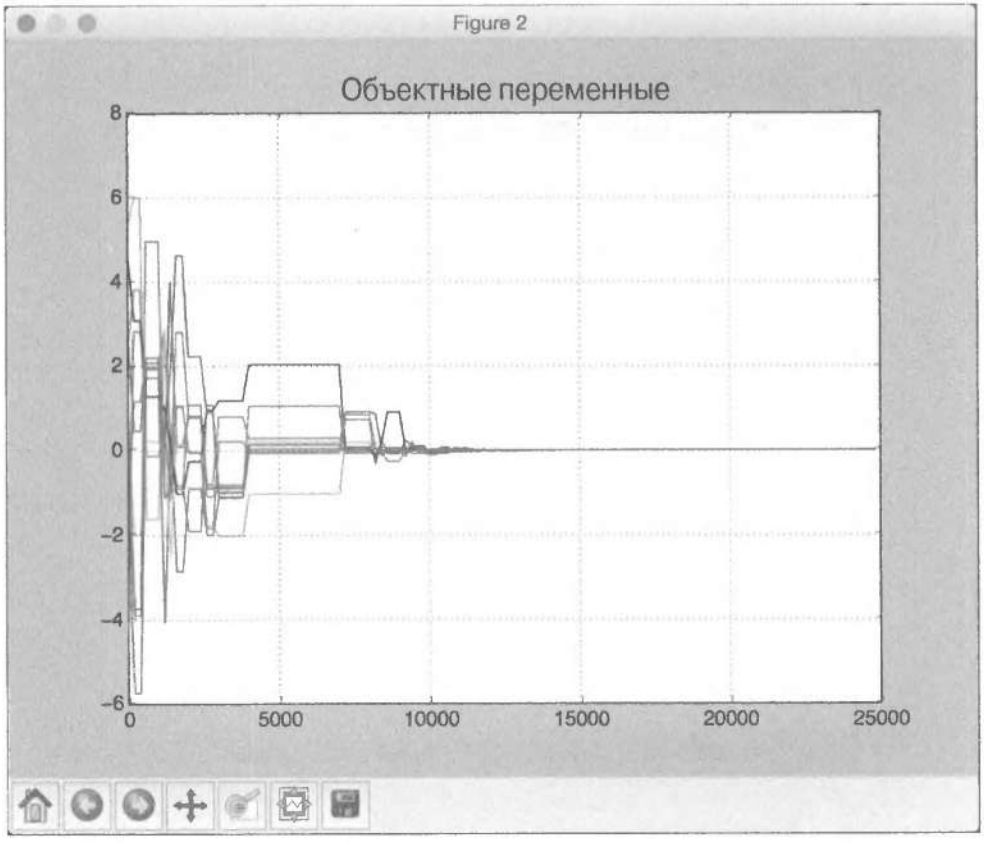

Рис. 8.4

На третьем экранном снимке представлены результаты масштабирования (рис. 8.5).

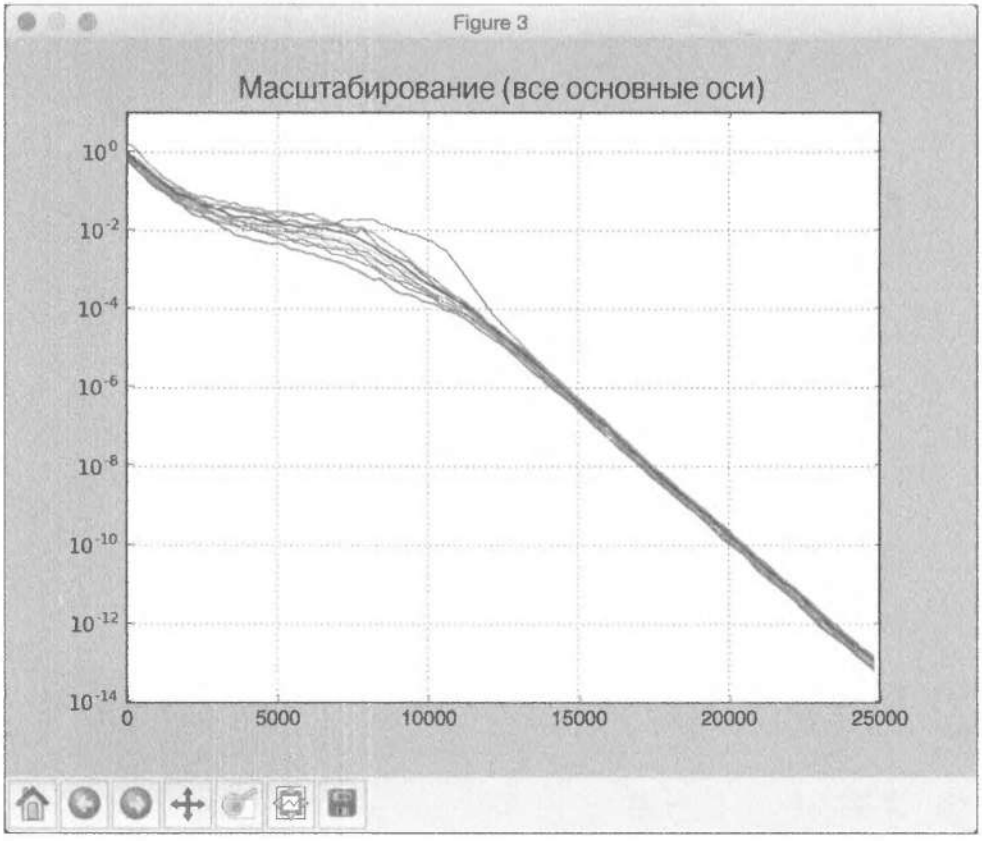

Рис. 8.5

На четвертом экранном снимке представлены стандартные отклонения (рис. 8.6).

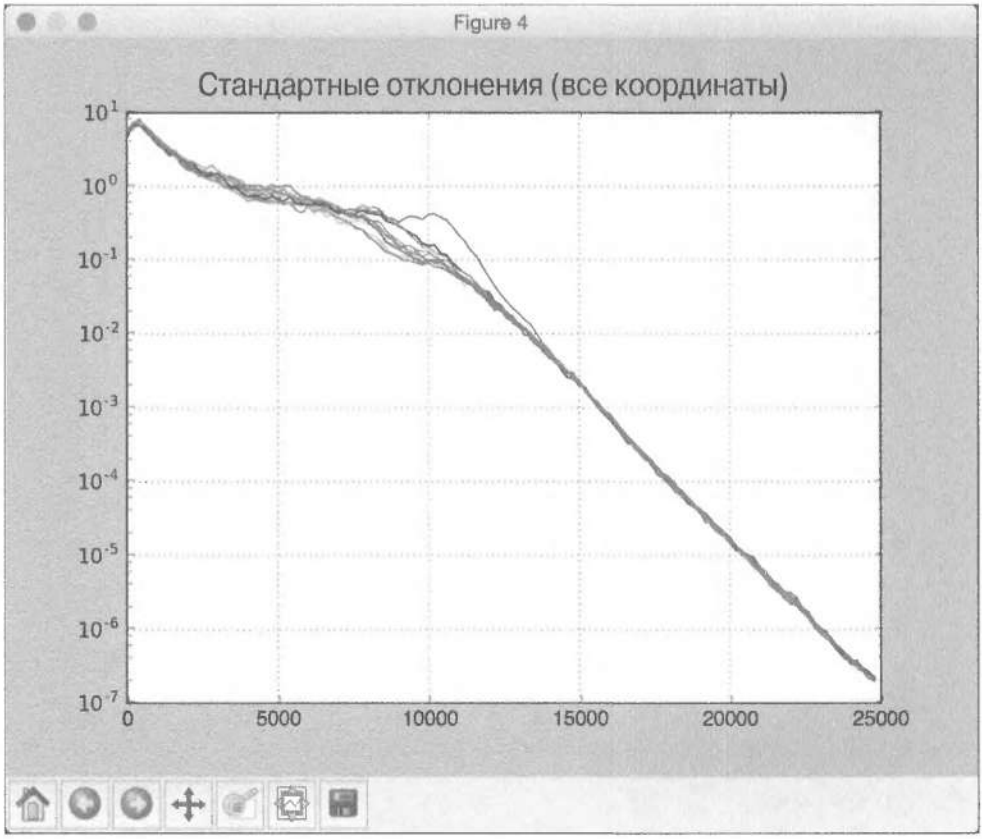

Рис. 8.6

В окне терминала будет выведена информация о ходе процесса. Начальный фрагмент этой информации будет выглядеть так (рис. 8.7).

```
gen     evals   std     min     avg     max
0       200     188.36  217.082 576.281 1199.71
1       200     250.543 196.583 659.389 1869.02
2       200     273.081 199.455 683.641 1770.65
3       200     215.326 111.298 503.933 1579.3
4       200     133.046 149.47  373.124 790.899
5       200     75.4405 131.117 274.092 585.433
6       200     61.2622 91.7121 232.624 426.666
7       200     49.8303 88.8185 201.117 373.543
8       200     39.9533 85.0531 178.645 326.209
9       200     31.3781 87.4824 159.211 261.132
10      200     31.3488 54.0743 144.561 274.877
11      200     30.8796 63.6032 136.791 240.739
12      200     24.1975 70.4913 125.691 190.684
13      200     21.2274 50.6409 122.293 177.483
14      200     25.4931 67.9873 124.132 199.296
15      200     26.9804 46.3411 119.295 205.331
16      200     24.8993 56.0033 115.614 176.702
17      200     21.9789 61.4999 113.417 170.156
18      200     21.2823 50.2455 112.419 190.677
19      200     22.5016 48.153  111.543 166.2
20      200     21.1602 32.1864 106.044 171.899
21      200     23.3864 52.8601 107.301 163.617
22      200     23.1008 51.1226 109.628 185.777
23      200     22.0836 51.3058 106.402 179.673
```

Рис. 8.7

Завершающий фрагмент этой информации будет иметь следующий вид (рис. 8.8).

100	200	2.38865e-07	1.12678e-07	5.18814e-07	1.23527e-06
101	200	1.49444e-07	5.56979e-08	3.3199e-07	7.98774e-07
102	200	1.11635e-07	2.07109e-08	2.41361e-07	7.96738e-07
103	200	9.50257e-08	3.69117e-08	1.94641e-07	5.75896e-07
104	200	5.63849e-08	2.09827e-08	1.26148e-07	2.887e-07
105	200	4.42488e-08	1.64212e-08	8.6972e-08	2.58639e-07
106	200	2.34933e-08	1.28302e-08	5.47789e-08	1.54658e-07
107	200	1.74434e-08	7.13185e-09	3.64705e-08	9.88235e-08
108	200	1.17157e-08	6.32208e-09	2.54673e-08	7.13075e-08
109	200	8.73027e-09	4.60369e-09	1.79681e-08	5.88066e-08
110	200	6.39874e-09	1.92573e-09	1.43229e-08	4.00087e-08
111	200	5.31196e-09	2.05551e-09	1.13736e-08	3.16793e-08
112	200	3.15607e-09	1.72427e-09	7.28548e-09	1.67727e-08
113	200	2.3789e-09	1.01164e-09	5.01177e-09	1.24541e-08
114	200	1.38424e-09	6.43112e-10	2.94696e-09	9.25819e-09
115	200	1.04172e-09	2.87571e-10	2.06068e-09	7.90436e-09
116	200	6.08685e-10	4.32905e-10	1.4704e-09	3.80221e-09
117	200	4.51515e-10	2.1538e-10	9.23627e-10	2.2759e-09
118	200	2.77204e-10	1.46869e-10	6.3507e-10	1.44637e-09
119	200	2.06475e-10	7.54881e-11	4.41427e-10	1.33167e-09
120	200	1.3138e-10	5.97282e-11	2.98116e-10	8.60453e-10
121	200	9.52385e-11	6.753e-11	2.32358e-10	5.45441e-10
122	200	7.55001e-11	4.1851e-11	1.72688e-10	5.05054e-10
123	200	5.52125e-11	3.2216e-11	1.23505e-10	3.10081e-10
124	200	4.38068e-11	1.32871e-11	8.94929e-11	2.57202e-10

Рис. 8.8

Из этого рисунка видно, что в ходе эволюции значения уменьшаются. Это указывает на то, что процесс сходится.

Решение задачи символической регрессии

Рассмотрим, как применить генетическое программирование для решения задач символической регрессии. Важно понимать, что генетическое программирование — это не то же самое, что генетические алгоритмы. Генетическое программирование — это тип эволюционного алгоритма, в котором решения выступают в виде компьютерных программ. По сути, в каждом поколении индивидуумами будут компьютерные программы, уровни приспособленности которых соответствуют их способности решать задачи. Эти программы изменяются на каждой итерации генетическими алгоритмами. Резюмируя, можно сказать, что генетическое программирование — это применение генетических алгоритмов.

Перейдем к рассмотрению задачи символической регрессии. Предположим, у нас имеется полиномиальное выражение, которое мы должны

аппроксимировать. Это классическая регрессионная задача, в которой мы пытаемся дать приближенную оценку базовой функции. В данном примере мы будем использовать следующее выражение:

$$f(x) = 2x^3 - 3x^2 + 4x - 1$$

Обсуждаемый здесь код является вариантом решения задачи символической регрессии, приведенным в библиотеке DEAP. Создайте новый файл Python и импортируйте следующие пакеты.

```
import operator
import math
import random

import numpy as np
from deap import algorithms, base, creator, tools, gp
```

Создадим оператор деления, способный корректно обрабатывать ошибку деления на нуль.

```
# Определение новых функций
def division_operator(numerator, denominator):
    if denominator == 0:
    return 1

    return numerator / denominator
```

Определим оценочную функцию, которую будем использовать для вычисления приспособленности. Чтобы выполнить необходимые вычисления для входного индивидуума, мы должны определить вызываемую функцию.

```
# Определение оценочной функции
def eval_func(individual, points):
    # Преобразование дерева выражений в вызываемую функцию
    func = toolbox.compile(expr=individual)
```

Вычислим среднеквадратическую ошибку (MSE) для разности между определенной перед этим функцией и оригинальным выражением.

```
    # Вычисление среднеквадратической ошибки
    mse = ((func(x) - (2 * x**3 - 3 * x**2 - 4 * x + 1))**2
           for x in points)
    return math.fsum(mse) / len(points),
```

Определим функцию для создания набора инструментов. В данном случае, чтобы создать набор инструментов, мы должны определить набор примитивов. Эти примитивы, по сути, являются операторами, которые будут

использоваться в процессе эволюции. Они служат строительными кирпичиками для индивидуумов. В качестве примитивов мы будем использовать базовые арифметические функции.

```
# Функция для создания набора инструментов
def create_toolbox():
    pset = gp.PrimitiveSet("MAIN", 1)
    pset.addPrimitive(operator.add, 2)
    pset.addPrimitive(operator.sub, 2)
    pset.addPrimitive(operator.mul, 2)
    pset.addPrimitive(division_operator, 2)
    pset.addPrimitive(operator.neg, 1)
    pset.addPrimitive(math.cos, 1)
    pset.addPrimitive(math.sin, 1)
```

Далее мы должны определить "эфемерную" константу. Это специальный терминальный тип, не имеющий фиксированного значения. Когда программа присоединяет эфемерную константу к дереву, выполняется функция. Затем результат вставляется в дерево в качестве терминальной константы. Терминальные константы могут иметь значения -1, 0 или 1.

```
    pset.addEphemeralConstant("rand101", lambda: random.randint(-1,1))
```

Именами аргументов по умолчанию являются ARGx. Переименуем аргумент в x. Поступать так необязательно, но подобная возможность иногда оказывается очень удобной.

```
    pset.renameArguments(ARG0='x')
```

Нам нужно определить два объектных типа: `fitness` и `individual`. Используем для этого объект `creator`.

```
    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
    creator.create("Individual", gp.PrimitiveTree,
                   fitness=creator.FitnessMin)
```

Создадим набор инструментов и зарегистрируем функции. Процесс регистрации выполняется аналогично тому, как это делалось в предыдущих разделах.

```
    toolbox = base.Toolbox()

    toolbox.register("expr", gp.genHalfAndHalf, pset=pset,
                     min_=1, max_=2)
    toolbox.register("individual", tools.initIterate,
                     creator.Individual, toolbox.expr)
```

```
    toolbox.register("population", tools.initRepeat, list,
                     toolbox.individual)
    toolbox.register("compile", gp.compile, pset=pset)
    toolbox.register("evaluate", eval_func, points=[x/10.
                     for x in range(-10,10)])
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("mate", gp.cxOnePoint)
    toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
    toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut,
                     pset=pset)

    toolbox.decorate("mate",
gp.staticLimit(key=operator.attrgetter("height"), max_value=17))
    toolbox.decorate("mutate",
gp.staticLimit(key=operator.attrgetter("height"), max_value=17))

    return toolbox
```

Определим основную функцию и начнем с предоставления затравочного значения для генератора случайных чисел.

```
if __name__ == "__main__":
    random.seed(7)
```

Создадим объект `toolbox`.

```
toolbox = create_toolbox()
```

Определим начальную популяцию с помощью метода, предоставляемого объектом `toolbox`. Мы будем использовать `450` индивидуумов. Можете смело поэкспериментировать с ним, задавая для него различные значения. Кроме того, определим объект `hall_of_fame`.

```
    population = toolbox.population(n=450)
    hall_of_fame = tools.HallOfFame(1)
```

При создании генетических алгоритмов полезно использовать статистики.

```
    stats_fit = tools.Statistics(lambda x: x.fitness.values)
    stats_size = tools.Statistics(len)
```

Зарегистрируем статистики, используя объекты, которые были определены ранее.

```
    mstats = tools.MultiStatistics(fitness=stats_fit,
                                   size=stats_size)
```

```
mstats.register("avg", np.mean)
mstats.register("std", np.std)
mstats.register("min", np.min)
mstats.register("max", np.max)
```

Определим вероятности скрещивания и мутации, а также количество поколений.

```
probab_crossover = 0.4
probab_mutate = 0.2
num_generations = 60
```

Выполним эволюционный алгоритм, используя определенные выше параметры.

```
population, log = algorithms.eaSimple(population, toolbox,
    probab_crossover, probab_mutate, num_generations,
    stats=mstats, halloffame=hall_of_fame, verbose=True)
```

Полный код этого примера содержится в файле `symbol_regression.py`. В процессе выполнения кода в окне терминала отобразится информация, начальный фрагмент которой выглядит так (рис. 8.9).

		fitness				size			
gen	nevals	avg	max	min	std	avg	max	min	std
0	450	18.6918	47.1923	7.39087	6.27543	3.73556	7	2	1.62449
1	251	15.4572	41.3823	4.46965	4.54993	3.80222	12	1	1.81316
2	236	13.2545	37.7223	4.46965	4.06145	3.96889	12	1	1.98861
3	251	12.2299	60.828	4.46965	4.70055	4.19556	12	1	1.9971
4	235	11.001	47.1923	4.46965	4.48841	4.84222	13	1	2.17245
5	229	9.44483	31.478	4.46965	3.8796	5.56	19	1	2.43168
6	225	8.35975	22.0546	3.02133	3.40547	6.38889	15	1	2.40875
7	237	7.99309	31.1356	1.81133	4.08463	7.14667	16	1	2.57782
8	224	7.42611	359.418	1.17558	17.0167	8.33333	19	1	3.11127
9	237	5.70308	24.1921	1.17558	3.71991	9.64444	23	1	3.31365
10	254	5.27991	30.4315	1.13301	4.13556	10.5089	25	1	3.51898

Рис. 8.9

В конце эволюции будет выведена следующая информация (рис. 8.10).

36	209	1.10464	22.0546	0.0474957	2.71898	26.4867	46	1	5.23289
37	258	1.61958	86.0936	0.0382386	6.1839	27.2111	45	3	4.75557
38	257	2.03651	70.4768	0.0342642	5.15243	26.5311	49	1	6.22327
39	235	1.95531	185.328	0.0472693	9.32516	26.9711	48	1	6.00345
40	234	1.51403	28.5529	0.0472693	3.24513	26.6867	52	1	5.39811
41	230	1.4753	70.4768	0.0472693	5.4607	27.1	46	3	4.7433
42	233	12.3648	4880.09	0.0396503	229.754	26.88	53	1	5.18192
43	251	1.807	86.0936	0.0396503	5.85281	26.4889	50	1	5.43741
44	236	9.30096	3481.25	0.0277886	163.888	26.9622	55	1	6.27169
45	231	1.73196	86.7372	0.0342642	6.8119	27.4711	51	2	5.27807
46	227	1.86086	185.328	0.0342642	10.1143	28.0644	56	1	6.10812
47	216	12.5214	4923.66	0.0342642	231.837	29.1022	54	1	6.45898
48	232	14.3469	5830.89	0.0322462	274.536	29.8244	58	3	6.24093
49	242	2.56984	272.833	0.0322462	18.2752	29.9267	51	1	6.31446
50	227	2.80136	356.613	0.0322462	21.0416	29.7978	56	4	6.50275
51	243	1.75099	86.0936	0.0322462	5.70833	29.8089	56	1	6.62379
52	253	10.9184	3435.84	0.0227048	163.602	29.9911	55	1	6.66833
53	243	1.80265	48.0418	0.0227048	4.73856	29.88	55	1	7.33084
54	234	1.74487	86.0936	0.0227048	6.0249	30.6067	55	1	6.85782
55	220	1.58888	31.094	0.0132398	3.82809	30.5644	54	1	6.96669
56	234	1.46711	103.287	0.00766444	6.81157	30.6689	55	3	6.6806
57	250	17.0896	6544.17	0.00424267	308.689	31.1267	60	4	7.25837
58	231	1.66757	141.584	0.00144401	7.35306	32	52	1	7.23295
59	229	2.22325	265.224	0.00144401	13.388	33.5489	64	1	8.38351
60	248	2.60303	521.804	0.00144401	24.7018	35.2533	58	1	7.61506

Рис. 8.10

Создание контроллера интеллектуального робота

В этом разделе демонстрируется применение генетического алгоритма для создания контроллера робота. Предположим, имеется карта с расположенными на ней целевыми объектами (рис. 8.11).

Всего на карте находится 124 таких объекта. Нашей задачей является создание контроллера робота, который будет автоматически обходить карту и поглощать эти объекты. Приведенная ниже программа представляет собой вариант программы "муравьиной логистики", пример которой приведен в библиотеке DEAP.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import copy
import random
from functools import partial

import numpy as np
from deap import algorithms, base, creator, tools, gp
```

Создадим класс, предназначенный для управления роботом.

```
class RobotController(object):
    def __init__(self, max_moves):
        self.max_moves = max_moves
```

```
self.moves = 0
self.consumed = 0
self.routine = None
```

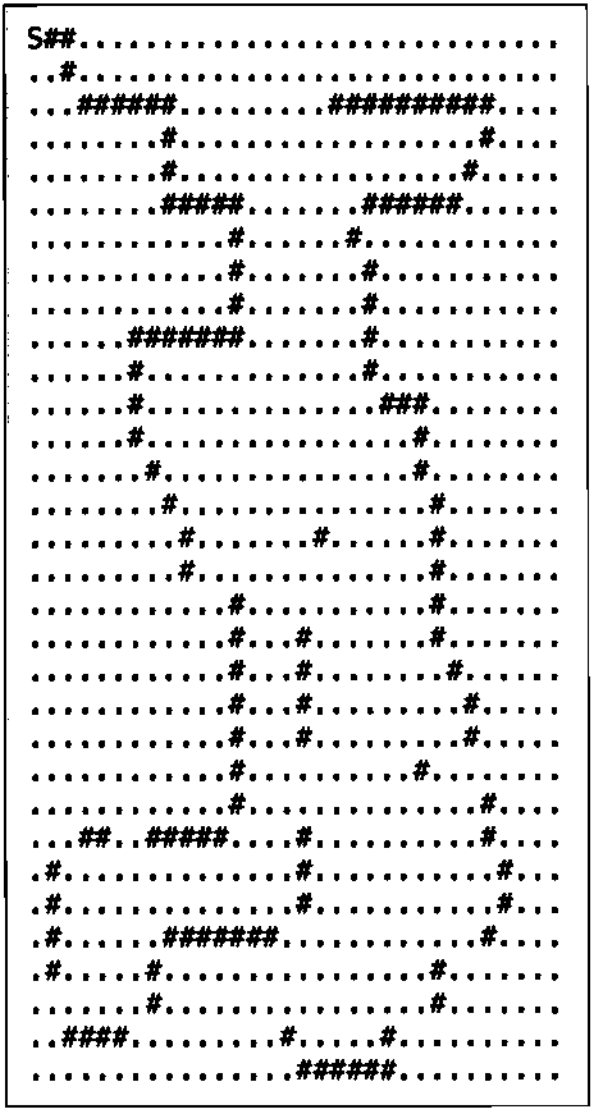

Рис. 8.11

Определим направления и перемещения.

```
self.direction = ["north", "east", "south", "west"]
self.direction_row = [1, 0, -1, 0]
self.direction_col = [0, 1, 0, -1]
```

Определим функциональность сброса параметров.

```
def _reset(self):
    self.row = self.row_start
    self.col = self.col_start
    self.direction = 1
    self.moves = 0
    self.consumed = 0
    self.matrix_exc = copy.deepcopy(self.matrix)
```

Определим условный оператор.

```
def _conditional(self, condition, out1, out2):
    out1() if condition() else out2()
```

Определим оператор левого поворота.

```
def turn_left(self):
    if self.moves < self.max_moves:
        self.moves += 1
        self.direction = (self.direction - 1) % 4
```

Определим оператор правого поворота.

```
def turn_right(self):
    if self.moves < self.max_moves:
        self.moves += 1
        self.direction = (self.direction + 1) % 4
```

Определим метод, управляющий продвижением робота.

```
def move_forward(self):
    if self.moves < self.max_moves:
        self.moves += 1
        self.row = (self.row +
                    self.direction_row[self.direction]) %
                    self.matrix_row
        self.col = (self.col +
                    self.direction_col[self.direction]) %
                    self.matrix_col

        if self.matrix_exc[self.row][self.col] == "target":
            self.consumed += 1

        self.matrix_exc[self.row][self.col] = "passed"
```

Определим метод для обнаружения целевых объектов. Если впереди обнаруживается объект, матрица соответствующим образом обновляется.

```
def sense_target(self):
    ahead_row = (self.row + self.direction_row[self.direction]) %
                 self.matrix_row
    ahead_col = (self.col + self.direction_col[self.direction]) %
                 self.matrix_col
    return self.matrix_exc[ahead_row][ahead_col] == "target"
```

Если впереди обнаруживается объект, создаем соответствующую функцию и возвращаем ее.

```
def if_target_ahead(self, out1, out2):
    return partial(self._conditional, self.sense_target,
                   out1, out2)
```

Определим метод для выполнения этой функции.

```
def run(self,routine):
    self._reset()
    while self.moves < self.max_moves:
        routine()
```

Определим функцию для прохождения заданной карты. Символы # обозначают расположенные на карте целевые объекты, а символы S — отправную точку.

```
def traverse_map(self, matrix):
    self.matrix = list()
    for i, line in enumerate(matrix):
        self.matrix.append(list())

        for j, col in enumerate(line):
            if col == "#":
                self.matrix[-1].append("target")

            elif col == ".":
                self.matrix[-1].append("empty")

            elif col == "S":
                self.matrix[-1].append("empty")

            self.row_start = self.row = i
                self.col_start = self.col = j
                self.direction = 1
```

```
        self.matrix_row = len(self.matrix)
        self.matrix_col = len(self.matrix[0])
        self.matrix_exc = copy.deepcopy(self.matrix)
```

Определим класс, который генерирует функции в зависимости от количества входных аргументов.

```
class Prog(object):
    def _progn(self, *args):
        for arg in args:
            arg()

    def prog2(self, out1, out2):
        return partial(self._progn, out1, out2)

    def prog3(self, out1, out2, out3):
        return partial(self._progn, out1, out2, out3)
```

Определим оценочную функцию для каждого индивидуума.

```
def eval_func(individual):
    global robot, pset

    # Преобразование дерева выражений в функциональный
    # код Python
    routine = gp.compile(individual, pset)
```

Выполним текущую программу.

```
    # Выполнение сгенерированной программы
    robot.run(routine)
    return robot.consumed,
```

Определим функцию для создания набора инструментов и добавим примитивы.

```
def create_toolbox():
    global robot, pset

    pset = gp.PrimitiveSet("MAIN", 0)
    pset.addPrimitive(robot.if_target_ahead, 2)
    pset.addPrimitive(Prog().prog2, 2)
    pset.addPrimitive(Prog().prog3, 3)
    pset.addTerminal(robot.move_forward)
    pset.addTerminal(robot.turn_left)
    pset.addTerminal(robot.turn_right)
```

Создадим объектные типы, используя функцию приспособленности.

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", gp.PrimitiveTree,
               fitness=creator.FitnessMax)
```

Создадим набор инструментов и зарегистрируем все операторы.

```
toolbox = base.Toolbox()

# Генератор атрибутов
toolbox.register("expr_init", gp.genFull, pset=pset, min_=1,
                 max_=2)
# Инициализаторы структур
toolbox.register("individual", tools.initIterate,
                 creator.Individual, toolbox.expr_init)
toolbox.register("population", tools.initRepeat, list,
                 toolbox.individual)

toolbox.register("evaluate", eval_func)
toolbox.register("select", tools.selTournament, tournsize=7)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut,
                 pset=pset)

return toolbox
```

Определим функцию main и начнем с предоставления затравочного значения для генератора случайных чисел.

```
if __name__ == "__main__":
    global robot

    # Затравочное значение для генератора случайных чисел
    random.seed(7)
```

Создадим объект контроллера робота, используя параметр инициализации.

```
    # Определим максимальное количество перемещений
    max_moves = 750
    # Создадим объект робота
    robot = RobotController(max_moves)
```

Создадим набор инструментов, используя определенную ранее функцию.

```
# Создание набора инструментов
toolbox = create_toolbox()
```

Прочитаем данные карты из входного файла.

```
# Чтение данных карты
with open('target_map.txt', 'r') as f:
    robot.traverse_map(f)
```

Определим популяцию с 400 индивидуумами и объект `hall_of_fame`.

```
# Определение популяции и объекта hall_of_fame
population = toolbox.population(n=400)
hall_of_fame = tools.HallOfFame(1)
```

Зарегистрируем статистики.

```
# Регистрация статистик
stats = tools.Statistics(lambda x: x.fitness.values)
stats.register("avg", np.mean)
stats.register("std", np.std)
stats.register("min", np.min)
stats.register("max", np.max)
```

Определим вероятности скрещивания и мутации, а также количество поколений.

```
# Определение параметров
probab_crossover = 0.4
probab_mutate = 0.3
num_generations = 50
```

Выполним эволюционный алгоритм, используя определенные ранее параметры.

```
# Выполнение алгоритма для решения задачи
algorithms.eaSimple(population, toolbox, probab_crossover,
                    probab_mutate, num_generations, stats,
                    halloffame=hall_of_fame)
```

Полный код этого примера содержится в файле `robot.py`. В процессе выполнения этого кода в окне вашего терминала отобразится информация, начальный фрагмент которой выглядит так (рис. 8.12).

```
gen     nevals  avg     std     min     max
0       400     1.4875  4.37491 0       62
1       231     4.285   7.56993 0       73
2       235     10.8925 14.8493 0       73
3       231     21.72   22.1239 0       73
4       238     29.9775 27.7861 0       76
5       224     37.6275 31.8698 0       76
6       231     42.845  33.0541 0       80
7       223     43.55   33.9369 0       83
8       234     44.0675 34.5201 0       83
9       231     49.2975 34.3065 0       83
10      249     47.075  36.4106 0       93
11      222     52.7925 36.2826 0       97
12      248     51.0725 37.2598 0       97
13      234     54.01   37.4614 0       97
14      229     59.615  37.7894 0       97
15      228     63.3    39.8205 0       97
16      220     64.605  40.3962 0       97
17      236     62.545  40.5607 0       97
18      233     67.99   38.9033 0       97
19      236     66.4025 39.6574 0       97
20      221     69.785  38.7117 0       97
21      244     65.705  39.0957 0       97
22      230     70.32   37.1206 0       97
23      241     67.3825 39.4028 0       97
```

Рис. 8.12

Конечный фрагмент информации будет иметь следующий вид (рис. 8.13).

Резюме

Из этой главы вы узнали о том, что такое генетические алгоритмы, и познакомились с соответствующими понятиями. Мы обсудили эволюционные алгоритмы и генетическое программирование и их связь с генетическими алгоритмами. Мы рассмотрели фундаментальные понятия генетических алгоритмов, включая популяцию, кроссовер, мутацию, отбор и функцию приспособленности. Вы узнали о том, как генерировать битовый образ с заданными параметрами. Мы обсудили визуализацию эволюционного процесса с помощью алгоритма CMA-ES. Вы узнали о способах решения задач символической регрессии в рамках этой парадигмы. Наконец, мы использовали эти концепции для создания контроллера робота, обходящего карту и

поглощающего целевые объекты. В следующей главе будет рассказано о методе обучения с подкреплением и продемонстрировано его применение для создания интеллектуального агента.

```
26      214     71.505  36.964  0       97
27      246     72.72   37.1637 0       97
28      238     73.5975 36.5385 0       97
29      239     76.405  35.5696 0       97
30      246     78.6025 33.4281 0       97
31      240     74.83   36.5157 0       97
32      216     80.2625 32.6659 0       97
33      220     80.6425 33.0933 0       97
34      247     78.245  34.6022 0       97
35      241     81.22   32.1885 0       97
36      234     83.6375 29.0002 0       97
37      228     82.485  31.7354 0       97
38      219     83.4625 30.0592 0       97
39      212     88.64   24.2702 0       97
40      231     86.7275 27.0879 0       97
41      229     89.1825 23.8773 0       97
42      216     87.96   25.1649 0       97
43      218     86.85   27.1116 0       97
44      236     88.78   23.7278 0       97
45      225     89.115  23.4212 0       97
46      232     88.5425 24.187  0       97
47      245     87.7775 25.3909 0       97
48      231     87.78   26.3786 0       97
49      238     88.8525 24.5115 0       97
50      233     87.82   25.4164 1       97
```

Рис. 8.13

9

Создание игр с помощью искусственного интеллекта

В этой главе рассказывается о создании игр с помощью искусственного интеллекта. Вы узнаете о применении алгоритмов поиска для разработки эффективных стратегий выигрыша. Затем мы используем эти алгоритмы с целью создания интеллектуальных роботов для различных игр.

К концу главы вы освоите следующие темы:

- использование поисковых алгоритмов в играх;
- комбинаторный поиск;
- алгоритм MiniMax;
- альфа-бета-отсечение;
- алгоритм NegaMax;
- игра Last Coin Standing;
- создание робота для игры Tic-Tac-Toe;
- создание двух роботов, играющих друг против друга в игру Connect Four;
- создание двух роботов, играющих друг против друга в игру Hexapawn.

Использование поисковых алгоритмов в играх

Для определения стратегии в играх используются алгоритмы поиска. Поисковый алгоритм просматривает возможные варианты ходов и выбирает из них наилучший. При этом приходится учитывать ряд различных параметров: скорость, точность, сложность и т.п. Алгоритмы анализируют все действия, доступные в текущей игровой ситуации, и используют эту информацию для просчитывания последующих действий. Целью подобных алгоритмов является нахождение оптимального набора ходов, обеспечивающих

удовлетворение условий выигрыша. У каждой игры имеется свой набор выигрышных условий. Именно эти условия используются алгоритмами для выработки очередного набора ходов.

Приведенное в предыдущем абзаце описание идеально подходит для игр, в которых у вас отсутствует противник. В случае нескольких игроков не все так просто. В качестве примера рассмотрим игру, в которой участвуют два игрока. На каждый ход, сделанный одним игроком, его противник ответит ходом, препятствующим первому игроку в достижении его цели. Поэтому, если поисковый алгоритм найдет набор ходов, оптимальный для текущего состояния игры, простого выполнения намеченной к этому времени последовательности ходов будет недостаточно, поскольку противник в любой момент может нарушить ее. По сути, это означает, что алгоритмы поиска должны осуществлять непрерывную переоценку ситуации после каждого хода.

Давайте обсудим, каким образом компьютер воспринимает любую конкретную игру. Игру можно рассматривать как дерево поиска. Каждый узел этого дерева представляет будущее состояние. Например, если вы играете в крестики-нолики, то можете сконструировать дерево, представляющее различные игровые ходы. Мы начинаем с корня этого дерева, который служит в игре отправной точкой. Этот узел будет иметь несколько дочерних узлов, представляющих различные возможные ходы. В свою очередь, эти узлы будут иметь свои дочерние узлы, представляющие состояния игры после ходов противника. Терминальные (оконечные) узлы дерева представляют конечные результаты игры после различных серий ходов. Игра заканчивается либо вничью, либо выигрышем одного из игроков. Поисковые алгоритмы просматривают это дерево для принятия решений на каждом этапе игры.

Комбинаторный поиск

Несмотря на то что поисковые алгоритмы позволяют добавлять интеллектуальность в игры, у них имеется один недостаток. В таких алгоритмах используется так называемый *исчерпывающий поиск* (exhaustive search), также известный как *поиск методом грубой силы* (brute force search). По сути, такой тип поиска предполагает исследование всего поискового пространства и тестирование всех возможных вариантов решения. Это означает, что в худшем случае, прежде чем будет найдено правильное решение, мы должны перебрать все без исключения возможные варианты.

Однако по мере усложнения игр мы не можем полагаться на метод грубой силы ввиду огромного числа вариантов, подлежащих перебору. При таком подходе проведение необходимых вычислений очень быстро становится практически неосуществимым. Для преодоления этой проблемы задействуют комбинаторный поиск. В подобном случае эффективность алгоритмов,

исследующих пространства решений, повышают за счет использования эвристик или уменьшения размера поискового пространства. В частности, применение этого метода оказалось чрезвычайно полезным в таких играх, как шахматы или го. Комбинаторный подход эффективно работает за счет использования *стратегий отсечения* (pruning strategies). Эти стратегии позволяют избежать тестирования всех возможных решений за счет исключения тех из них, которые заведомо являются неправильными, что обеспечивает экономию времени и усилий.

Алгоритм MiniMax

Теперь, когда мы вкратце обсудили принципы комбинаторного поиска, давайте поговорим об эвристиках, которые используются алгоритмами, реализующими этот вид поиска. Эвристики ускоряют стратегию поиска, и алгоритм MiniMax является одной из стратегий, используемых комбинаторным поиском. Если два игрока играют друг против друга, то фактически они преследуют противоположные цели. Поэтому, чтобы выиграть, каждая из сторон должна прогнозировать действия другой стороны. Алгоритм MiniMax пытается достигнуть этого посредством стратегии, заключающейся в том, чтобы минимизировать функцию, которую противная сторона пытается максимизировать.

Как мы знаем, метод грубой силы нам не подходит. Компьютер не в состоянии перебрать все возможные состояния и выбрать наилучшую возможную серию ходов, обеспечивающих выигрыш в игре. Компьютер может лишь оптимизировать ходы, исходя из текущего состояния, используя эвристику. Он конструирует дерево, начиная с самого низа. Компьютер вычисляет, какие ходы будут выгодны его оппоненту. В двух словах: компьютеру известно, какие действия собирается предпринять оппонент, на основании тех соображений, что оппонент будет делать ходы, которые наиболее выгодны для него и поэтому наименее выгодны для компьютера. Результатом является один из терминальных узлов дерева, и компьютер использует эту позицию для проведения обратного поиска. Каждому варианту, доступному для компьютера, может быть приписано некоторое значение, и он предпринимает действие, которому соответствует наибольшее значение.

Альфа-бета-отсечение

Стратегия поиска MiniMax эффективна, но она по-прежнему включает исследование частей дерева, являющихся нерелевантными. Рассмотрим дерево, в котором должен выполняться поиск решений. Если в каком-то узле мы обнаруживаем признаки того, что в данном поддереве решения отсутствуют,

то в вычислении этого дерева нет никакой необходимости. Однако поиск MiniMax немного консервативен, поэтому он продолжает исследовать данное поддерево.

Мы должны быть разумными в этом отношении и избегать выполнения поиска в подобных частях дерева. Такой процесс называется *отсечением*, и альфа-бета-отсечение является его разновидностью, которая используется для того, чтобы избегать поиска в частях дерева, не содержащих решения.

В стратегии альфа-бета-отсечения параметры Alpha и Beta относятся к двум границам, которые используются в процессе вычислений. Значения этих параметров ограничивают набор возможных решений. Они определяются на основании информации об уже исследованных разделах дерева. Параметр Alpha — это максимальное значение нижней границы количества возможных решений, а параметр Beta — аналогичная минимальная верхняя граница.

Как уже отмечалось, каждому узлу может быть приписано некоторое значение, исходя из информации о текущем состоянии. Если алгоритм рассматривает любой новый узел как потенциальный путь к решению, то он может выяснить, попадает ли текущая оценка значения данного узла в интервал значений от альфа до бета. В этом и заключается суть отсечения путей поиска.

Алгоритм NegaMax

Алгоритм NegaMax — это разновидность алгоритма MiniMax, которая часто используется на практике. Обычно игры с двумя игроками являются играми с нулевой суммой в том смысле, что проигрыш одного игрока равен выигрышу другого, и наоборот. В алгоритме NegaMax это свойство интенсивно используется для выработки стратегии, увеличивающей шансы выиграть игру.

В терминах игры значение данной позиции для первого игрока равно значению этого же узла для второго игрока, взятому с противоположным знаком. Каждый игрок стремится найти ход, максимизирующий урон противника. Результирующее значение хода должно быть таким, чтобы противник получил наименьшее значение. Эта стратегия работает одинаково хорошо в обоих направлениях в том смысле, что для оценки позиций может использоваться единственный метод. В этом упрощении и заключается преимущество данного подхода по сравнению с алгоритмом MiniMax. Алгоритм MiniMax требует, чтобы первый игрок выбрал ход с максимальным значением, в то время как второй игрок должен выбирать ход с минимальным значением. Здесь также может использоваться альфа-бета-отсечение.

Установка библиотеки easyAI

В этой главе мы будем использовать библиотеку easyAI. Этот фреймворк для разработок в области искусственного интеллекта предоставляет всю необходимую функциональность для создания игр, в которых участвуют два игрока. Вы сможете узнать больше об этой библиотеке на сайте http://zulko.github.io/easyAI.

Установите ее, выполнив в окне своего терминала следующую команду:

```
$ pip3 install easyAI
```

Чтобы использовать некоторые из предварительно созданных программ, нам нужен доступ к определенным файлам. Для упрощения вашей работы в состав файлов примеров, предоставляемых вместе с книгой, включена папка easyAI. Скопируйте эту папку в ту же папку, в которой находятся ваши файлы с кодом. Эта папка в основном содержит подмножество репозитория easyAI на сайте GitHub, доступное по адресу https://github.com/Zulko/easyAI. Вам будет полезно просмотреть этот код, чтобы ближе ознакомиться с ним.

Создание робота для игры Last Coin Standing ("Последняя монета")

В этой игре имеется кучка монет, и игроки поочередно берут из нее по нескольку монет. Количество монет, которые разрешается брать каждому игроку, ограничено сверху и снизу. Цель игры заключается в том, чтобы не оказаться тем игроком, который берет последнюю монету. Эти правила являются вариантом игры Game of Bones ("Игра в кости"), предоставленной в библиотеке easyAI. Ниже продемонстрировано, как создать игру, в которой против пользователя играет компьютер.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player
from easyAI.AI import TT
```

Создадим класс, который будет обрабатывать все операции, выполняемые в процессе игры. Мы наследуем этот класс от базового класса TwoPlayersGame, доступного в библиотеке easyAI. Чтобы этот класс функционировал, как нам нужно, необходимо определить некоторые параметры. Одним из них является переменная players. Об объекте player мы поговорим чуть позже. Создадим класс с помощью следующего кода.

```
class LastCoinStanding(TwoPlayersGame):
    def __init__(self, players):
        # Определение переменной players (необходимый параметр)
        self.players = players
```

Определим того, кто начинает игру. Нумерация игроков начинается с единицы. Поэтому в данном случае игру начинает первый игрок.

```
        # Определение того, кто начинает игру
        # (необходимый параметр)
        self.nplayer = 1
```

Определим количество монет в куче. Для этого параметра вы можете свободно использовать любое значение. В нашем случае мы выберем его равным 25.

```
        # Общее количество монет в куче
        self.num_coins = 25
```

Определим максимальное количество монет, которые разрешается брать при выполнении любого хода. Для этого параметра вы также можете использовать любое значение. В нашем случае мы выберем его равным 4.

```
        # Определение максимального количества монет,
        # которое разрешено брать за один ход
        self.max_coins = 4
```

Определим все возможные ходы. В нашем случае игроки могут брать 1, 2, 3 или 4 монеты за один ход.

```
    # Определение возможных ходов
    def possible_moves(self):
        return [str(x) for x in range(1, self.max_coins + 1)]
```

Определим метод для изъятия монет и подсчета количества монет, остающихся в куче.

```
    # Удаление монет
    def make_move(self, move):
        self.num_coins -= int(move)
```

Проверим, не удалось ли кому-либо выиграть игру, путем проверки количества оставшихся монет.

```
    # Взял ли противник последнюю монету?
    def win(self):
        return self.num_coins <= 0
```

Прекращаем игру, если кто-то выиграл ее.

```
# Прекращение игры в случае чьей-либо победы
def is_over(self):
    return self.win()
```

Вычислим оценку, используя метод `win`. Нам нужно определить этот метод.

```
# Вычисление оценки
def scoring(self):
    return 100 if self.win() else 0
```

Определим метод, отображающий текущее состояние кучи.

```
# Отображение количества монет, оставшихся в куче
def show(self):
    print(self.num_coins, 'coins left in the pile')
```

Определим основную функцию, начав с определения таблицы записи ходов. Такие таблицы используются в играх для хранения позиций и перемещений с целью ускорить выполнение алгоритма. Введите следующий код.

```
if __name__ == "__main__":
    # Определение таблицы ходов
    tt = TT()
```

Определим метод `ttentry` для получения количества монет. Это необязательный метод, используемый для создания строки, которая описывает игру.

```
# Определение метода
LastCoinStanding.ttentry = lambda self: self.num_coins
```

Приступим к реализации игры с помощью ИИ. Для этого применим функцию `id_solve`, использующую итеративное углубление. В основном она определяет, кто может выиграть игру, используя все пути. Данная функция пытается получить ответы на вопросы наподобие "Может ли первый игрок обеспечить себе победу, играя идеально?" или "Будет ли компьютер всегда проигрывать, играя против идеального соперника?"

Метод `id_solve` несколько раз исследует различные варианты в алгоритме NegaMax игры. Он всегда начинает с исходного состояния игры и продолжает анализ, переходя на все более глубокие уровни. Так будет продолжаться до тех пор, пока оценка не укажет на выигрыш или проигрыш одного из игроков. Второй аргумент этого метода принимает список уровней глубины, которые будут испытываться. В нашем случае метод будет испытывать все уровни от 2 до 20.

```
# Решение задачи
result, depth, move = id_solve(LastCoinStanding,
        range(2, 20), win_score=100, tt=tt)
print(result, depth, move)
```

Начнем игру против компьютера.

```
# Начало игры
game = LastCoinStanding([AI_Player(tt), Human_Player()])
game.play()
```

Полный код примера содержится в файле `coins.py`. Это интерактивная программа, поэтому она ожидает получения ввода от пользователя. Если вы запустите этот код, то будете играть против компьютера.

Ваша цель состоит в том, чтобы заставить компьютер взять последнюю монету, что принесет вам выигрыш в игре. После запуска кода и выполнения нескольких ходов в окне вашего терминала отобразится примерно следующий вывод (рис. 9.1).

```
d:2, a:0, m:1
d:3, a:0, m:1
d:4, a:0, m:1
d:5, a:0, m:1
d:6, a:0, m:1
d:7, a:0, m:1
d:8, a:0, m:1
d:9, a:0, m:1
d:10, a:100, m:4
1 10 4
25 coins left in the pile

Move #1: player 1 plays 4 :
21 coins left in the pile

Player 2 what do you play ? 1

Move #2: player 2 plays 1 :
20 coins left in the pile

Move #3: player 1 plays 4 :
16 coins left in the pile
```

Рис. 9.1

Завершающий фрагмент игры будет выглядеть примерно так (рис. 9.2).

```
Move #5: player 1 plays 2 :
11 coins left in the pile

Player 2 what do you play ? 4

Move #6: player 2 plays 4 :
7 coins left in the pile

Move #7: player 1 plays 1 :
6 coins left in the pile

Player 2 what do you play ? 2

Move #8: player 2 plays 2 :
4 coins left in the pile

Move #9: player 1 plays 3 :
1 coins left in the pile

Player 2 what do you play ? 1

Move #10: player 2 plays 1 :
0 coins left in the pile
```

Рис. 9.2

Как видите, игру выиграл компьютер, поскольку последняя монета была взята пользователем.

Создание робота для игры Tic-Tac-Toe ("Крестики-нолики")

Вероятно, одной из наиболее популярных игр является игра Tic-Tac-Toe ("Крестики-нолики"). Ниже продемонстрировано, как можно создать игру, в которой компьютер играет против пользователя. Это будет незначительно видоизмененный вариант программы для игры Tic-Tac-Toe, представленной в библиотеке `easyAI`.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from easyAI import TwoPlayersGame, AI_Player, Negamax
from easyAI.Player import Human_Player
```

Определим класс, содержащий все методы, необходимые для игры. Начнем с определения игроков и того, кто начинает игру.

```
class GameController(TwoPlayersGame):
    def __init__(self, players):
        # Определение игроков
        self.players = players

        # Определение того, кто начинает игру
        self.nplayer = 1
```

Мы будем использовать доску размером 3×3 с нумерацией вдоль строк от одного до девяти.

```
        # Определение доски
        self.board = [0] * 9
```

Определим метод, устанавливающий все возможные ходы.

```
    # Определение возможных ходов
    def possible_moves(self):
        return [a + 1 for a, b in enumerate(self.board) if b == 0]
```

Определим метод, обновляющий состояние доски после выполнения хода.

```
    # Выполнение хода
    def make_move(self, move):
        self.board[int(move) - 1] = self.nplayer
```

Определим метод, проверяющий, не проиграл ли кто-либо игру. Мы будем проверять, заполнил ли кто-либо три клетки подряд по горизонтали, вертикали или диагонали.

```
    # Заполнил ли противник три клетки подряд?
    def loss_condition(self):
        possible_combinations = [[1,2,3], [4,5,6], [7,8,9],
            [1,4,7], [2,5,8], [3,6,9], [1,5,9], [3,5,7]]
        return any([all([(self.board[i-1] == self.nopponent)
                for i in combination]) for combination in
                possible_combinations])
```

Проверим, не закончилась ли игра, используя метод `loss_condition`.

```
# Проверка того, закончилась ли игра
def is_over(self):
    return (self.possible_moves() == []) or
           self.loss_condition()
```

Определим метод, отображающий ход игры.

```
# Отображение текущей позиции
def show(self):
    print('\n'+'\n'.join([' '.join([['. ', 'O',
          'X'][self.board[3*j + i]]
          for i in range(3)]) for j in range(3)]))
```

Вычислим оценку, используя метод `loss_condition`.

```
# Вычисление оценки
def scoring(self):
    return -100 if self.loss_condition() else 0
```

Определим основную функцию, начав с определения алгоритма. В качестве алгоритма ИИ в этой игре мы будем использовать алгоритм NegaMax. Мы можем заранее указать количество шагов, которые должен продумывать алгоритм. В данном случае мы выберем его равным 7.

```
if __name__ == "__main__":
    # Определение алгоритма
    algorithm = Negamax(7)
```

Начнем игру.

```
    # Запуск игры
    GameController([Human_Player(), AI_Player(algorithm)]).play()
```

Полный код примера содержится в файле `tic_tac_toe.py`. Это интерактивная игра, в которой вы играете против компьютера. После запуска этого кода и выполнения нескольких ходов в окне терминала отобразится примерно следующий вывод (рис. 9.3).

Завершающий фрагмент игры будет выглядеть примерно так (рис. 9.4).

Как видим, данная игра закончилась вничью.

```
. . .
. . .
. . .

Player 1 what do you play ? 5

Move #1: player 1 plays 5 :

. . .
. O .
. . .

Move #2: player 2 plays 1 :

X . .
. O .
. . .

Player 1 what do you play ? 9

Move #3: player 1 plays 9 :

X . .
. O .
. . O
```

Рис. 9.3

```
X O X
. O .
. X O

Player 1 what do you play ? 4

Move #7: player 1 plays 4 :

X O X
O O .
. X O

Move #8: player 2 plays 6 :

X O X
O O X
. X O

Player 1 what do you play ? 7

Move #9: player 1 plays 7 :

X O X
O O X
O X O
```

Рис. 9.4

Создание двух роботов, играющих между собой в игру Connect Four ("Четыре в ряд")

Connect Four ("Четыре в ряд") — это популярная игра для двух участников, продаваемая под торговой маркой Milton Bradley. Она также известна под названиями "Four in a Row" и "Four Up". В этой игре игроки поочередно вставляют фишки в ячейки вертикальной доски, включающей шесть рядов и семь столбцов. Цель игры заключается в том, чтобы расположить раньше противника четыре фишки своего цвета подряд по вертикали, горизонтали или диагонали. Ниже описан видоизмененный вариант игры, приведенной в библиотеке easyAI. В нашем варианте не пользователь будет играть с компьютером, а два робота будут играть между собой. Чтобы увидеть, кто из них выиграет, мы будем использовать для них разные алгоритмы.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from easyAI import TwoPlayersGame, Human_Player, AI_Player, \
                  Negamax, SSS
```

Определим класс, содержащий все методы, необходимые для игры.

```
class GameController(TwoPlayersGame):
    def __init__(self, players, board = None):
        # Определение игроков
        self.players = players
```

Определим доску с шестью рядами и семью столбцами.

```
        # Определение конфигурации доски
        self.board = board if (board != None) else (
            np.array([[0 for i in range(7)] for j in range(6)]))
```

Определим того, кто начинает игру. В данном случае игру будет начинать первый игрок.

```
        # Определение того, кто начинает игру
        self.nplayer = 1
```

Определим позиции.

```
        # Определение позиций
        self.pos_dir = np.array([[[i, 0], [0, 1]] for i in
                                                    range(6)] +
                    [[[0, i], [1, 0]] for i in range(7)] +
                    [[[i, 0], [1, 1]] for i in range(1, 3)] +
```

```
                [[[0, i], [1, 1]] for i in range(4)] +
                [[[i, 6], [1, -1]] for i in range(1, 3)] +
                [[[0, i], [1, -1]] for i in range(3, 7)])
```

Определим метод, устанавливающий все возможные ходы.

```
# Определение возможных ходов
def possible_moves(self):
    return [i for i in range(7) if (self.board[:, i].min() == 0)]
```

Определим метод, устанавливающий ход, который следует сделать.

```
# Определение хода, который следует сделать
def make_move(self, column):
    line = np.argmin(self.board[:, column] != 0)
    self.board[line, column] = self.nplayer
```

Определим метод, отображающий текущее состояние.

```
# Отображение текущего состояния
def show(self):
    print('\n' + '\n'.join(
            ['0 1 2 3 4 5 6', 13 * '-'] +
            [' '.join([['.', 'O', 'X'][self.board[5 - j][i]]
            for i in range(7)]) for j in range(6)]))
```

Определим метод, вычисляющий условие проигрыша. Как только один из игроков выстраивает четыре фишки подряд в одну линию, этот игрок выигрывает игру.

```
# Определение условий проигрыша
def loss_condition(self):
    for pos, direction in self.pos_dir:
        streak = 0
        while (0 <= pos[0] <= 5) and (0 <= pos[1] <= 6):
            if self.board[pos[0], pos[1]] == self.nopponent:
                streak += 1
                if streak == 4:
                    return True
            else:
                streak = 0

            pos = pos + direction

    return False
```

Проверим, закончилась ли игра, используя метод `loss_condition`.

```
    # Проверка того, закончилась ли игра
    def is_over(self):
        return (self.board.min() > 0) or self.loss_condition()
```

Вычислим оценку.

```
    # Вычисление оценки
    def scoring(self):
        return -100 if self.loss_condition() else 0
```

Определим основную функцию, начав с определения алгоритмов. Мы позволим двум алгоритмам играть друг против друга. Для первого игрока-компьютера мы используем алгоритм NegaMax, а для второго — алгоритм SSS*. Алгоритм SSS* в основном представляет собой поисковый алгоритм, который выполняет поиск в пространстве состояний, обходя узлы дерева и останавливаясь на первом же наилучшем варианте. В качестве входного аргумента оба метода получают количество наперед продумываемых ходов. В данном случае мы используем значение 5 для обоих алгоритмов.

```
if __name__ == '__main__':
    # Определение используемых алгоритмов
    algo_neg = Negamax(5)
    algo_sss = SSS(5)
```

Начнем игру.

```
    # Начало игры
    game = GameController([AI_Player(algo_neg),
                           AI_Player(algo_sss)])
    game.play()
```

Выведем результат.

```
    # Вывод результата
    if game.loss_condition():
        print('\nPlayer', game.nopponent, 'wins. ')
    else:
        print("\nIt's a draw.")
```

Полный код примера содержится в файле `connect_four.py`. Это не интерактивная игра. Мы всего лишь "натравливаем" один алгоритм на другой. Алгоритм NegaMax — это первый игрок, алгоритм SSS* — второй.

Запустив этот код, вы увидите в начале выполнения программы следующий вывод в окне своего терминала (рис. 9.5).

Фрагмент вывода, соответствующий окончанию игры, будет выглядеть так (рис. 9.6).

```
0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .

Move #1: player 1 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
O . . . . . .

Move #2: player 2 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
```

Рис. 9.5

```
O O O X O O .

Move #35: player 1 plays 6 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X .
O O O X O O O

Move #36: player 2 plays 6 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X X
O O O X O O O

Player 2 wins.
```

Рис. 9.6

Как видите, игру выиграл второй игрок.

Создание двух роботов, играющих между собой в игру Hexapawn ("Шесть пешек")

Игра Hexapawn ("Шесть пешек") — это игра для двоих, которая происходит на шахматной доске размером N×M. Первоначально пешки располагаются на противоположных сторонах доски, и задачей каждого игрока является продвижение пешек с целью достигнуть другой стороны. При этом действуют стандартные правила, как в обычных шахматах. Это видоизмененный вариант программы, приведенной в библиотеке easyAI. Мы создадим двух роботов и столкнем алгоритм с самим собой для того, чтобы посмотреть, что при этом произойдет.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from easyAI import TwoPlayersGame, AI_Player, \ Human_Player, Negamax
```

Определим класс, содержащий все методы, необходимые для управления игрой. Начнем с определения количества пешек с каждой стороны и длину доски. Создадим список кортежей, содержащих позиции.

```
class GameController(TwoPlayersGame):
    def __init__(self, players, size = (4, 4)):
        self.size = size
        num_pawns, len_board = size
        p = [[(i, j) for j in range(len_board)]
             for i in [0, num_pawns - 1]]
```

Определим направления, цели и пешки для каждого игрока.

```
        for i, d, goal, pawns in [(0, 1, num_pawns - 1,
                                   p[0]), (1, -1, 0, p[1])]:
            players[i].direction = d
            players[i].goal_line = goal
            players[i].pawns = pawns
```

Определим игроков и укажем, кто из них начинает игру.

```
        # Определение игроков
        self.players = players

        # Определение того, кто начинает игру
        self.nplayer = 1
```

Определим буквы, которые будут использоваться для нумерации позиций (например, B6 или C7) на шахматной доске.

```
# Определение рабочего алфавита
self.alphabets = 'ABCDEFGHIJ'
```

Определим лямбда-функцию, преобразующую строки в кортежи.

```
# Преобразование B4 в (1, 3)
self.to_tuple = lambda s: (self.alphabets.index(s[0]),
        int(s[1:]) - 1)
```

Определим лямбда-функцию, преобразующую кортежи в строки.

```
# Преобразование (1, 3) в B4
self.to_string = lambda move: ' '.join([self.alphabets[
        move[i][0]] + str(move[i][1] + 1)
        for i in (0, 1)])
```

Определим метод, вычисляющий возможные ходы.

```
# Определение возможных ходов
def possible_moves(self):
    moves = []
    opponent_pawns = self.opponent.pawns
    d = self.player.direction
```

Если в намеченной позиции отсутствует пешка противника, то переход в эту позицию является допустимым.

```
    for i, j in self.player.pawns:
        if (i + d, j) not in opponent_pawns:
            moves.append(((i, j), (i + d, j)))

        if (i + d, j + 1) in opponent_pawns:
            moves.append(((i, j), (i + d, j + 1)))

        if (i + d, j - 1) in opponent_pawns:
            moves.append(((i, j), (i + d, j - 1)))

    return list(map(self.to_string, [(i, j) for i, j in moves]))
```

Определим ход, который следует сделать, и соответствующим образом обновим позиции пешек.

```
    # Определение хода
    def make_move(self, move):
        move = list(map(self.to_tuple, move.split(' ')))
        ind = self.player.pawns.index(move[0])
        self.player.pawns[ind] = move[1]

        if move[1] in self.opponent.pawns:
            self.opponent.pawns.remove(move[1])
```

Определим условия проигрыша. Если один из игроков выстраивает четыре пешки в одну линию, то это означает проигрыш его оппонента.

```
    # Определение условий проигрыша
    def loss_condition(self):
        return (any([i == self.opponent.goal_line
                for i, j in self.opponent.pawns])
                or (self.possible_moves() == []) )
```

Проверим, закончилась ли игра, с помощью метода `loss_condition`.

```
    # Проверка того, закончилась ли игра
    def is_over(self):
        return self.loss_condition()
```

Выведем текущее состояние.

```
    # Отображение текущего состояния
    def show(self):
        f = lambda x: '1' if x in self.players[0].pawns else (
                '2' if x in self.players[1].pawns else '.')

        print("\n".join([" ".join([f((i, j))
                for j in range(self.size[1])])
                for i in range(self.size[0])]))
```

Определим основную функцию, начав с определения оценочной лямбда-функции.

```
if __name__=='__main__':
    # Вычисление оценки
    scoring = lambda game: -100 if game.loss_condition() else 0
```

Определим используемый алгоритм. В данном случае мы будем использовать алгоритм NegaMax, который просчитывает 12 ходов наперед и использует оценочную лямбда-функцию для определения стратегии.

```
# Определение алгоритма
algorithm = Negamax(12, scoring)
```

Начнем игру.

```
# Запуск игры
game = GameController([AI_Player(algorithm),
                       AI_Player(algorithm)])
game.play()
print('\nPlayer', game.nopponent, 'wins after', game.nmove, 'turns')
```

Полный код примера содержится в файле `hexapawn.py`. Это не интерактивная игра. Мы стравливаем два алгоритма ИИ между собой. Запустив этот код, вы увидите в начале выполнения программы следующий вывод в окне своего терминала (рис. 9.7).

```
1 1 1 1
. . . .
. . . .
2 2 2 2

Move #1: player 1 plays A1 B1 :
. 1 1 1
1 . . .
. . . .
2 2 2 2

Move #2: player 2 plays D1 C1 :
. 1 1 1
1 . . .
2 . . .
. 2 2 2

Move #3: player 1 plays A2 B2 :
. . 1 1
1 1 . .
2 . . .
. 2 2 2

Move #4: player 2 plays D2 C2 :
  . 1 1
```

Рис. 9.7

Фрагмент вывода, соответствующий окончанию игры, будет выглядеть так (рис. 9.8).

```
Move #4: player 2 plays D2 C2 :
. . 1 1
1 1 . .
2 2 . .
. . 2 2

Move #5: player 1 plays B1 C2 :
. . 1 1
. 1 . .
2 1 . .
. . 2 2

Move #6: player 2 plays C1 B1 :
. . 1 1
2 1 . .
. 1 . .
. . 2 2

Move #7: player 1 plays C2 D2 :
. . 1 1
2 1 . .
. . . .
. 1 2 2

Player 1 wins after 8 turns
```

Рис. 9.8

Как видите, игру выиграл второй игрок.

Резюме

В этой главе мы обсудили создание игр с помощью искусственного интеллекта и использование поисковых алгоритмов для выработки эффективных стратегий игры, приводящих к выигрышу. Также был рассмотрен комбинаторный поиск и показано, как с его помощью можно ускорить процесс поиска. Вы узнали об алгоритмах MiniMax и альфа-бета-отсечении. Вы также познакомились с использованием алгоритма NegaMax на практике. Наконец, мы применили изученные алгоритмы при создании роботов для игр Last Coin Standing и Tic-Tac-Toe.

Кроме того, вы узнали о том, как создать двух роботов, играющих между собой в игры Connect Four и Hexapawn. Следующая глава посвящена обработке естественного языка и ее применениям для анализа текста посредством его моделирования и классификации.

10
Обработка естественного языка

В этой главе вы познакомитесь с обработкой естественного языка. Мы обсудим такие понятия, как токенизация, стемминг и лемматизация, которые используются при обработке текста. Затем мы перейдем к обсуждению модели Bag of Words и применим ее для классификации текста. Будет показано, как задействовать машинное обучение для сентимент-анализа предложений. После этого мы обсудим тематическое моделирование и реализуем систему идентификации тем в документах.

К концу главы вы освоите следующие темы:

- установка необходимых пакетов;
- токенизация текстовых данных;
- преобразование слов в их базовые формы с помощью стемминга;
- преобразование слов в их базовые формы с помощью лемматизации;
- разбиение текста на информационные блоки;
- извлечение терм-документной матрицы с помощью модели Bag of Words;
- создание прогнозатора категорий;
- построение анализатора грамматических родов;
- создание сентимент-анализатора;
- тематическое моделирование с использованием латентного размещения Дирихле.

Введение и установка пакетов

Обработка естественного языка (Natural Language Processing — NLP) становится важной частью современных систем. Она интенсивно применяется в поисковых системах, речевых интерфейсах, процессорах документов

и т.п. Машины отлично справляются со структурированными данными. Но если речь идет об обработке текста в свободной форме, то машинам приходится нелегко. Целью NLP является разработка алгоритмов, которые позволяли бы компьютерам распознавать свободный текст и понимать живую речь. Уже одно только количество возможных вариаций является одной из наибольших трудностей, связанных с обработкой естественного языка. Для понимания смысла конкретных предложений большое значение имеет контекст. Люди замечательным образом справляются с этим, поскольку учатся этому на протяжении многих лет. Мы немедленно применяем наши знания для понимания контекста и знаем, о чем именно говорит другой человек.

Для преодоления этой проблемы исследователи в области NLP начали разрабатывать различные приложения, используя подходы на основе машинного обучения. Чтобы разрабатывать подобные приложения, мы должны собирать огромные массивы текста, а затем обучать алгоритм для выполнения различных задач, таких как категоризация текста, сентимент-анализ или тематическое моделирование. При этом алгоритмы учатся обнаруживать повторяющиеся шаблоны во входном тексте и извлекать содержащийся в нем смысл.

В этой главе обсуждаются базовые понятия, которые используются в ходе анализа текста и создания приложений NLP. Это позволит вам понять, на чем основывается извлечение смысловой информации из предоставленных текстовых данных. Для построения соответствующих приложений мы будем использовать пакет Python **Natural Language Toolkit (NLTK)**. Обязательно установите этот пакет, прежде чем читать дальше. Введите в окне терминала следующую команду:

```
$ pip3 install nltk
```

Более подробную информацию о пакете NLTK можно найти на сайте `http://www.nltk.org`.

Чтобы получить доступ ко всем наборам данных, предоставляемых в пакете NLTK, мы должны загрузить их. Откройте оболочку Python, введя в окне терминала следующую команду:

```
$ python3
```

Теперь мы находимся в оболочке. Введите следующие команды для загрузки данных.

```
>>> import nltk
>>> nltk.download()
```

В этой главе мы также будем использовать пакет `gensim`. Это надежная библиотека средств семантического моделирования, которая может быть полезной для многих приложений. Установите этот пакет, выполнив в окне терминала следующую команду:

```
$ pip3 install gensim
```

Для правильной работы пакета `gensim` вам может понадобиться еще один пакет: `pattern`. Его можно установить с помощью следующей команды:

```
$ pip3 install pattern
```

Более подробную информацию о пакете `gensim` можно найти по адресу `https://radimrehurek.com/gensim`. Теперь, когда у вас установлены пакеты NLTK и `gensim`, мы можем перейти к непосредственному обсуждению материала.

Токенизация текстовых данных

Работая с текстом, мы должны разбивать его на более мелкие фрагменты для проведения анализа. И здесь мы сталкиваемся с токенизацией. *Токенизация* — это процесс разбиения входного текста на меньшие элементы, такие как слова или предложения. Эти элементы называются *токенами* (лексемами). В зависимости от того, что мы хотим сделать, мы можем определить собственные методы для разбиения текста на множество токенов. Покажем, как токенизировать входной текст с помощью пакета NLTK.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from nltk.tokenize import sent_tokenize, word_tokenize, WordPunctTokenizer
```

Определим входной текст, который будем использовать для токенизации.

```
# Определение входного текста
input_text = "Do you know how tokenization works? It's actually
quite interesting! Let's analyze a couple of sentences and
figure it out."
```

Выполним разбивку входного текста с помощью токенизатора предложений.

```
# Токенизация предложений
print("\nSentence tokenizer:")
print(sent_tokenize(input_text))
```

Выполним разбивку входного текста с помощью токенизатора слов.

```
# Токенизатор слов
print("\nWord tokenizer:")
print(word_tokenize(input_text))
```

Выполним разбивку входного текста с помощью токенизатора слов и токенизатора пунктуации.

```
# Токенизатор пунктуации
print("\nWord punct tokenizer:")
print(WordPunctTokenizer().tokenize(input_text))
```

Полный код этого примера содержится в файле `tokenizer.py`. После выполнения этого кода в окне терминала отобразится следующий вывод (рис. 10.1).

```
Sentence tokenizer:
['Do you know how tokenization works?', "It's actually quite interesting!", "Let's analyze a couple of se
ntences and figure it out."]

Word tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'s", 'actually', 'quite', 'interesting'
, '!', 'Let', "'s", 'analyze', 'a', 'couple', 'of', 'sentences', 'and', 'figure', 'it', 'out', '.']

Word punct tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'", 's', 'actually', 'quite', 'interest
ing', '!', 'Let', "'", 's', 'analyze', 'a', 'couple', 'of', 'sentences', 'and', 'figure', 'it', 'out', '.
']
```

Рис. 10.1

Как видите, токенизатор предложений разбивает входной текст на предложения. Два токенизатора слов ведут себя по-разному в отношении знаков пунктуации. Например, слово "It's" разбивается токенизатором пунктуации иначе, нежели обычным токенизатором.

Преобразование слов в их базовые формы с помощью стемминга

Работа с текстом связана с множеством нюансов. Мы должны работать с различными формами одного и того же слова и хотим добиться того, чтобы компьютер понимал, что эти различные слова имеют одну и ту же корневую форму. Например, слово *sing* может встречаться во многих формах, таких как *sang*, *singer*, *singing*, *singer* и др. Все только что перечисленные слова имеют общий корень. Люди легко распознают такие базовые формы и определяют контекст.

В процессе анализа текста полезно извлекать такие корневые формы. Это позволяет получать полезные статистики, помогающие анализировать входной текст. Одним из способов обеспечения этого является *стемминг*. Целью стемминга является приведение слов в их различных формах к общему корню. В основном это эвристический процесс, заключающийся в отсечении окончаний слов для выделения их корневых форм. Покажем, как это можно сделать с помощью NLTK.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer
```

Определим входные слова.

```
input_words = ['writing', 'calves', 'be', 'branded', 'horse',
               'randomize', 'possibly', 'provision', 'hospital',
               'kept', 'scratchy', 'code']
```

Создадим объекты для стеммеров Портера (`Porter`), Ланкастера (`Lancaster`) и Сноуболла (`Snowball`).

```
# Создание объектов различных стеммеров
porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer('english')
```

Создадим список имен стеммеров для их отображения в виде таблицы и соответствующим образом отформатируем выходной текст.

```
# Создание списка имен стеммеров для отображения
stemmer_names = ['PORTER', 'LANCASTER', 'SNOWBALL']
formatted_text = '{:>16}' * (len(stemmer_names) + 1)
print('\n', formatted_text.format('INPUT WORD', *stemmer_names),
      '\n', '='*68)
```

Выполним итеративный стемминг слов, используя три различных стеммера.

```
# Стемминг слов и отображение результатов
for word in input_words:
    output = [word, porter.stem(word),
              lancaster.stem(word), snowball.stem(word)]
    print(formatted_text.format(*output))
```

Полный код этого примера содержится в файле `stemmer.py`. Выполнив этот код, вы увидите в окне терминала следующий вывод (рис. 10.2).

INPUT WORD	PORTER	LANCASTER	SNOWBALL
writing	write	writ	write
calves	calv	calv	calv
be	be	be	be
branded	brand	brand	brand
horse	hors	hors	hors
randomize	random	random	random
possibly	possibl	poss	possibl
provision	provis	provid	provis
hospital	hospit	hospit	hospit
kept	kept	kept	kept
scratchy	scratchi	scratchy	scratchi
code	code	cod	code

Рис. 10.2

Следует сделать несколько замечаний относительно трех алгоритмов стемминга, использованных в этом коде. В общем, все они пытаются достичь одной и той же цели. Различия между ними заключаются в степени строгости, используемой при получении базовой формы.

Наименее строгим из них является стеммер Портера, а наиболее строгим — стеммер Ланскастера. Если вы внимательно присмотритесь к выводу, то заметите различия. Стеммеры ведут себя по-разному в отношении слов наподобие `possibly` или `provision`. Результаты, полученные с помощью стеммера Ланкастера, немного сбивают с толку, поскольку он слишком урезает слова. В то же время этот алгоритм демонстрирует высокую скорость. В большинстве случае неплохо использовать стеммер Сноуболла, который обеспечивает разумный компромисс между быстродействием и строгостью.

Преобразование слов в их корневые формы с помощью лемматизации

Лемматизация — еще один способ редукции слов к их корневым формам. Как вы могли заметить в предыдущем разделе, корневые формы, полученные с помощью стиммеров, не имели смысла. Например, в качестве корня слова `calves` все три стеммера указали форму `calv`, в действительности не являющуюся словом. Лемматизация предпринимает более структурированный подход для разрешения этой проблемы.

Процеес лемматизации использует словарь и морфологический анализ слов. В нем корневые формы получают путем удаления окончаний, таких как *ing* или *ed*, из форм с окончаниями. Такую базовую форму любого слова называют *леммой*. Если вы лемматизируете слово `calves`, то получите `calf` в качестве результата. Следует отметить, что конечный результат зависит от того, является слово глаголом или существительным. Рассмотрим пример лемматизации с помощью пакета NLTK.

Создайте новый файл Python и импортируйте следующие пакеты:

```
from nltk.stem import WordNetLemmatizer
```

Определим входные слова. Мы будем использовать тот же набор слов, что и в предыдущем разделе, чтобы можно было сравнить результаты.

```
input_words = ['writing', 'calves', 'be', 'branded', 'horse',
               'randomize', 'possibly', 'provision', 'hospital',
               'kept', 'scratchy', 'code']
```

Создадим объект `lemmatizer`.

```
# Создание объекта лемматизатора
lemmatizer = WordNetLemmatizer()
```

Создадим список имен лемматизаторов для их отображения в виде таблицы и соответствующим образом отформатируем выходной текст.

```
# Создание списка имен лемматизаторов для их отображения
lemmatizer_names = ['NOUN LEMMATIZER', 'VERB LEMMATIZER']
formatted_text = '{:>24}' * (len(lemmatizer_names) + 1)
print('\n', formatted_text.format('INPUT WORD',
      *lemmatizer_names), '\n', '='*75)
```

Выполним итеративную лемматизацию слов, используя лемматизаторы `Noun` (существительное) и `Verb` (глагол).

```
# Лемматизация слов и отображение результатов
for word in input_words:
    output = [word, lemmatizer.lemmatize(word, pos='n'),
              lemmatizer.lemmatize(word, pos='v')]
    print(formatted_text.format(*output))
```

Полный код этого примера содержится в файле `lemmatizer.py`. Выполнив этот код, вы увидите в окне своего терминала следующий вывод (рис. 10.3).

INPUT WORD	NOUN LEMMATIZER	VERB LEMMATIZER
writing	writing	write
calves	calf	calve
be	be	be
branded	branded	brand
horse	horse	horse
randomize	randomize	randomize
possibly	possibly	possibly
provision	provision	provision
hospital	hospital	hospital
kept	kept	keep
scratchy	scratchy	scratchy
code	code	code

Рис. 10.3

Нетрудно заметить, что в отношении таких слов, как `writing` или `calves`, лемматизатор существительных работает иначе, чем лемматизатор глаголов. Если вы сравните эти результаты с теми, которые были получены с помощью стеммеров, то увидите, что здесь также имеются различия. Все результаты, полученные с помощью лемматизаторов, имеют смысл, тогда как результаты, полученные с помощью стеммеров, могут как иметь смысл, так и не иметь.

Разбиение текстовых данных на информационные блоки

Обычно текстовые данные приходится разбивать на отдельные информационные блоки (чанки) для последующего анализа. Этот процесс называется *чанкингом*, и он часто используется при анализе текста. Условия разбиения текста на блоки могут меняться в зависимости от конкретной задачи. Чанкинг не следует смешивать с токенизацией, которая также связана с разбиением текста на отдельные части. В процессе чанкинга мы не придерживаемся никаких ограничений, а результирующие блоки обязаны иметь смысл.

При работе с большими документами важно разбивать текст на блоки для извлечения смысловой информации. В этом разделе будет показано, как разбить входной текст на ряд блоков.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from nltk. Corpus import brown
```

Определим функцию, разбивающую входной текст на блоки. Первым параметром является текст, вторым — количество слов в каждом блоке.

```
# Разбиение входного текста на блоки,
# причем каждый блок содержит N слов
def chunkier(input_data, N):
    input_words = input_data.split(' ')
    output = []
```

Выполним цикл по словам и разобьем их на блоки, используя входной параметр. Данная функция возвращает список.

```
    cur_chunk = []
    count = 0
    for word in input_words:
        cur_chunk.append(word)
        count += 1
        if count == N:
            output.append(' '.join(cur_chunk))
            count, cur_chunk = 0, []

    output.append(' '.join(cur_chunk))

    return output
```

Определим основную функцию и прочитаем входные данные, используя коллекцию текстов Brown. В данном случае мы читаем 12 тысяч слов, но вы можете задать для этого параметра любое другое желаемое значение.

```
if __name__=='__main__':
    # Чтение первых 12000 слов из коллекции Brown
    input_data = ' '.join(brown.words()[:12000])
```

Определим количество слов в каждом блоке.

```
    # Определение количества слов в каждом блоке
    chunk_size = 700
```

Разобьем текст на блоки и отобразим результат.

```
    chunks = chunker(input_data, chunk_size)
    print('\nNumber of text chunks =', len(chunks), '\n')
    for i, chunk in enumerate(chunks):
        print('Chunk', i+1, '==>', chunk[:50])
```

Полный код этого примера содержится в файле `text_chunker.py`. Выполнив этот код, вы увидите в окне терминала следующий вывод (рис. 10.4).

```
Number of text chunks = 18

Chunk 1 ==> The Fulton County Grand Jury said Friday an invest
Chunk 2 ==> '' . ( 2 ) Fulton legislators `` work with city of
Chunk 3 ==> . Construction bonds Meanwhile , it was learned th
Chunk 4 ==> , anonymous midnight phone calls and veiled threat
Chunk 5 ==> Harris , Bexar , Tarrant and El Paso would be $451
Chunk 6 ==> set it for public hearing on Feb. 22 . The proposa
Chunk 7 ==> College . He has served as a border patrolman and
Chunk 8 ==> of his staff were doing on the address involved co
Chunk 9 ==> plan alone would boost the base to $5,000 a year a
Chunk 10 ==> nursing homes In the area of `` community health s
Chunk 11 ==> of its Angola policy prove harsh , there has been
Chunk 12 ==> system which will prevent Laos from being used as
Chunk 13 ==> reform in recipient nations . In Laos , the admini
Chunk 14 ==> . He is not interested in being named a full-time
Chunk 15 ==> said , `` to obtain the views of the general publi
Chunk 16 ==> '' . Mr. Reama , far from really being retired , i
Chunk 17 ==> making enforcement of minor offenses more effectiv
Chunk 18 ==> to tell the people where he stands on the tax issu
```

Рис. 10.4

На этом снимке экрана представлены первые 50 символов каждого блока.

Извлечение частотности слов с помощью модели Bag of Words

Одной из основных задач текстового анализа является преобразование текста в числовую форму, к которой можно было бы применить машинное обучение. Рассмотрим случай текстовых документов, содержащих миллионы слов. Анализ таких документов предполагает извлечение текста и преобразование его в числовое представление.

Чтобы алгоритмы машинного обучения могли анализировать данные и извлекать содержательную информацию, они должны получать эти данные в числовом виде. И здесь на помощь приходит модель Bag of Words (мешок слов). Эта модель извлекает словарь, состоящий из всех слов, которые встречаются в документе, и строит модель с помощью терм-документной матрицы. Это позволяет нам представить любой документ в виде "мешка слов". При этом мы поддерживаем лишь счетчик слов, игнорируя их порядок следования и грамматические детали.

Поговорим о том, что такое *терм-документная матрица* (document term matrix). Это таблица, которая содержит значения счетчиков различных слов, встречающихся в документе. Поэтому текстовый документ можно представить в виде взвешенной комбинации различных слов. Мы можем устанавливать пороговые значения и выбирать наиболее значимые слова. В некотором смысле мы строим гистограмму всех слов в документе, совокупность которых будет служить вектором признаков. Этот вектор признаков используется для классификации текста.

Рассмотрим следующие предложения.

- *Предложение 1*: The children are playing in the hall.
- *Предложение 2*: The hall has a lot of space.
- *Предложение 3*: Lots of children like playing in an open space.

Среди всех слов, входящих в состав всех трех предложений, часть являются уникальными.

- the
- children
- are
- playing
- in
- hall
- has
- a
- lot
- of
- space
- like
- an
- open

Всего здесь имеется 14 различных слов. Построим гистограмму для каждого предложения, используя их счетчики слов. Каждый вектор признаков будет 14-мерным, поскольку общее количество слов составляет 14.

- *Предложение 1*: [2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
- *Предложение 2*: [1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0]
- *Предложение 3*: [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1]

Теперь, когда векторы признаков извлечены, мы можем использовать алгоритмы машинного обучения для анализа данных.

Покажем, как создать модель Bag of Words в NLTK. Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import brown
from text_chunker import chunker
```

Прочитаем входные данные из коллекции Brown. Мы прочитаем 5400 слов, однако вы можете задать любое другое их количество.

```
# Чтение данных из коллекции Brown
input_data = ' '.join(brown.words()[:5400])
```

Определим количество слов в каждом блоке.

```
# Количество слов в каждом блоке
chunk_size = 800
```

Разобьем входной текст на блоки.

```
text_chunks = chunker(input_data, chunk_size)
```

Преобразуем блоки в элементы словаря.

```
# Преобразование в элементы словаря
chunks = []
for count, chunk in enumerate(text_chunks):
    d = {'index': count, 'text': chunk}
    chunks.append(d)
```

Извлечем терм-документную матрицу, из которой мы получаем значения счетчиков каждого слова. Для этого используем метод `CountVectorizer`, который имеет два входных параметра. Первый параметр — минимальная, а второй — максимальная частота документа. Здесь термин "частота" относится к количеству вхождений слова в тексте.

```
# Извлечение терм-документной матрицы
count_vectorizer = CountVectorizer(min_df=7, max_df=20)
document_term_matrix = count_vectorizer.fit_transform([chunk['text']
        for chunk in chunks])
```

Извлечем словарь и отобразим его. Здесь термин "словарь" относится к списку отдельных слов, извлеченных на предыдущем шаге.

```
# Извлечение и отображение словаря
vocabulary = np.array(count_vectorizer.get_feature_names())
print("\nVocabulary:\n", vocabulary)
```

Сгенерируем отображаемые имена.

```
# Генерация имен блоков
chunk_names = []
for i in range(len(text_chunks)):
    chunk_names.append('Chunk-' + str(i+1))
```

Выведем терм-документную матрицу.

```
# Вывод терм-документной матрицы
print("\nDocument term matrix:")
formatted_text = '{:>12}' * (len(chunk_names) + 1)
print('\n', formatted_text.format('Word', *chunk_names), '\n')
for word, item in zip(vocabulary, document_term_matrix.T):
    # 'item' - это структура данных 'csr_matrix'
    output = [word] + [str(freq) for freq in item.data]
    print(formatted_text.format(*output))
```

Полный код этого примера содержится в файле `bag_of_words.py`. Выполнив этот код, вы увидите в окне терминала следующий вывод (рис. 10.5).

```
Document term matrix:

        Word      Chunk-1      Chunk-2      Chunk-3      Chunk-4      Chunk-5      Chunk-6      Chunk-7

         and           23            9            9           11            9           17           10
         are            2            2            1            1            2            2            1
          be            6            8            7            7            6            2            1
          by            3            4            4            5           14            3            6
      county            6            2            7            3            1            2            2
         for            7           13            4           10            7            6            4
          in           15           11           15           11           13           14           17
          is            2            7            3            4            5            5            2
          it            8            6            8            9            3            1            2
          of           31           20           20           30           29           35           26
          on            4            3            5           10            6            5            2
         one            1            3            1            2            2            1            1
        said           12            5            7            7            4            3            7
       state            3            7            2            6            3            4            1
        that           13            8            9            2            7            1            7
         the           71           51           43           51           43           52           49
          to           11           26           20           26           21           15           11
         two            2            1            1            1            1            2            2
         was            5            6            7            7            4            7            3
       which            7            4            5            4            3            1            1
        with            2            2            3            1            2            2            3
```

Рис. 10.5

Этот вывод представляет все слова, содержащиеся в терм-документной матрице, и значения соответствующих счетчиков для каждого блока.

Создание прогнозатора категорий

Прогнозаторы категорий используются для предсказания категории, к которой принадлежит заданный элемент текста. Их часто используют при классификации текста для категоризации текстовых документов. Поисковые механизмы часто используют этот инструмент для поиска результатов по их релевантности. Предположим, мы хотим спрогнозировать, относится ли данное предложение к спорту, политике или науке. Для этого мы создаем массив данных и тренируем алгоритм. Далее этот алгоритм может быть использован для вынесения суждений о неизвестных данных.

Для построения такого прогнозатора мы будем использовать статистику **tf-idf** (**Term Frequency — InverseDocument Frequency** — "частота слова — обратная частота документа"). Мы должны понимать важность каждого слова в наборе документов. Статистика tf-idf помогает оценить, насколько важным является данное слово в документе из набора.

Рассмотрим первую часть этой статистики, **tf** (*term frequency* — частота слова). Это мера частотности каждого слова, встречающегося в документе. Поскольку общее количество слов в разных документах разное, абсолютные числа в гистограмме будут меняться. Чтобы уравнять условия, гистограмму необходимо нормализовать. Поэтому мы делим значение счетчика каждого слова на общее количество слов в данном документе для получения значения частотности.

Второй частью данной статистики является **idf** (*inverse document frequency* — обратная частота документа) — мера уникальности слова в данном документе коллекции. Вычисляя частотность, мы полагаем, что все слова одинаково важны. Но сама по себе частотность слов не может служить надежной характеристикой, поскольку, например, такие общеупотребительные слова, как *like* и *the*, встречаются намного чаще других. Чтобы сбалансировать частотность таких общеупотребительных слов, мы должны уменьшить их вес и одновременно увеличить вес редко используемых слов. Благодаря этому мы получаем возможность идентифицировать слова, являющиеся уникальными для всех документов коллекции, что, в свою очередь, позволяет формулировать более точные векторы признаков, учитывающие характеристичность слов.

Чтобы вычислить эту статистику, нужно вычислить отношение количества документов, содержащих данное слово, к общему количеству документов в коллекции. По сути, это отношение характеризует долю документов, содержащих данное слово. Обратная частота документа рассчитывается далее как логарифм этого отношения, взятый со знаком "минус".

После этого мы объединяем частоту слов и обратную частоту документа, чтобы сформировать вектор признаков для категоризации документов. Рассмотрим, как создать прогнозатор категорий.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
```

Определим карту категорий, которые будут использоваться для тренировки. В данном случае у нас будет пять категорий. В этом словаре ключи ссылаются на имена в пакете `scikit-learn`.

```
# Определение карты категорий
category_map = {'talk.politics.misc': 'Politics',
        'rec.autos': 'Autos', 'rec.sport.hockey': 'Hockey',
        'sci.electronics': 'Electronics', 'sci.med': 'Medicine'}
```

Получим тренировочный набор, используя вызов `fetch_20newsgroups`.

```
# Получение тренировочного набора данных
training_data = fetch_20newsgroups(subset='train',
        categories=category_map.keys(), shuffle=True,
        random_state=5)
```

Извлечем значения счетчиков слов с помощью объекта `CountVectorizer`.

```
# Создание векторизатора и извлечение счетчиков слов
count_vectorizer = CountVectorizer()
train_tc = count_vectorizer.fit_transform(training_data.data)
print("\nDimensions of training data:", train_tc.shape)
```

Создадим преобразователь "Term Frequency — Inverse Document Frequency" (tf-idf) и обучим его, используя имеющиеся данные.

```
# Создание преобразователя tf-idf
tfidf = TfidfTransformer()
train_tfidf = tfidf.fit_transform(train_tc)
```

Определим выборку входных предложений, которые будут использованы для тестирования.

```
# Определение тестовых данных
input_data = [
    'You need to be careful with cars when you are driving on slippery
roads',
```

```
    'A lot of devices can be operated wirelessly',
    'Players need to be careful when they are close to goal posts',
    'Political debates help us understand the perspectives of both
sides'
]
```

Обучим мультиномиальный байесовский классификатор, используя тренировочные данные.

```
# Обучение мультиномиального байесовского классификатора
classifier = MultinomialNB().fit(train_tfidf, training_data.target)
```

Преобразуем входные данные, используя векторизатор счетчиков.

```
# Преобразование входных данных с помощью
# векторизатора счетчиков
input_tc = count_vectorizer.transform(input_data)
```

Преобразуем векторизованные данные с помощью преобразователя tf-idf, чтобы их можно было пропустить через модель выводов.

```
# Преобразование векторизованных данных с помощью
# преобразователя tf-idf
input_tfidf = tfidf.transform(input_tc)
```

Предскажем результат, используя вектор, полученный с помощью преобразователя `tf-idf`.

```
# Прогнозирование результирующих категорий
predictions = classifier.predict(input_tfidf)
```

Выведем результирующие категории для каждого из образцов во входных тестовых данных.

```
# Вывод результатов
for sent, category in zip(input_data, predictions):
    print('\nInput:', sent, '\nPredicted category:',
          category_map[training_data.target_names[category]])
```

Полный код этого примера содержится в файле `category_predictor.py`. Выполнив этот код, вы увидите в окне терминала следующие результаты (рис. 10.6).

Как нетрудно убедиться, категории предсказаны верно.

```
Dimensions of training data: (2844, 40321)

Input: You need to be careful with cars when you are driving on slippery roads
Predicted category: Autos

Input: A lot of devices can be operated wirelessly
Predicted category: Electronics

Input: Players need to be careful when they are close to goal posts
Predicted category: Hockey

Input: Political debates help us understand the perspectives of both sides
Predicted category: Politics
```

Рис. 10.6

Создание анализатора грамматических родов

Одной из задач, представляющих интерес в связи с обработкой текста, является идентификация грамматического рода. В данном случае мы создадим вектор признаков с помощью эвристики и используем его для обучения классификатора. В качестве эвристики выберем последние N букв заданного имени. Например, если имя заканчивается буквами *ia*, то, вероятнее всего, это женское имя, такое как *Amelia* или *Genelia*. С другой стороны, если имя заканчивается буквами *rk*, то оно, вероятно, является мужским именем, таким как *Mark* или *Clark*. Поскольку точное количество букв, которые при этом следует учитывать, нам неизвестно, мы проведем эксперименты, в ходе которых выясним, какое количество букв обеспечивает наилучшие ответы. Рассмотрим, как это делается.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import random
from nltk import NaiveBayesClassifier
from nltk.classify import accuracy as nltk_accuracy
from nltk.corpus import names
```

Определим функцию, извлекающую последние N букв из входного слова.

```
# Извлечение последних N букв из входного слова
# и возврат значения, выступающего в качестве "признака"
def extract_features(word, N=2):
    last_n_letters = word[-N:]
    return {'feature': last_n_letters.lower()}
```

Определим основную функцию и извлечем обучающие данные из пакета `scikit-learn`. Эти данные содержат маркированные мужские и женские имена.

```
if __name__=='__main__':
    # Создание обучающих данных с использованием
    # помеченных имен, доступных в NLTK
    male_list = [(name, 'male') for name in
            names.words('male.txt')]
    female_list = [(name, 'female') for name in
            names.words('female.txt')]
    data = (male_list + female_list)
```

Зададим затравочное значение для генератора случайных чисел и перемешаем данные.

```
    # Затравочное значение для генератора случайных чисел
    random.seed(5)

    # Перемешивание данных
    random.shuffle(data)
```

Создадим выборку имен, которые будут использоваться для тестирования.

```
    # Создание тестовых данных
    input_names = ['Alexander', 'Danielle', 'David', 'Cheryl']
```

Определим процентные доли данных, которые будут использоваться для тренировки и тестирования.

```
    # Определение количеств образцов, используемых
    # для тренировки и тестирования
    num_train = int(0.8 * len(data))
```

Для прогнозирования рода мы будем использовать в качестве вектора признаков последние N символов имени. Мы будем варьировать этот параметр, чтобы выяснить, как при этом меняются результаты. В данном случае мы исследуем интервал значений от 1 до 5.

```
    # Итерирование по различным длинам конечного
    # фрагмента для сравнения точности
    for i in range(1, 6):
        print('\nNumber of end letters:', i)
        features = [(extract_features(n, i), gender) for (n,
                    gender) in data]
```

Разделим данные на тренировочный и тестовый наборы.

```
        train_data, test_data = features[:num_train],
                                features[num_train:]
```

Создадим наивный байесовский классификатор, используя тренировочные данные.

```
classifier = NaiveBayesClassifier.train(train_data)
```

Вычислим точность классификатора, используя встроенный метод, доступный в библиотеке NLTK.

```
# Вычисление точности классификатора
accuracy = round(100 * nltk_accuracy(classifier, test_data), 2)
print('Accuracy = ' + str(accuracy) + '%')
```

Предскажем результат для каждого имени из входного тестового списка.

```
# Предсказание результатов для входных имен
# с использованием обученной модели классификатора
for name in input_names:
    print(name, '==>',
          classifier.classify(extract_features(name, i)))
```

Полный код этого примера содержится в файле `gender_identifier.py`. Выполнив этот код, вы получите следующий вывод в окне своего терминала (рис. 10.7).

```
Number of end letters: 1
Accuracy = 74.7%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> male

Number of end letters: 2
Accuracy = 78.79%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female

Number of end letters: 3
Accuracy = 77.22%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female
```

Рис. 10.7

На этом снимке экрана представлены точность и предсказанные результаты для тестовых данных. Посмотрим, как обстоят дела с большими значениями N (рис. 10.8).

```
Number of end letters: 4
Accuracy = 69.98%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female

Number of end letters: 5
Accuracy = 64.63%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female
```

Рис. 10.8

Нетрудно заметить, что точность достигает своего максимума при N = 2, а затем начинает снижаться.

Создание сентимент-анализатора

Сентимент-анализ — это процесс определения тональности заданного фрагмента текста. Его, например, можно использовать для того, чтобы определить, является ли отзыв о кинофильме положительным или отрицательным. Это одно из наиболее популярных применений обработки естественного языка. В зависимости от конкретной задачи мы можем расширять количество категорий. Обычно эту методику используют для того, чтобы определить отношение людей к конкретному продукту, торговой марке или теме. Ее часто применяют для анализа результатов маркетинговых компаний, опросов общественного мнения, присутствия в социальных сетях, составления обзоров продуктов на торговых сайтах т.п. Рассмотрим, как определить тональность рецензии на фильм.

Для создания данного анализатора мы используем наивный байесовский классификатор. Сначала нам предстоит извлечь все уникальные слова из текста. Для работы NLTK-классификатора требуется, чтобы данные были подготовлены в виде словаря. Как только текстовые данные будут разделены на тренировочный и тестовый наборы, мы обучим наивный байесовский

классификатор распределять обзоры на основании того, являются ли они положительными и отрицательными. Мы также выведем наиболее информативные слова, указывающие на положительные или отрицательные отзывы. Эта информация представляет для нас интерес, поскольку позволяет понять, какие именно слова используются для выражения той или иной реакции.

Создайте новый файл Python и импортируйте следующие пакеты.

```
from nltk.corpus import movie_reviews
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy as nltk_accuracy
```

Определим функцию, которая создает объект словаря на основании входных слов и возвращает его.

```
# Извлечение признаков из входного списка слов
def extract_features(words):
    return dict([(word, True) for word in words])
```

Определим основную функцию и загрузим помеченные отзывы о фильмах.

```
if __name__=='__main__':
    # Загрузка отзывов из коллекции
    fileids_pos = movie_reviews.fileids('pos')
    fileids_neg = movie_reviews.fileids('neg')
```

Извлечем признаки из отзывов о фильмах и снабдим их соответствующими метками.

```
    # Извлечение признаков из отзывов
    features_pos = [(extract_features(movie_reviews.words(
            fileids=[f])), 'Positive') for f in fileids_pos]
    features_neg = [(extract_features(movie_reviews.words(
            fileids=[f])), 'Negative') for f in fileids_neg]
```

Определим разбиение данных на тренировочный и тестовый наборы. В нашем случае 80% данных будут тренировочными, а 20% — тестовыми.

```
    # Определение относительных долей тренировочного
    # и тестового наборов (80% и 20%)
    threshold = 0.8
    num_pos = int(threshold * len(features_pos))
    num_neg = int(threshold * len(features_neg))
```

Разделим векторы признаков для тренировочного и тестового наборов.

```
    # Создание тренировочного и тестового наборов
    features_train = features_pos[:num_pos] +
```

```
                    features_neg[:num_neg]
features_test = features_pos[num_pos:] +
                    features_neg[num_neg:]
```

Выведем количество точек данных, используемых для тренировки и тестирования.

```
# Вывод количества используемых точек данных
print('\nNumber of training datapoints:',
      len(features_train))
print('Number of test datapoints:', len(features_test))
```

Обучим наивный байесовский классификатор, используя тренировочные данные, и вычислим точность, используя встроенный метод, доступный в NLTK.

```
# Обучение наивного байесовского классификатора
classifier = NaiveBayesClassifier.train(features_train)
print('\nAccuracy of the classifier:', nltk_accuracy(
      classifier, features_test))
```

Выведем первые N наиболее информативных слов.

```
N = 15
print('\nTop ' + str(N) + ' most informative words:')
for i, item in
        enumerate(classifier.most_informative_features()):
    print(str(i+1) + '. ' + item[0])
    if i == N - 1:
        break
```

Определим выборку предложений, используемых для тестирования.

```
# Тестирование входных отзывов о фильмах
input_reviews = [
    'The costumes in this movie were great',
    'I think the story was terrible and the characters were very
weak',
    'People say that the director of the movie is amazing',
    'This is such an idiotic movie. I will not recommend it to
anyone.'
]
```

Выполним итерации по выборочным данным и спрогнозируем результаты.

```
print("\nMovie review predictions:")
for review in input_reviews:
    print("\nReview:", review)
```

Вычислим вероятности для каждого класса.

```
        # Вычисление вероятностей
        probabilities =
classifier.prob_classify(extract_features(review.split()))
```

Выберем вероятность с наибольшим значением.

```
        # Выбор максимального значения
        predicted_sentiment = probabilities.max()
```

Выберем предсказанный выходной класс (положительная или отрицательная тональность).

```
        # Вывод результатов
        print("Predicted sentiment:", predicted_sentiment)
        print("Probability:",
              round(probabilities.prob(predicted_sentiment), 2))
```

Полный код этого примера содержится в файле `sentiment_analyzer.py`. Выполнив этот код, вы увидите следующий вывод в окне терминала (рис. 10.9).

```
Number of training datapoints: 1600
Number of test datapoints: 400

Accuracy of the classifier: 0.735

Top 15 most informative words:
1. outstanding
2. insulting
3. vulnerable
4. ludicrous
5. uninvolving
6. astounding
7. avoids
8. fascination
9. symbol
10. seagal
11. affecting
12. anna
13. darker
14. animators
15. idiotic
```

Рис. 10.9

На этом экранном снимке отображены первые 15 наиболее информативных слов. Прокрутив окно терминала, вы увидите следующий вывод (рис. 10.10).

```
Movie review predictions:

Review: The costumes in this movie were great
Predicted sentiment: Positive
Probability: 0.59

Review: I think the story was terrible and the characters were very weak
Predicted sentiment: Negative
Probability: 0.8

Review: People say that the director of the movie is amazing
Predicted sentiment: Positive
Probability: 0.6

Review: This is such an idiotic movie. I will not recommend it to anyone.
Predicted sentiment: Negative
Probability: 0.87
```

Рис. 10.10

Эта информация убеждает нас в том, что все прогнозы оказались корректными.

Тематическое моделирование с использованием латентного размещения Дирихле

Тематическое моделирование — это процесс идентификации шаблонов в текстовых данных, которые соответствуют некоторой теме. Если текст содержит несколько тем, то эта методика может быть использована для идентификации и разделения тем в пределах входного текста. Это делается для обнаружения скрытой тематической структуры в данном наборе документов.

Тематическое моделирование помогает оптимально организовать документы, которые впоследствии могут быть использованы для анализа. Следует отметить одну особенность алгоритмов тематического моделирования, которая заключается в том, что в этом случае мы не нуждаемся ни в каких помеченных данных. Это напоминает ситуацию с обучением без учителя, когда идентификация шаблонов осуществляется без использования маркеров. С учетом огромных объемов текстовых данных, генерируемых в Интернете,

тематическое моделирование приобретает особую важность, поскольку обеспечивает возможность *суммаризации* (summerization) всех этих данных, что иначе было бы невозможным.

Латентное размещение Дирихле (Latent Dirichlet Allocation) — это метод тематического моделирования, базовая предпосылка которого состоит в том, что каждый заданный фрагмент документа представляет собой смесь нескольких тем. В качестве примера рассмотрим следующее предложение: "Data visualization is an important tool in financial analysis". В этом предложении затрагиваются несколько тем, таких как данные, визуализация, финансы и др. Такое конкретное сочетание помогает нам идентифицировать данный текст в большом документе. В сущности, это статистическая модель, которая пытается воспользоваться идеей сочетания тем и построить модель на ее основе. В данной модели предполагается, что документы генерируются в ходе случайного процесса, основанного на этих темах. При таком подходе *тема* — это распределение слов из фиксированного словаря. Рассмотрим, как выполняется тематическое моделирование в Python.

В этом разделе мы будем использовать библиотеку `gensim`. Мы уже установили ее в первом разделе главы. Убедитесь в этом, прежде чем продолжить. Создайте новый файл Python и импортируйте следующие файлы.

```
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from gensim import models, corpora
```

Определим функцию, загружающую входные данные. Входной файл содержит 10 предложений, разделенных символами новой строки.

```
# Загрузка входных данных
def load_data(input_file):
    data = []
    with open(input_file, 'r') as f:
        for line in f.readlines():
            data.append(line[:-1])
    return data
```

Определим функцию, обрабатывающую входной текст. Первый шаг состоит в токенизации текста.

```
# Функция обработки, предназначенная для токенизации текста,
# удаления стоп-слов и выполнения стемминга
def process(input_text):
    # Создание регулярного выражения для токенизатора
    tokenizer = RegexpTokenizer(r'\w+')
```

Нам нужно выполнить стемминг токенизированного текста.

```
# Создание стеммера Сноуболла
stemmer = SnowballStemmer('english')
```

Кроме того, мы должны удалить стоп-слова из входного текста, поскольку они не добавляют никакой информации. Получим список стоп-слов.

```
# Получение списка стоп-слов
stop_words = stopwords.words('english')
```

Токенизируем входную строку.

```
# Токенизация входной строки
tokens = tokenizer.tokenize(input_text.lower())
```

Удалим стоп-слова.

```
# Удаление стоп-слов
tokens = [x for x in tokens if not x in stop_words]
```

Выполним стемминг токенизированных слов и вернем список.

```
# Выполнение стемминга токенизированных слов
tokens_stemmed = [stemmer.stem(x) for x in tokens]
return tokens_stemmed
```

Определим основную функцию и загрузим входные данные из файла `data.txt`, который вам предоставляется.

```
if __name__=='__main__':
    # Загрузка входных данных
    data = load_data('data.txt')
```

Токенизируем текст.

```
# Создание списка токенов предложений
tokens = [process(x) for x in data]
```

Создадим словарь на основе токенизированных предложений.

```
# Создание словаря на основе токенизированных предложений
dict_tokens = corpora.Dictionary(tokens)
```

Создадим терм-документную матрицу, используя токены предложений.

```
# Создание терм-документной матрицы
doc_term_mat = [dict_tokens.doc2bow(token) for token in tokens]
```

В качестве входного параметра мы должны предоставить количество тем. В данном случае нам известно, что входной текст содержит две различные темы. Укажем это.

```
# Определим количество тем для LDA-модели
num_topics = 2
```

Сгенерируем латентную модель Дирихле.

```
# Генерирование LDA-модели
ldamodel = models.ldamodel.LdaModel(doc_term_mat,
        num_topics=num_topics, id2word=dict_tokens, passes=25)
```

Выведем для каждой темы первые 5 представительных слов.

```
num_words = 5
print('\nTop ' + str(num_words) + ' contributing words to each
      topic:')
for item in ldamodel.print_topics(num_topics=num_topics,
                                  num_words=num_words):
    print('\nTopic', item[0])

    # Вывод представительных слов вместе с их
    # относительными вкладами
    list_of_strings = item[1].split(' + ')
    for text in list_of_strings:
        weight = text.split('*')[0]
        word = text.split('*')[1]
        print(word, '==>', str(round(float(weight) * 100,
              2)) + '%')
```

Полный код этого примера содержится в файле `topic_modeler.py`. Выполнив его, вы увидите в окне терминала следующий вывод (рис. 10.11).

Как видите, программа неплохо справилась с задачей разделения двух тем: математики и истории. Если вы обратитесь к тексту, то убедитесь в том, что речь в нем идет либо о математике, либо об истории.

```
Top 5 contributing words to each topic:

Topic 0
mathemat ==> 2.7%
structur ==> 2.6%
set ==> 2.6%
formul ==> 2.6%
tradit ==> 1.6%

Topic 1
empir ==> 4.7%
expand ==> 3.3%
time ==> 2.0%
peopl ==> 2.0%
histor ==> 2.0%
```

Рис. 10.11

Резюме

В этой главе вы познакомились с различными базовыми понятиями, относящимися к обработке естественного языка. Вы узнали о том, что такое токенизация и как разделить входной текст на отдельные токены. Мы обсудили приведение слов к их корневым формам посредством стемминга и лемматизации и реализовали инструмент, разделяющий тест на блоки в соответствии с заданными условиями.

Мы обсудили модель Bag of Words (мешок слов) и создали терм-документную матрицу для входного текста. Вы узнали о том, как категоризировать текст, используя машинное обучение. Мы создали идентификатор грамматических родов, используя эвристику. Затем мы использовали машинное обучение для сентимент-анализа отзывов о фильмах. После этого мы обсудили тематическое моделирование и реализовали систему для анализа тем в заданном документе.

В следующей главе мы расскажем о моделировании последовательных данных посредством скрытых марковских моделей и используем такую модель для анализа биржевых курсов.

11

Вероятностный подход к обработке последовательных данных

В этой главе вы познакомитесь с последовательными моделями обучения. Мы поговорим об обработке временных рядов в Pandas. Вы узнаете о выделении срезов временных рядов данных и выполнении различных операций над ними. Мы обсудим извлечение различных статистик из данных временных рядов на скользящей основе. Далее будет рассказано о скрытых марковских моделях и реализации системы для создания таких моделей. Вы узнаете о том, как использовать условные случайные поля для анализа буквенных последовательностей. В завершение мы обсудим использование изложенных методов для анализа биржевых курсов.

К концу главы вы освоите следующие темы:

- обработка временных рядов с помощью библиотеки Pandas;
- извлечение срезов временных рядов данных;
- выполнение операций над временными рядами;
- извлечение статистик из временных рядов данных;
- генерирование данных с использованием скрытых марковских моделей;
- идентификация буквенных последовательностей с помощью условных случайных полей;
- анализ биржевых курсов.

Что такое последовательные данные

В мире машинного обучения мы сталкиваемся со многими типами данных, такими как изображения, текст, видео и т.п. Различные типы данных требуют различных типов моделирования. Термин *последовательные данные*

относится к данным, порядок следования которых имеет значение. Временные ряды — это частный случай последовательных данных.

В основном именно значения с отметками времени поступают от таких источников данных, как датчики, микрофоны, биржевые рынки и т.п. Временные ряды обладают множеством важных характеристик, нуждающихся в моделировании для эффективного анализа данных.

Измерения, связанные с временными рядами, должны производиться через регулярные промежутки времени и соответствовать предварительно установленным параметрам. Результаты этих измерений сохраняются вместе с отметками времени, и поэтому порядок их появления очень важен. Мы используем этот порядок для установления закономерностей, существующих между данными.

В этой главе будет показано, как строить модели, описывающие временные ряды и вообще любые последовательные данные. Эти модели помогают понять поведение переменных временного ряда, а затем применяются для прогнозирования будущих результатов на основе знания поведения переменных в прошлом.

Временные ряды интенсивно используются в финансовом анализе, экономике, производстве, анализе данных сенсоров, а также в системах распознавания речи, при составлении прогнозов погоды и т.п. Мы исследуем целый ряд сценариев, в которых встречаются временные ряды, и продемонстрируем способы создания решений. Для выполнения всех операций, связанных с временными рядами, мы используем библиотеку Pandas. Кроме того, на протяжении главы мы будем использовать и другие пакеты, в частности `hmmlearn` и `pystruc`. Обязательно установите их, прежде чем продолжить чтение.

Чтобы установить указанные пакеты, выполните в окне терминала следующие команды.

```
$ pip3 install pandas
$ pip3 install hmmlearn
$ pip3 install pystruct
$ pip3 install cvxopt
```

В случае вывода сообщения об ошибке в процессе установки пакета `cvxopt` перейдите по адресу `http://cvxopt.org/install` для получения дальнейших инструкций. После успешной установки пакетов можете приступать к чтению следующего раздела.

Обработка временных рядов с помощью библиотеки Pandas

Начнем с изучения того, как обрабатывать временные ряды данных с помощью библиотеки Pandas. В этом разделе мы преобразуем последовательность чисел во временной ряд и визуализируем его. Библиотека Pandas предоставляет возможности снабжения данных временными метками, структурирования данных и эффективного выполнения операций над ними.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Определим функцию, осуществляющую чтение данных из входного файла. Параметр index указывает номер столбца, содержащего соответствующие данные.

```
def read_data(input_file, index):
    # Чтение данных из входного файла
    input_data = np.loadtxt(input_file, delimiter=',')
```

Определим лямбда-функцию, преобразующую строки в формат данных Pandas.

```
    # Лямбда-функция для преобразования
    # строк в формат данных Pandas
    to_date = lambda x, y: str(int(x)) + '-' + str(int(y))
```

Используем эту лямбда-функцию для получения начальной даты из первой строки входного файла.

```
    # Извлечение начальной даты
    start = to_date(input_data[0, 0], input_data[0, 1])
```

Мы должны предоставить библиотеке Pandas конечную дату для ограничения временного диапазона, прежде чем выполнять операции с ее помощью, поэтому увеличим значение поля даты в последней строке на один месяц.

```
    # Извлечение конечной даты
    if input_data[-1, 1] == 12:
        year = input_data[-1, 0] + 1
        month = 1
    else:
        year = input_data[-1, 0]
        month = input_data[-1, 1] + 1
        end = to_date(year, month)
```

Создадим список индексов с датами, используя начальную и конечную даты с ежемесячной частотой.

```
    # Создание списка дат с ежемесячной частотой
    date_indices = pd.date_range(start, end, freq='M')
```

Создадим ряд данных, используя отметки времени.

```
    # Добавление меток во входные данные для создания
    # временного ряда данных
    output = pd.Series(input_data[:, index], index=date_indices)

    return output
```

Определим основную функцию и укажем входной файл.

```
if __name__=='__main__':
    # Имя входного файла
    input_file = 'data_2D.txt'
```

Зададим столбцы, содержащие данные.

```
    # Указание столбцов, подлежащих преобразованию
    # во временной ряд данных
    indices = [2, 3]
```

Выполним итерации по столбцам и прочитаем данные каждого из них.

```
    # Итерирование по столбцам и построение графика данных
    for index in indices:
        # Преобразование столбца в формат временного ряда
        timeseries = read_data(input_file, index)
```

Построим график данных временного ряда.

```
        # Построение графика
        plt.figure()
        timeseries.plot()
        plt.title('Размерность ' + str(index - 1))
    plt.show()
```

Полный код примера содержится в файле `timeseries.py`. В процессе этого кода на экране отобразятся два графика.

На первом снимке экрана представлены данные, соответствующие первому измерению (рис. 11.1).

На втором снимке экрана представлены данные, соответствующие второму измерению (рис. 11.2).

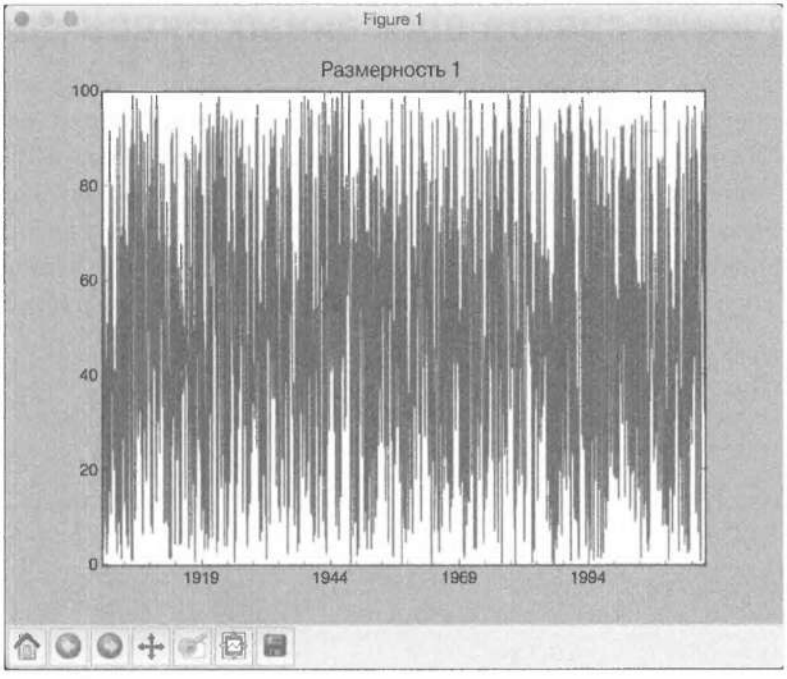

Рис. 11.1

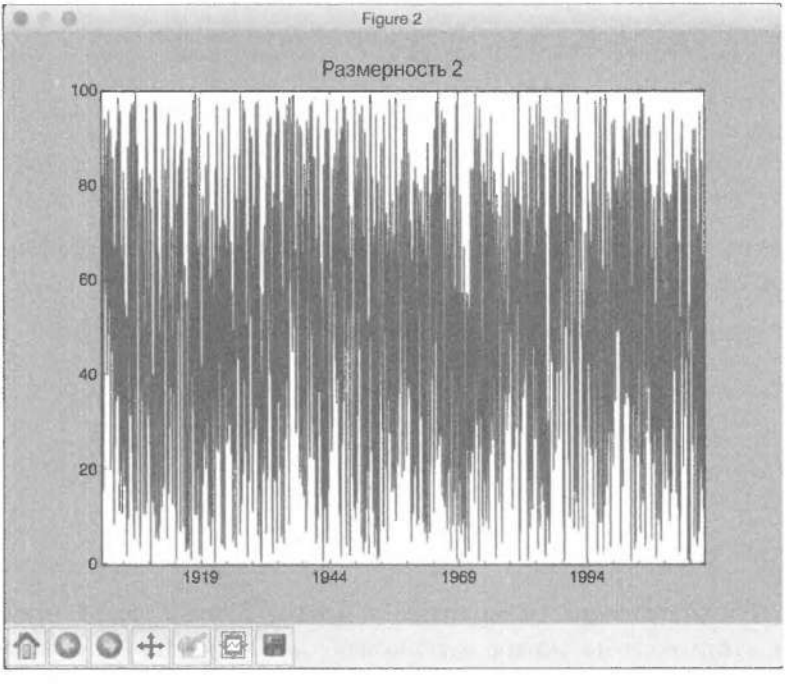

Рис. 11.2

Извлечение срезов временных рядов данных

Теперь, когда вы научились обрабатывать временные ряды данных, мы можем перейти к рассмотрению извлечения их срезов. Процесс извлечения срезов — это разбиение данных на различные подынтервалы и извлечение соответствующей информации. Эта возможность оказывается очень полезной при работе с наборами данных временных рядов. Для извлечения срезов наших данных мы будем использовать отметки времени вместо индексов.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from timeseries import read_data
```

Загрузим третий столбец (индексация ведется с нуля) из файла входных данных.

```
# Загрузка входных данных
index = 2
data = read_data('data_2D.txt', index)
```

Определим начальный и конечный годы и построим график с гранулярностью на уровне года.

```
# Построение графика данных с гранулярностью на уровне года
start = '2003'
end = '2011'
plt.figure()
data[start:end].plot()
plt.title('Входные данные с' + start + ' по ' + end)
```

Определим начальный и конечный месяцы и построим график с гранулярностью на уровне месяца.

```
# Построение графика данных с гранулярностью на уровне месяца
start = '1998-2'
end = '2006-7'
plt.figure()
data[start:end].plot()
plt.title('Входные данные с' + start + ' по ' + end)

plt.show()
```

Полный код примера содержится в файле `slicer.py`. В процессе выполнения этого кода на экране отобразятся два графика. На первом снимке экрана представлены данные, соответствующие периоду с 2003 по 2011 годы (рис. 11.3).

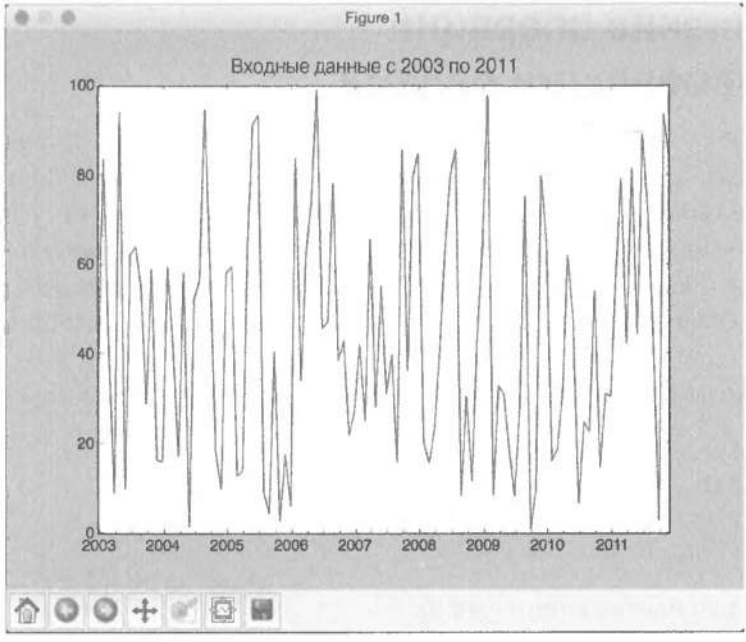

Рис. 11.3

На втором снимке экрана представлены данные, соответствующие периоду с февраля 1998 года по июль 2006 года (рис. 11.4).

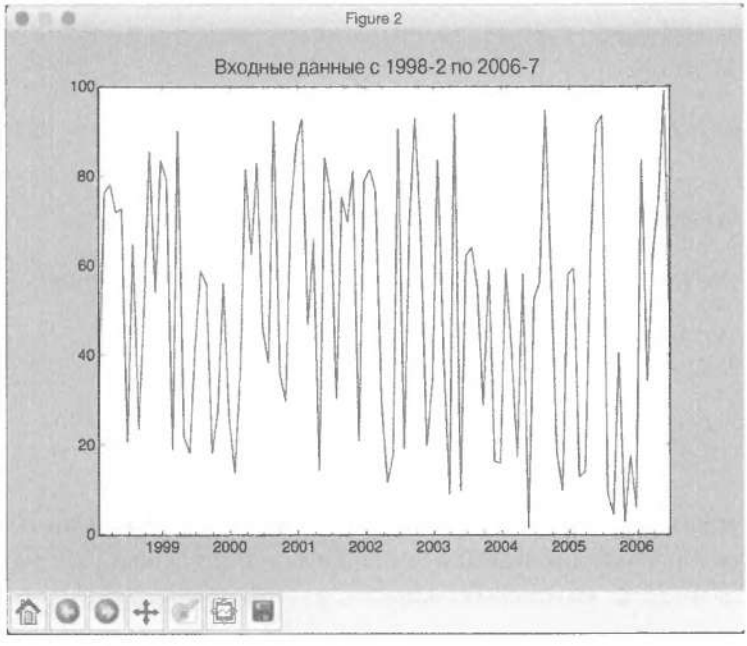

Рис. 11.4

Выполнение операций над временными рядами

Библиотека Pandas обеспечивает эффективную работу с временными рядами данных и выполнение над ними таких операций, как фильтрация и добавление данных. Вам нужно всего лишь задать требуемые условия, чтобы Pandas выполнила фильтрацию набора данных и вернула желаемый результат. Вы также можете добавлять две переменные временных рядов. Это позволяет быстро создавать различные приложения, не изобретая заново колесо.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from timeseries import read_data
```

Определим имя входного файла.

```
# Имя входного файла
input_file = 'data_2D.txt'
```

Загрузим третий и четвертый столбцы в отдельные переменные.

```
# Загрузка данных
x1 = read_data(input_file, 2)
x2 = read_data(input_file, 3)
```

Создадим объект `DataFrame` в Pandas, присвоив имена двум измерениям.

```
# Создание фрейма данных Pandas для извлечения срезов
data = pd.DataFrame({'dim1': x1, 'dim2': x2})
```

Построим график данных, задав начальный и конечный годы.

```
# Построение графика
start = '1968'
end = '1975'
data[start:end].plot()
plt.title('Наложение двух графиков')
```

Отфильтруем данные в соответствии с заданными условиями и отобразим их. В данном случае мы возьмем все значения из `dim1`, которые меньше `45`, и все значения из `dim2`, которые больше `30`.

```
# Фильтрация с использованием условий
# - 'dim1'- значения меньше заданного порога
# - 'dim2' - значения больше заданного порога
data[(data['dim1'] < 45) & (data['dim2'] > 30)].plot()
plt.title('dim1 < 45 и dim2 > 30')
```

Мы также можем добавить два ряда в Pandas. Добавим ряды `dim1` и `dim2` между заданными начальной и конечной датами.

```
# Сложение двух фреймов данных
plt.figure()
diff = data[start:end]['dim1'] + data[start:end]['dim2']
diff.plot()
plt.title('Результат суммирования (dim1 + dim2)')

plt.show()
```

Полный код примера содержится в файле `operator.py`. В процессе выполнения этого кода на экране отобразятся три графика. На первом снимке экрана представлены данные, соответствующие периоду с 1968 по 1975 годы (рис. 11.5).

Рис. 11.5

На втором снимке экрана представлены отфильтрованные данные (рис. 11.6).

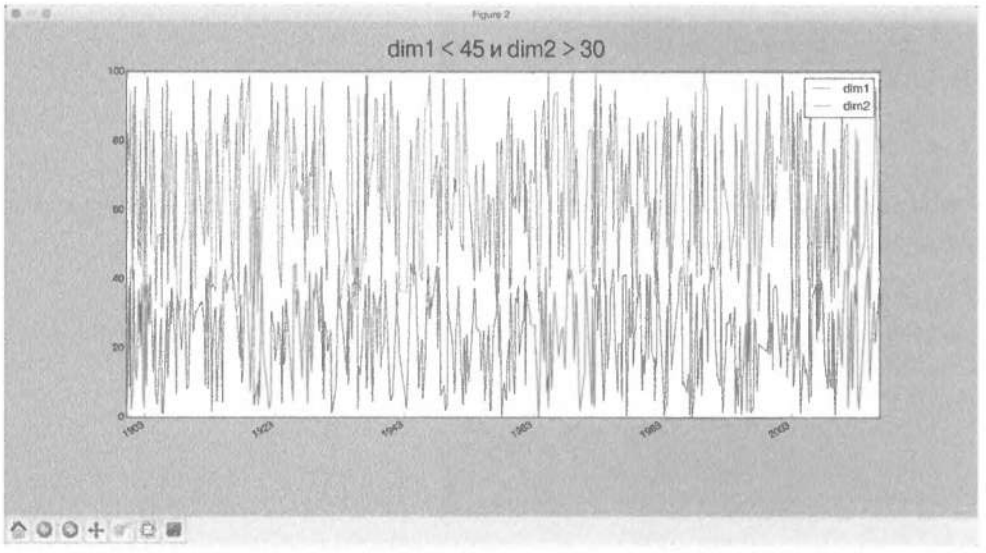

Рис. 11.6

На третьем снимке экрана представлен результат суммирования данных (рис. 11.7).

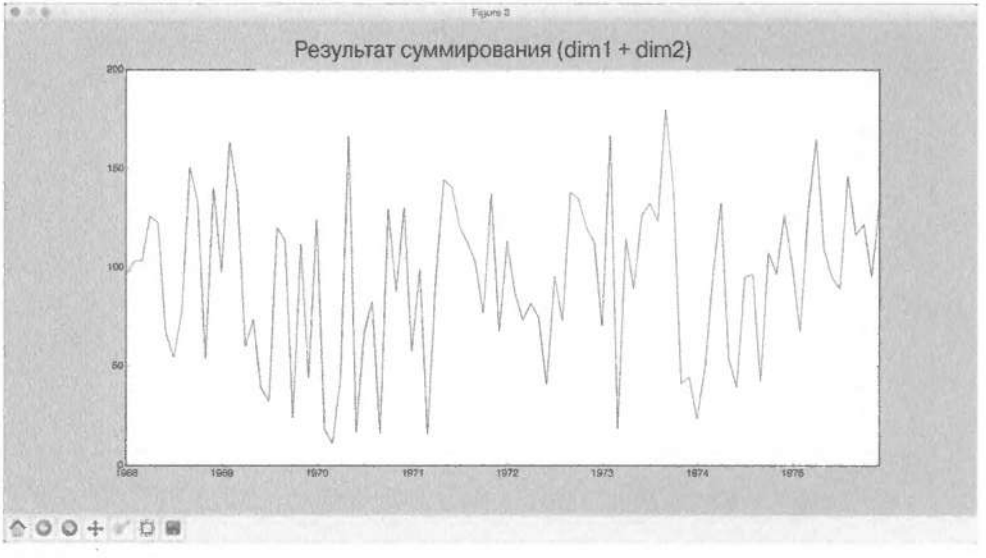

Рис. 11.7

Извлечение статистик из временных рядов данных

Для извлечения смысла, содержащегося в данных временных рядов, мы должны обращаться к вычислению статистик на их основе. Этими статистиками могут быть среднее значение, дисперсия, корреляция, максимальное значение и т.п. Эти статистики должны вычисляться на скользящей основе с использованием окна заданного размера. Визуализируя изменения этих статистик со временем, мы будем наблюдать интересные закономерности. Рассмотрим конкретный пример извлечения статистик из временных рядов данных.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from timeseries import read_data
```

Определим имя входного файла.

```
# Имя входного файла
input_file = 'data_2D.txt'
```

Загрузим третий и четвертый столбцы в отдельные переменные.

```
# Загрузка входных данных в формате временных рядов
x1 = read_data(input_file, 2)
x2 = read_data(input_file, 3)
```

Создадим фрейм данных Pandas, присвоив имена двум измерениям.

```
# Создание фрейма данных Pandas для извлечения срезов
data = pd.DataFrame({'dim1': x1, 'dim2': x2})
```

Извлечем максимальное и минимальное значения вдоль каждого измерения.

```
# Извлечение максимального и минимального значений
print('\nMaximum values for each dimension:')
print(data.max())
print('\nMinimum values for each dimension:')
print(data.min())
```

Извлечение общего среднего и средних по строкам для первых *12* строк.

```
# Извлечение общего среднего и среднего по строкам
print('\nOverall mean:')
print(data.mean())
print('\nRow-wise mean:')
print(data.mean(1)[:12])
```

Построим график скользящего среднего, используя окно шириной 24.

```
# Построение графика скользящего среднего
# с использованием окна шириной 24
data.rolling(center=False, window=24).mean().plot()
plt.title('Скользящее среднее')
```

Выведем значения коэффициентов корреляции.

```
# Извлечение коэффициентов корреляции
print('\nCorrelation coefficients:\n', data.corr())
```

Построим график скользящего коэффициента корреляции, используя окно шириной 60.

```
# Построение графика скользящей корреляции
# с использованием окна шириной 60
plt.figure()
plt.title('Скользящий коэффициент корреляции')
data['dim1'].rolling(window=60).corr(other=data['dim2']).plot()
plt.show()
```

Полный код примера содержится в файле `stats_extractor.py`. В процессе выполнения кода на экране отобразятся два графика. На первом снимке экрана отображено скользящее среднее (рис. 11.8).

На втором снимке экрана отображен скользящий коэффициент корреляции (рис. 11.9).

В окне терминала вы увидите следующий вывод (рис. 11.10).

Прокрутив окно, вы увидите значения построчных средних и коэффициентов корреляции (рис. 11.11).

Отображенные на предыдущих рисунках значения корреляции указывают на степень корреляции между каждым измерением и всеми остальными измерениями. Значение `1.0` указывает на идеальную корреляцию, тогда как значение `0.0` указывает на то, что переменные никак не связаны между собой.

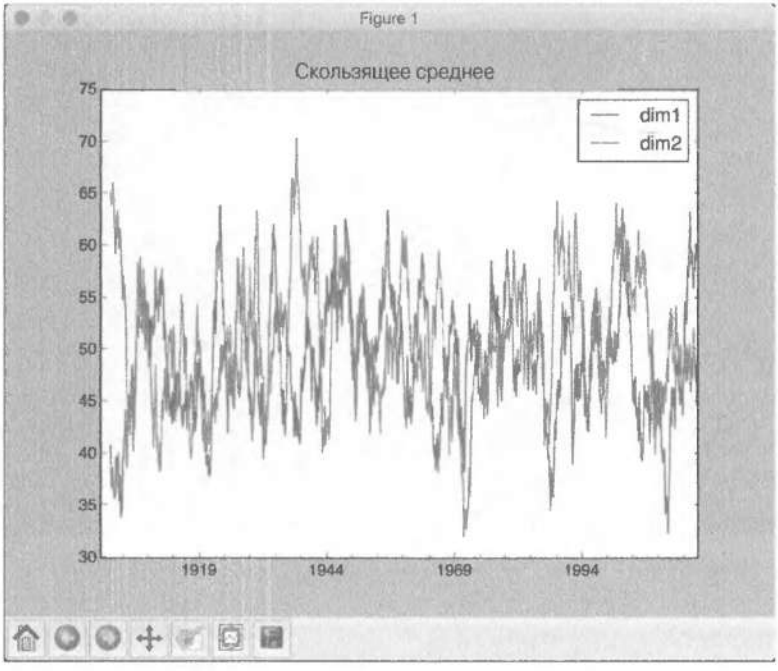

Рис. 11.8

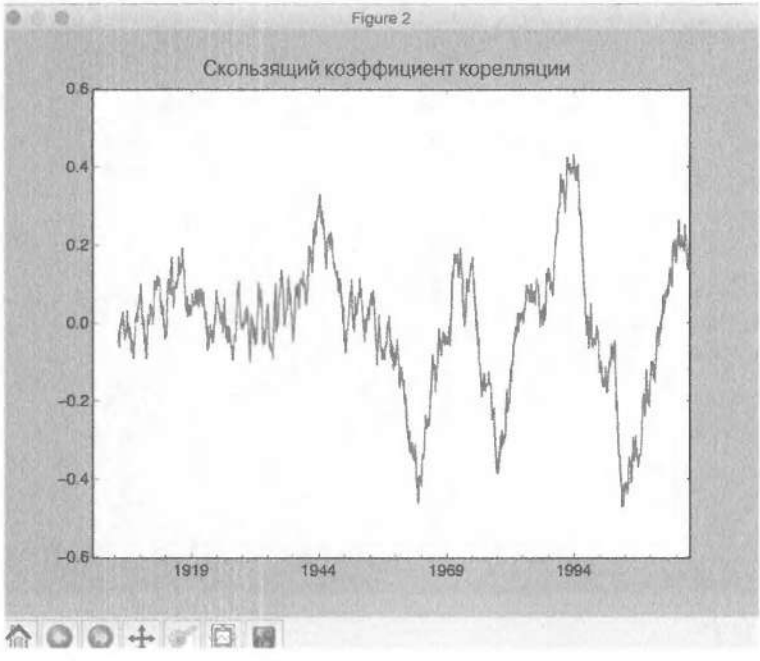

Рис. 11.9

```
Maximum values for each dimension:
dim1    99.98
dim2    99.97
dtype: float64

Minimum values for each dimension:
dim1    0.18
dim2    0.16
dtype: float64

Overall mean:
dim1    49.030541
dim2    50.983291
dtype: float64
```

Рис. 11.10

```
Row-wise mean:
1900-01-31    85.595
1900-02-28    75.310
1900-03-31    27.700
1900-04-30    44.675
1900-05-31    31.295
1900-06-30    44.160
1900-07-31    67.415
1900-08-31    56.160
1900-09-30    51.495
1900-10-31    61.260
1900-11-30    30.925
1900-12-31    30.785
Freq: M, dtype: float64

Correlation coefficients:
          dim1     dim2
dim1  1.00000  0.00627
dim2  0.00627  1.00000
```

Рис. 11.11

Генерация данных с использованием скрытых марковских моделей

Скрытая марковская модель (Hidden Markov Model — HMM) — мощная методика анализа последовательных данных. В ней предполагается, что моделируемая система представляет собой марковский процесс со скрытыми состояниями. Это означает, что базовая система может находиться в одном из возможных состояний некоторого набора. Последовательно переходя из одного состояния в другое, она производит последовательность результатов. Мы можем наблюдать лишь результаты, но не состояния. Следовательно, эти состояния скрыты от нас. Нашей целью является моделирование данных таким образом, чтобы можно было делать выводы относительно переходов из одного состояния в другое для неизвестных данных.

Чтобы лучше понять модель HMM, рассмотрим в качестве примера коммивояжера, которому по роду работы приходится курсировать между следующими тремя городами: Лондоном, Барселоной и Нью-Йорком. Его целью является минимизация времени поездок, чтобы сделать их более эффективными. Учитывая круг его обязанностей и рабочий график, мы имеем дело с набором значений вероятности, которые диктуют шансы поездок из города X в город Y. В приведенной ниже информации *P(X-> Y)* — это вероятность поездки из города *X* в город *Y*.

P(London -> London) = 0.10

P(London -> Barcelona) = 0.70

P(London -> NY) = 0.20

P(Barcelona -> Barcelona) = 0.15

P(Barcelona -> London) = 0.75

P(Barcelona -> NY) = 0.10

P(NY -> NY) = 0.05

P(NY -> London) = 0.60

P(NY -> Barcelona) = 0.35

Представим эту информацию в виде матрицы переходов.

London Barcelona NY

London 0.10 0.70 0.20

Barcelona 0.75 0.15 0.10

NY 0.60 0.35 0.05

Теперь, когда мы располагаем всей этой информацией, мы можем продвинуться дальше и сформулировать задачу. Коммивояжер начинает свое путешествие во вторник, отправляясь из Лондона, и должен запланировать какой-то город на пятницу. Но все будет зависеть от того, где он будет на тот момент находиться. Какова вероятность того, что в пятницу он попадет в Барселону? Приведенная выше таблица поможет нам это определить.

Не имея марковской цепи для моделирования задачи, мы не будем знать, как может выглядеть график поездок коммивояжера. Нашей целью является возможность предсказания с большой степенью достоверности, в каком городе он будет находиться в определенный день. Если мы обозначим матрицу перехода как *T*, а текущий день — как *X(i)*, то:

$$X(i+1) = X(i).T$$

В нашем случае пятница отстоит от вторника на 3 дня. Это означает, что мы должны вычислить величину *X(i+3)*. Соответствующие вычисления будут выглядеть примерно так:

$$X(i+1) = X(i).T$$

$$X(i+2) = X(i+1).T$$

$$X(i+3) = X(i+2).T$$

Поэтому в итоге:

$$X(i+3) = X(i).T^3$$

Мы должны задать *X(i)* в таком виде:

$$X(i) = [0.10\ 0.70\ 0.20]$$

Следующий шаг заключается в вычислении куба матрицы. В сети доступно множество инструментов для выполнения матричных операций, например тот, который предлагается по адресу `http://matrix.reshish.com/multiplication.php`. Выполнив необходимые матричные вычисления, вы убедитесь в том, что для вторника получаются следующие значения вероятностей:

$$P(London) = 0.31$$

$$P(Barcelona) = 0.53$$

$$P(NY) = 0.16$$

Мы видим, что вероятность его пребывания в Барселоне больше, чем для любого другого города. Это правдоподобно с географической точки зрения, поскольку Барселона находится ближе к Лондону, чем Нью-Йорк. Рассмотрим, как смоделировать HMM в Python.

Создадим новый файл Python и импортируем следующие пакеты.

```
import datetime

import numpy as np
import matplotlib.pyplot as plt
from hmmlearn.hmm import GaussianHMM

from timeseries import read_data
```

Загрузим данные из входного файла.

```
# Загрузка входных данных
data = np.loadtxt('data_1D.txt', delimiter=',')
```

Извлечем третий столбец для тренировки.

```
# Извлечение столбца данных (третий по счету) для тренировки
X = np.column_stack([data[:, 2]])
```

Создадим гауссовскую модель HMM с 5 компонентами и диагональной ковариацией.

```
# Создание гауссовской модели HMM
num_components = 5
hmm = GaussianHMM(n_components=num_components,
                  covariance_type='diag', n_iter=1000)
```

Выведем значения среднего и дисперсии для каждого компонента HMM.

```
# Вывод статистик HMM
print('\nMeans and variances:')
for i in range(hmm.n_components):
print('\nHidden state', i+1)
print('Mean =', round(hmm.means_[i][0], 2))
print('Variance =', round(np.diag(hmm.covars_[i])[0], 2))
```

Сгенерируем 1200 выборочных данных, используя обученную модель HMM, и построим соответствующий график.

```
# Генерирование данных с использованием модели HMM
num_samples = 1200
generated_data, _ = hmm.sample(num_samples)
plt.plot(np.arange(num_samples), generated_data[:, 0], c='black')
plt.title('Сгенерированные данные')
plt.show()
```

Полный код примера содержится в файле `hmm.py`. В процессе выполнения кода на экране отобразится следующий график, отображающий 1200 сгенерированных образцов данных (рис. 11.12).

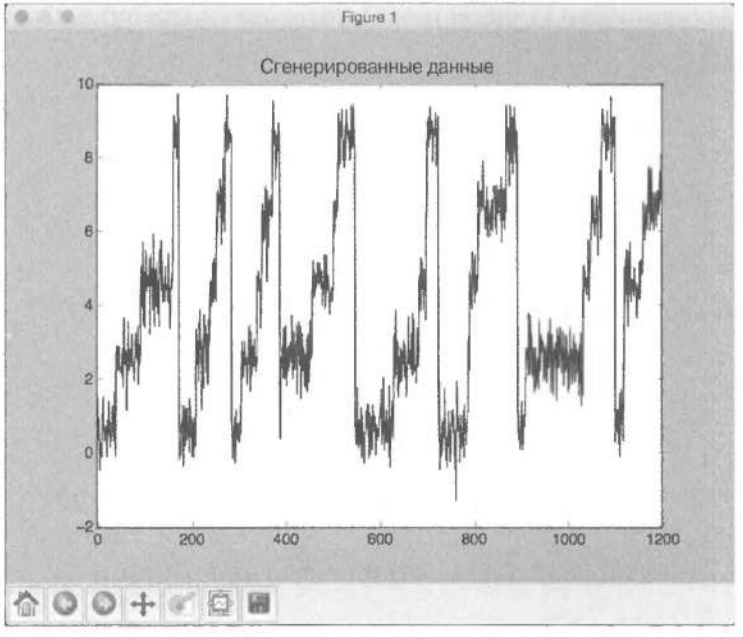

Рис. 11.12

В окне терминала отобразится следующая информация (рис. 11.13).

```
Training the Hidden Markov Model...

Means and variances:

Hidden state 1
Mean = 4.6
Variance = 0.25

Hidden state 2
Mean = 6.59
Variance = 0.25

Hidden state 3
Mean = 0.6
Variance = 0.25

Hidden state 4
Mean = 8.6
Variance = 0.26

Hidden state 5
Mean = 2.6
Variance = 0.26
```

Рис. 11.13

Идентификация буквенных последовательностей с помощью условных случайных полей

Условные случайные поля (Conditional Random Fields — CRF) — это вероятностные модели, которые часто применяют для анализа структурированных данных. Мы используем их для маркирования и сегментирования последовательных данных в различных формах. Следует отметить, что модели CRF — *дискриминантные*, в противоположность моделям HMM, являющимся *генеративными* (порождающими) моделями.

Мы можем определить условное распределение вероятностей для помеченной последовательности измерений. Мы используем его для построения CRF-модели. В моделях HMM мы должны определять совместное распределение для последовательности наблюдений и меток.

Одним из главных преимуществ CRF-моделей является то, что они условны по своей природе. В случае HMM-моделей это не так. В моделях CRF независимость переменных никоим образом не предполагается, тогда как в моделях HMM предполагается, что результат, полученный в любой момент времени, является статистически независимым от предыдущих результатов. В HMM-моделях это предположение необходимо для гарантии того, что процесс вывода будет работать надежно. Однако это предположение не всегда справедливо! Реальные данные всегда наполнены временными зависимостями.

Обычно модели CRF превосходят модели HMM в ряде приложений, таких как обработка естественного языка, распознавание речи, биотехнологии и т.п. В этом разделе мы будем обсуждать использование CRF для анализа последовательностей букв. Создайте новый файл Python и импортируйте следующие пакеты.

```
import os
import argparse
import string
import pickle

import numpy as np
import matplotlib.pyplot as plt
from pystruct.datasets import load_letters
from pystruct.models import ChainCRF
from pystruct.learners import FrankWolfeSSVM
```

Определим функцию для анализа входных аргументов. Мы можем передать значение C в качестве входного параметра. Параметр C управляет размером штрафа за неправильную классификацию. Более высокие значения C означают, что мы налагаем более высокие штрафы за ошибочную классификацию в процессе тренировки, но это может приводить к переобучению модели. С другой стороны, выбирая более низкие значения C, мы снижаем специфичность модели. Но это также означает, что мы налагаем более низкие штрафы за неверную классификацию в процессе обучения на тренировочных точках данных.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Trains a\
            Conditional Random Field classifier')
    parser.add_argument("--C", dest="c_val", required=False,
            type=float, default=1.0, help='C value to be used \
            for training')
    return parser
```

Определим класс, обрабатывающий всю функциональность построения CRF-модели. Мы будем использовать цепочечную модель CRF с помощью объекта `FrankWolfeSSVM`.

```
# Класс, моделирующий CRF
class CRFModel(object):
    def __init__(self, c_val=1.0):
        self.clf = FrankWolfeSSVM(model=ChainCRF(),
                                  C=c_val, max_iter=50)
```

Определим функцию, загружающую тренировочные данные.

```
    # Загрузка тренировочных данных
    def load_data(self):
        alphabets = load_letters()
        X = np.array(alphabets['data'])
        y = np.array(alphabets['labels'])
        folds = alphabets['folds']

        return X, y, folds
```

Определим функцию, обучающую модель CRF.

```
    # Тренировка CRF
    def train(self, X_train, y_train):
        self.clf.fit(X_train, y_train)
```

Определим функцию, вычисляющую точность модели CRF.

```
    # Вычисление точности модели CRF
    def evaluate(self, X_test, y_test):
        return self.clf.score(X_test, y_test)
```

Определим функцию, которая выполняет обученную модель CRF для неизвестных данных.

```
    # Выполнение CRF для неизвестных данных
    def classify(self, input_data):
        return self.clf.predict(input_data)[0]
```

Определим функцию, извлекающую подстроку из буквенной последовательности на основании списка индексов.

```
# Преобразование индексов в буквы
def convert_to_letters(indices):
    # Создание массива Numpy всех букв алфавита
    alphabets = np.array(list(string.ascii_lowercase))
```

Извлечем буквы.

```
    # Извлечение букв на основании индексов
    output = np.take(alphabets, indices)
    output = ''.join(output)

    return output
```

Определим основную функцию и разберем входные аргументы.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    c_val = args.c_val
```

Создадим объект CRF-модели.

```
    # Создание CRF-модели
    crf = CRFModel(c_val)
```

Загрузим входные данные и разделим их на тренировочный и тестовый наборы.

```
    # Загрузка тренировочных и тестовых данных
    X, y, folds = crf.load_data()
    X_train, X_test = X[folds == 1], X[folds != 1]
    y_train, y_test = y[folds == 1], y[folds != 1]
```

Обучим CRF-модель.

```
# Обучение CRF-модели
print('\nTraining the CRF model...')
crf.train(X_train, y_train)
```

Вычислим точность модели CRF и выведем ее.

```
# Вычисление точности
score = crf.evaluate(X_test, y_test)
print('\nAccuracy score =', str(round(score*100, 2)) + '%')
```

Выполним модель для тестовых точек данных и выведем результат.

```
indices = range(3000, len(y_test), 200)
for index in indices:
    print("\nOriginal =", convert_to_letters(y_test[index]))
    predicted = crf.classify([X_test[index]])
    print("Predicted =", convert_to_letters(predicted))
```

Полный код этого примера содержится в файле `crf.py`. Выполнив этот код, вы увидите следующий вывод в окне терминала (рис. 11.14).

```
Training the CRF model...

Accuracy score = 77.93%

Original  = rojections
Predicted = rojectiong

Original  = uff
Predicted = ufr

Original  = kiing
Predicted = kiing

Original  = ecompress
Predicted = ecomertig

Original  = uzz
Predicted = vex

Original  = poiling
Predicted = aciting
```

Рис. 11.14

Прокрутив окно до конца, вы увидите следующий вывод (рис. 11.15).

```
Original  = abulously
Predicted = abuloualy

Original  = ormalization
Predicted = ormalisation

Original  = ake
Predicted = aka

Original  = afeteria
Predicted = ateteria

Original  = obble
Predicted = obble

Original  = hadow
Predicted = habow

Original  = ndustrialized
Predicted = ndusqrialyled

Original  = ympathetically
Predicted = ympnshetically
```

Рис. 11.15

Мы видим, что большинство слов было предсказано правильно.

Анализ биржевого рынка

В этом разделе мы будем анализировать данные биржевого рынка, задействуя скрытые марковские модели. Это пример данных, уже организованных и снабженных отметками времени. Мы будем использовать набор данных, доступный в пакете `matplotlib`. В нем содержится информация о стоимости пакетов акций различных компаний на протяжении нескольких лет. Скрытые марковские модели — это порождающие модели, способные анализировать временные ряды данных и извлекать лежащую в их основе структуру. Мы применим эту модель для анализа колебаний стоимости акций и генерации результатов.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import datetime
import warnings

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.finance import quotes_historical_yahoo_ochl\
        as quotes_yahoo
from hmmlearn.hmm import GaussianHMM
```

Загрузим исторические биржевые котировки, относящиеся к периоду времени с 4 сентября 1970 по 17 мая 2016 года. Вместо него вы можете задать любой желаемый период времени.

```
# Загрузка исторических биржевых котировок из пакета matplotlib
start = datetime.date(1970, 9, 4)
end = datetime.date(2016, 5, 17)
stock_quotes = quotes_yahoo('INTC', start, end)
```

Извлечем данные о ежедневных котировках закрытия и объемах акций, проторгованных в этот день.

```
# Извлечение ежедневных котировок на момент закрытия биржи
closing_quotes = np.array([quote[2] for quote in stock_quotes])

# Извлечение ежедневных объемов проторгованных акций
volumes = np.array([quote[5] for quote in stock_quotes])[1:]
```

Определим процентные разницы ежедневных котировок закрытия.

```
# Определение процентной разницы котировок
# на момент закрытия биржи
diff_percentages = 100.0 * np.diff(closing_quotes)
                   closing_quotes[:-1]
```

Поскольку взятие разностей уменьшает длину массива на 1, необходимо соответствующим образом скорректировать массив данных.

```
# Взятие списка дат, начиная со второго значения
dates = np.array([quote[0] for quote in stock_quotes],
                 dtype=np.int)[1:]
```

Упакуем попарно элементы двух столбцов данных для создания тренировочного набора.

```
# Попарная упаковка разностей и объемов акций для тренировки
training_data = np.column_stack([diff_percentages, volumes])
```

Создадим и обучим гауссовскую HMM-модель с 7 компонентами и диагональной ковариацией.

```
# Создадим и обучим гауссовскую модель HMM
hmm = GaussianHMM(n_components=7, covariance_type='diag',
                  n_iter=1000)
with warnings.catch_warnings():
    warnings.simplefilter('ignore')
    hmm.fit(training_data)
```

Используем обученную HMM-модель для генерирования 300 выборочных данных. Вместо этого вы можете выбрать любое другое желаемое число.

```
# Генерирование данных с использованием HMM-модели
num_samples = 300
samples, _ = hmm.sample(num_samples)
```

Построим график сгенерированных значений для процентных разниц.

```
# Построение графика процентных разниц
plt.figure()
plt.title('Процентная разность')
plt.plot(np.arange(num_samples), samples[:, 0], c='black')
```

Построим график сгенерированных значений для объемов проторгованных акций.

```
# Построение графика объемов проторгованных акций
plt.figure()
plt.title('Объем проторгованных акций')
plt.plot(np.arange(num_samples), samples[:, 1], c='black')
plt.ylim(ymin=0)
plt.show()
```

Полный код примера содержится в файле `stock_market.py`. В процессе выполнения этого кода на экране отобразятся два графика. На первом снимке экрана отображены процентные разницы, сгенерированные HMM (рис. 11.16).

На втором снимке экрана отображены значения, сгенерированные HMM для ежедневных объемов проторгованных акций (рис. 11.17).

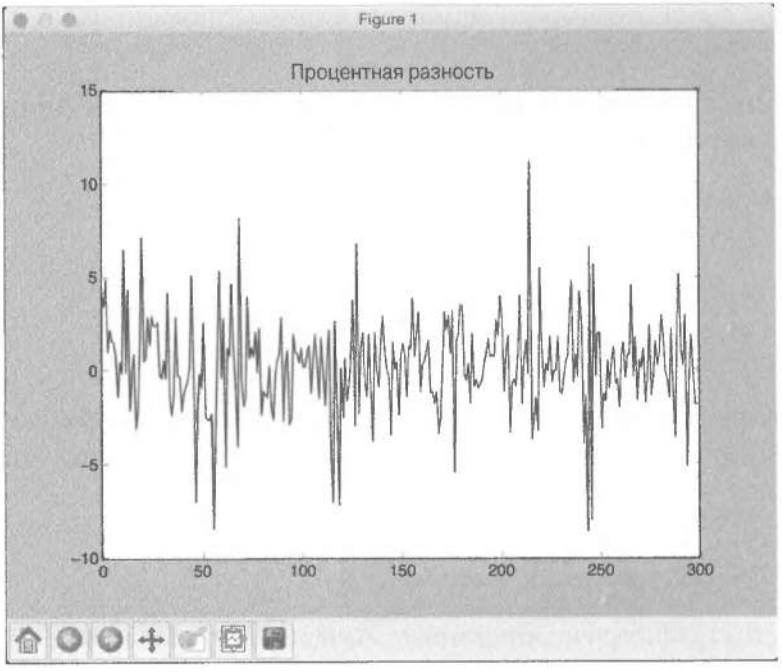

Рис. 11.16

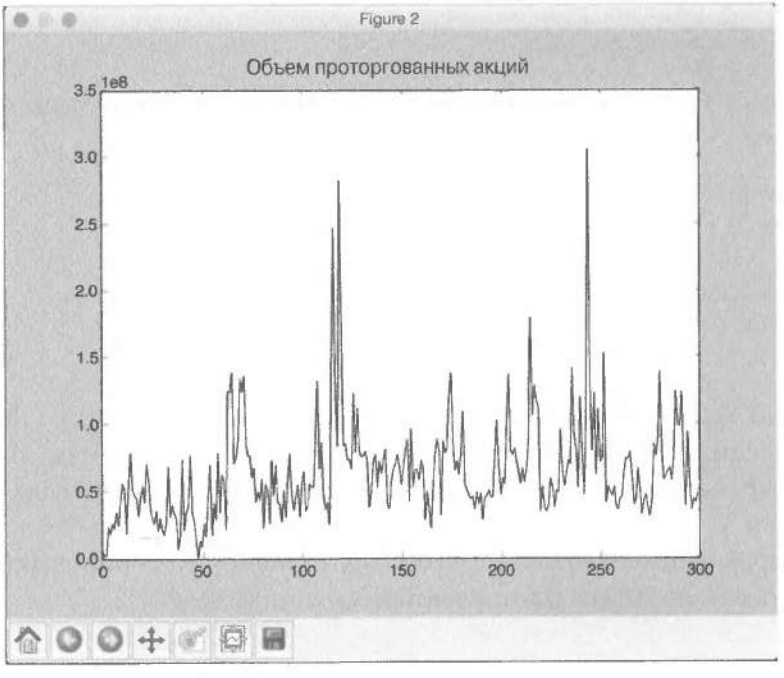

Рис. 11.17

Резюме

В этой главе вы научились создавать последовательные модели обучения. Мы обсудили обработку временных рядов данных средствами библиотеки Pandas. Было показано, как извлекать срезы временных рядов и выполнять операции над ними. Вы научились извлекать различные статистики из временных рядов с помощью скользящего окна. Мы обсудили скрытые марковские модели, а затем реализовали систему для построения такой модели.

Мы также обсудили использование условных случайных полей для анализа буквенных последовательностей. Вы узнали о том, как анализировать данные биржевых котировок с помощью различных методов. В следующей главе рассказывается о распознавании речи и создании системы, предназначенной для автоматического распознавания произносимых слов.

12

Создание систем распознавания речи

В этой главе мы поговорим о распознавании речи. Мы обсудим приемы работы с голосовыми сигналами и расскажем о том, как визуализировать различные аудиосигналы. После изучения методов обработки звуковых сигналов вы узнаете о том, как создать систему распознавания.

К концу главы вы освоите следующие темы:

- работа со звуковыми сигналами;
- визуализация аудиосигналов;
- преобразование аудиосигналов в частотные интервалы;
- генерирование аудиосигналов;
- синтезирование тонов;
- извлечение речевых признаков;
- распознавание голосовых команд.

Работа со звуковыми сигналами

Распознавание речи — это процесс распознавания слов, произносимых человеком. Голосовые сигналы улавливаются микрофоном, и система пытается распознать уловленные слова. Распознавание речи интенсивно применяется при взаимодействии человека с компьютерами и смартфонами, а также в транскрипторах речи, биометрических системах, системах безопасности и т.п.

Прежде чем приступить к анализу голосовых сигналов, очень важно понять их природу. Эти сигналы представляют собой смесь различных сигналов. Существует множество аспектов речи, вносящих свой вклад в эту сложность. Сюда входят эмоции, акцент, язык общения, посторонние шумы и многое другое.

Отсюда следует, что формулировка надежного набора правил для анализа голосовых сигналов — задача непростая. Однако люди успешно справляются с распознаванием речи, несмотря на многочисленные осложняющие факторы. Похоже, это дается им относительно легко. Если мы хотим, чтобы наши машины делали то же самое, мы должны помочь им понимать речь так же, как понимаем ее мы.

Исследователи работают над различными аспектами и приложениями распознавания речи, такими как распознавание произносимых слов, идентификация личности говорящего, распознавание эмоций, идентификация акцента и т.п. В этой главе мы сфокусируем внимание на понимании произносимых слов. Распознавание речи представляет важный шаг в отношении взаимодействия человека с компьютером. Если мы хотим создавать когнитивных роботов, способных взаимодействовать с человеком, то они должны разговаривать с нами на естественном языке. Именно по этой причине автоматическое распознавание речи остается в центре внимания многих исследователей на протяжении последних лет. Перейдем к обсуждению способов обработки речевых сигналов и созданию системы распознавания речи.

Визуализация аудиосигналов

Приступим к рассмотрению способов визуализации аудиосигналов. Вы узнаете о том, как читать аудиосигналы из файла и работать с ними. Это поможет вам понять, как структурируется аудиосигнал. Если аудиосигналы записываются с микрофона, то при этом семплируются фактические аудиосигналы и сохраняются их оцифрованные версии. Реальные сигналы представляют собой непрерывные волны, откуда следует, что мы не можем их сохранить в том виде, как они есть. Мы должны семплировать сигнал с определенной частотой и преобразовать его в дискретное цифровое представление.

Чаще всего речевые сигналы дискретизируют с частотой 44100 Гц. Это означает, что в течение одной секунды речевой сигнал разбивается на 44100 частей и значения, соответствующие различным отметкам времени, сохраняются в выходном файле. Мы сохраняем значение аудиосигнала каждую 1/44100 долю секунды. В таком случае мы говорим, что частота дискретизации аудиосигнала составляет 44100 Гц. В случае выбора достаточно высокой частоты дискретизации аудиосигнала у людей, слышащих его, создается впечатление, будто он непрерывный. Пойдем дальше и визуализируем аудиосигнал.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

Прочитаем входной аудиофайл, используя метод `file.read`. Этот метод возвращает два значения: частоту дискретизации и аудиосигнал.

```
# Чтение аудиофайла
sampling_freq, signal = wavfile.read('random_sound.wav')
```

Выведем форму сигнала, тип данных и длительность аудиосигнала.

```
# Отображение параметров
print('\nSignal shape:', signal.shape)
print('Datatype:', signal.dtype)
print('Signal duration:', round(signal.shape[0]
      float(sampling_freq), 2), 'seconds')
```

Нормализуем сигнал.

```
# Нормализация сигнала
signal = signal / np.power(2, 15)
```

Извлечем первые 50 значений из массива `numpy` для построения графика.

```
# Извлечение первых 50 значений
signal = signal[:50]
```

Построим ось времени в миллисекундах для графика.

```
# Построение оси времени в миллисекундах
time_axis = 1000 * np.arange(0, len(signal), 1) / float(sampling_freq)
```

Построим график аудиосигнала.

```
# Построение графика аудиосигнала
plt.plot(time_axis, signal, color='black')
plt.xlabel('Время (миллисекунды)')
plt.ylabel(Амплитуда')
plt.title('Входной аудиосигнал')
plt.show()
```

Полный код примера содержится в файле `audio_plotter.py`. В процессе выполнения этого кода на экране отобразится следующий график (рис. 12.1).

На этом снимке экрана отображены первые 50 семплов входного аудиосигнала. В окне терминала отобразится следующая информация (рис. 12.2).

Вывод, представленный на этом рисунке, отображает информацию, извлеченную из сигнала.

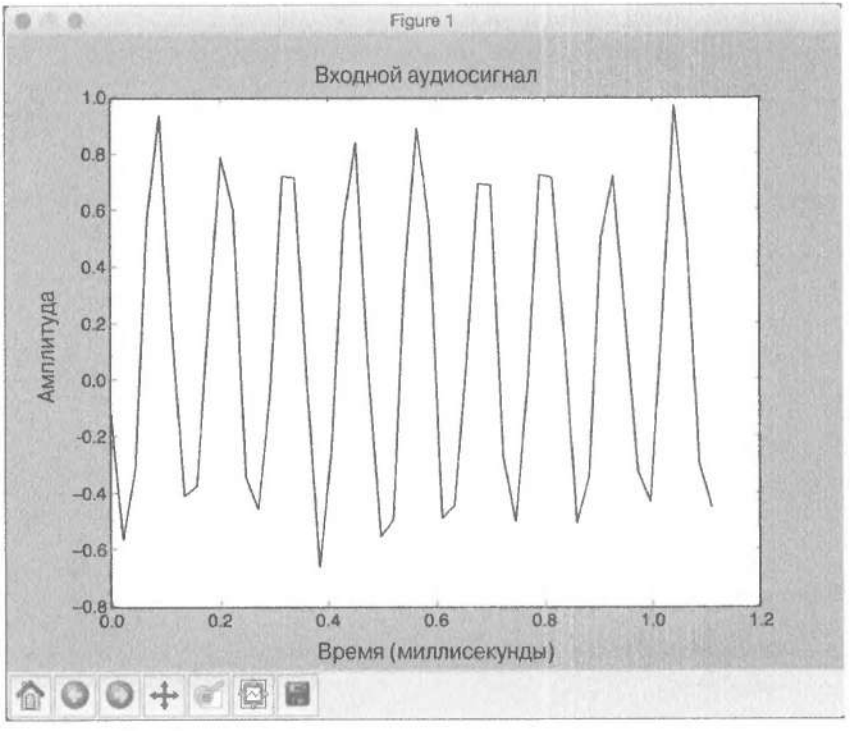

Рис. 12.1

```
Signal shape: (132300,)
Datatype: int16
Signal duration: 3.0 seconds
```

Рис. 12.2

Преобразование аудиосигналов в частотные интервалы

Прежде чем приступить к анализу аудиосигналов, следует сказать несколько слов об их базовых частотных компонентах. Это даст понимание того, как можно извлечь смысловую информацию из такого сигнала. Аудиосигналы состоят из смеси синусоидальных волн с различными частотами, фазами и амплитудами.

Анализируя частотные компоненты, можно идентифицировать множество характеристик. Любой аудиосигнал характеризуется своим распределением по частотному спектру. Чтобы преобразовать сигнал из дискретного времен-

ного представления в частотную область, мы должны использовать математический аппарат преобразований Фурье. Вы сможете быстро освежить свои знания в этой области, посетив сайт `http://www.thefouriertransform.com`. Перейдем к конкретному примеру преобразования аудиосигнала из дискретного в частотное представление.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

Прочитаем входной аудиофайл, используя метод `wavefile.read`. Этот метод возвращает два значения: частоту дискретизации и аудиосигнал.

```
# Чтение аудиофайла
sampling_freq, signal = wavfile.read('spoken_word.wav')
```

Нормализуем аудиосигнал.

```
# Нормализация значений
signal = signal / np.power(2, 15)
```

Извлечем длину и половинную длину сигнала.

```
# Извлечение длины сигнала
len_signal = len(signal)
# Извлечение половинной длины
len_half = np.ceil((len_signal + 1) / 2.0).astype(np.int)
```

Применим к сигналу преобразование Фурье.

```
# Применение преобразования Фурье
freq_signal = np.fft.fft(signal)
```

Нормализуем частотный сигнал и возведем его в квадрат.

```
# Нормализация
freq_signal = abs(freq_signal[0:len_half]) / len_signal

# Возведение в квадрат
freq_signal **= 2
```

Откорректируем преобразованный сигнал для четных и нечетных случаев.

```
# Извлечение длины преобразованного частотного сигнала
len_fts = len(freq_signal)

# Корректировка сигнала для четных и нечетных случаев
if len_signal % 2:
    freq_signal[1:len_fts] *= 2
else:
    freq_signal[1:len_fts-1] *= 2
```

Извлечем мощность сигнала в децибелах.

```
# Извлечение значения мощности в децибелах
signal_power = 10 * np.log10(freq_signal)
```

Построим ось X, вдоль которой в данном случае будем откладывать частоту в килогерцах.

```
# Построение оси X
x_axis = np.arange(0, len_half, 1) * (sampling_freq / len_signal) /
         1000.0
```

Построим график.

```
# Построение графика
plt.figure()
plt.plot(x_axis, signal_power, color='black')
plt.xlabel('Частота (кГц)')
plt.ylabel('Мощность сигнала (дБ)')
plt.show()
```

Полный код примера содержится в файле `frequency_transformer.py`. После выполнения этого кода на экране отобразится следующий график (рис. 12.3).

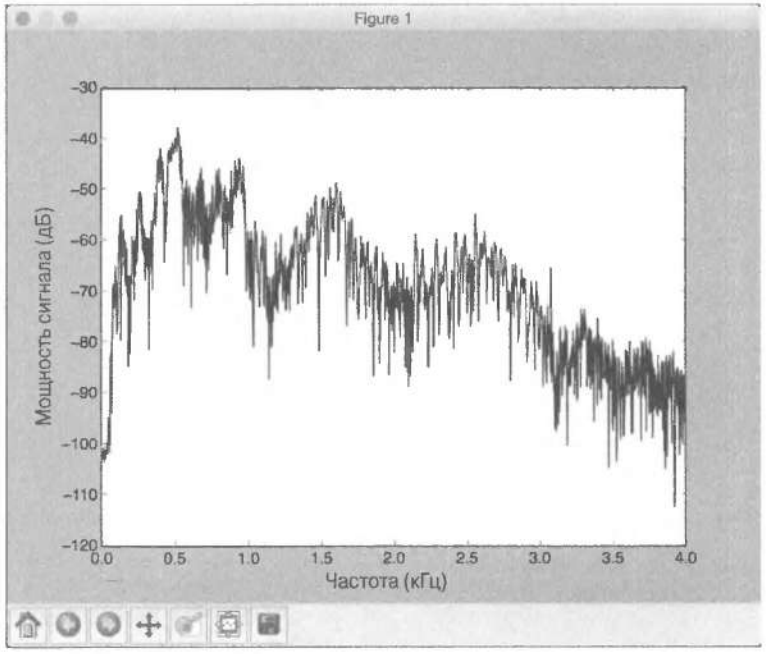

Рис. 12.3

На этом снимке экрана представлено изменение мощности сигнала вдоль частотного спектра.

Генерирование аудиосигналов

Теперь, когда вы узнали о том, как работают сигналы, мы можем сгенерировать один такой сигнал. Для генерации различных аудиосигналов можно воспользоваться пакетом NumPy. Поскольку аудиосигналы представляют собой смеси синусоид, мы можем использовать этот факт для того, чтобы сгенерировать аудиосигнал с заданными параметрами.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write
```

Определим имя выходного аудиофайла.

```
# Выходной файл для сохранения аудиосигнала
output_file = 'generated_audio.wav'
```

Укажем аудиопараметры, такие как длительность, частота дискретизации, частота звука, минимальное и максимальное значения.

```
# Задание аудиопараметров
duration = 4 # в секундах
sampling_freq = 44100 # в кГц
tone_freq = 784
min_val = -4 * np.pi
max_val = 4 * np.pi
```

Сгенерируем аудиосигнал, используя установленные параметры.

```
# Генерация аудиосигнала
t = np.linspace(min_val, max_val, duration * sampling_freq)
signal = np.sin(2 * np.pi * tone_freq * t)
```

Добавим шум в сигнал.

```
# Добавление шума в сигнал
noise = 0.5 * np.random.rand(duration * sampling_freq)
signal += noise
```

Нормализация и масштабирование сигнала.

```
# Масштабирование до 16-битовых целых значений
scaling_factor = np.power(2, 15) - 1
signal_normalized = signal / np.max(np.abs(signal))
signal_scaled = np.int16(signal_normalized * scaling_factor)
```

Сохраним сгенерированный аудиосигнал в выходном файле.

```
# Сохранение аудиосигнала в выходном файле
write(output_file, sampling_freq, signal_scaled)
```

Извлечем первые 200 значений для построения графика.

```
# Извлечение первых 200 значений из аудиосигнала
signal = signal[:200]
```

Построим временную ось в миллисекундах.

```
# Построение временной оси в миллисекундах
time_axis = 1000 * np.arange(0, len(signal), 1) / float(sampling_freq)
```

Построим график аудиосигнала.

```
# Построение графика аудиосигнала
plt.plot(time_axis, signal, color='black')
plt.xlabel('Время (миллисекунды)')
plt.ylabel('Амплитула')
plt.title('Сгенерированный аудиосигнал')
plt.show()
```

Полный код примера содержится в файле `audio_generator.py`. После выполнения этого кода на экране отобразится следующий график (рис. 12.4).

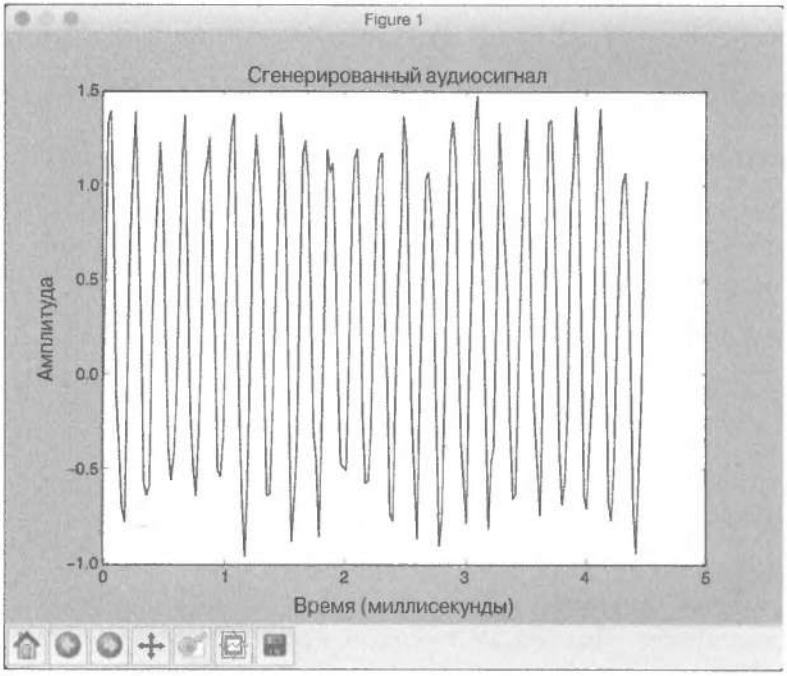

Рис. 12.4

Воспроизведите файл `generated_audio.wav` в своем медиапроигрывателе и послушайте, как он звучит. Это будет сигнал, представляющий собой смесь звука частотой 784 Гц и шума.

Синтезирование звуков для генерации музыки

В предыдущем разделе была описана генерация простого монотонного звука, но такой звук не очень интересен. На протяжении всего сигнала звучала только одна частота. Воспользуемся тем же принципом для синтезирования музыки посредством склеивания различных звуков. Для генерации музыки мы используем стандартные звуки, такие как `A` (ля), `C` (до), `G` (соль), `F` (фа) и т.д. Таблицу соответствий между частотой и стандартными звуками можно найти по адресу `http://www.phy.mtu.edu/~suits/notefreqs.html`. Используем эту информацию для того, чтобы сгенерировать музыкальный сигнал.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import json

import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write
```

Определим функцию, генерирующую звук на основании входных параметров.

```
# Синтез звука на основании входных параметров
def tone_synthesizer(freq, duration, amplitude=1.0,
                     sampling_freq=44100):
    # Создание временной оси
    time_axis = np.linspace(0, duration, duration *
                            sampling_freq)
```

Создадим аудиосигнал, используя указанные параметры, и вернем его.

```
    # Создание аудиосигнала
    signal = amplitude * np.sin(2 * np.pi * freq * time_axis)

    return signal.astype(np.int16)
```

Определим основную функцию, начав с определения имен выходных аудиофайлов.

```
if __name__=='__main__':
    # Имена выходных файлов
    file_tone_single = 'generated_tone_single.wav'
    file_tone_sequence = 'generated_tone_sequence.wav'
```

Мы будем использовать уже упоминавшийся файл, содержащий таблицу соответствий названий звуков (таких, как A, C, G и т.д.) и соответствующих частот.

```
# Источник: http://www.phy.mtu.edu/~suits/notefreqs.html
mapping_file = 'tone_mapping.json'
# Загрузка таблицы соответствия звуков частотам из файла
with open(mapping_file, 'r') as f:
    tone_map = json.loads(f.read())
```

Сгенерируем звук F (фа) длительностью 3 секунды.

```
# Зададим входные параметры для генерации звука 'F'
tone_name = 'F'
duration = 3 # секунды
amplitude = 12000
sampling_freq = 44100 # Гц
```

Извлечем соответствующую частоту звука.

```
# Извлечение частоты звука
tone_freq = tone_map[tone_name]
```

Сгенерируем звук, используя функцию синтезатора звука, определенную ранее.

```
# Генерация звука с использованием заданных выше параметров
synthesized_tone = tone_synthesizer(tone_freq, duration,
                                    amplitude, sampling_freq)
```

Запишем сгенерированный аудиосигнал в выходной файл.

```
# Запись аудиосигнала в выходной файл
write(file_tone_single, sampling_freq, synthesized_tone)
```

Сгенерируем последовательность звуков, которая будет звучать как музыка. Определим звуковую последовательность, одновременно задав соответствующие длительности в секундах.

```
# Определение последовательности звуков вместе
# с соответствующими длительностями в секундах
tone_sequence = [('G', 0.4), ('D', 0.5), ('F', 0.3), ('C', 0.6),
                 ('A', 0.4)]
```

Сконструируем аудиосигнал на базе последовательности звуков.

```
# Конструирование аудиосигнала на базе
# определенной выше последовательности
signal = np.array([])
for item in tone_sequence:
```

```
# Получение имени данного звука
tone_name = item[0]
```

Извлечем соответствующую частоту для каждого звука.

```
# Извлечение частоты данного звука
freq = tone_map[tone_name]
```

Извлечем соответствующую длительность:

```
# Извлечение длительности
duration = item[1]
```

Синтезируем звук с помощью функции-синтезатора.

```
# Синтез звука
synthesized_tone = tone_synthesizer(freq, duration,
                   amplitude, sampling_freq)
```

Присоединим его к основному выходному сигналу.

```
# Присоединение к выходному сигналу
signal = np.append(signal, synthesized_tone, axis=0)
```

Сохраним основной выходной сигнал в выходном файле.

```
# Сохранение звука в выходном файле
write(file_tone_sequence, sampling_freq, signal)
```

Полный код примера содержится в файле `synthesizer.py`. После выполнения этого кода будут сгенерированы два выходных файла: `generated_tone_single.wav` и `generated_tone_sequence.wav`. Воспроизведите их в своем медиапроигрывателе и послушайте, как они звучат.

Извлечение речевых признаков

Теперь вы знаете, как преобразовать дискретизированный по времени сигнал в частотные интервалы. Признаки частотных интервалов интенсивно применяются во всех системах распознавания речи. Понятия, которые мы до этого обсуждали, всего лишь дают представление об общей идее, но реальные свойства частотных интервалов намного сложнее. Как только сигнал преобразован в частотное представление, мы должны убедиться в том, что его можно использовать в качестве вектора признаков. И здесь на помощь приходят **коэффициенты MFCC** (Mel Frequency Cepstral Coefficients). MFCC — это инструмент для извлечения частотных характеристик из заданного аудиосигнала.

В процессе извлечения частотных признаков аудиосигнала система MFCC прежде всего извлекает спектр мощности. Затем она извлекает признаки, используя гребенки фильтров и *дискретное косинусное преобразование* (Discrete Cosine Transform — DCT). Для дальнейшего ознакомления с MFCC посетите следующую веб-страницу:

```
http://practicalcryptography.com/miscellaneous/machine-learning/
guide-mel-frequency-cepstral-coefficients-mfccs
```

Для извлечения MFCC-признаков мы будем использовать пакет `python_speech_features`, который доступен по адресу `http://python-speech-features.readthedocs.org/en/latest`. В целях упрощения в состав файлов примеров включена папка `features`, содержащая файлы, необходимые для работы с этим пакетом. Рассмотрим конкретный пример извлечения MFCC-признаков.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from features import mfcc, logfbank
```

Прочитаем входной аудиофайл и извлечем первые 10000 семплов для анализа.

```
# Чтение входного аудиофайла
sampling_freq, signal = wavfile.read('random_sound.wav')

# Взятие первых 10000 семплов для анализа
signal = signal[:10000]
```

Извлечем MFCC-признаки.

```
# Извлечение MFCC-признаков
features_mfcc = mfcc(signal, sampling_freq)
```

Выведем параметры MFCC.

```
# Вывод параметров для MFCC
print('\nMFCC:\nNumber of windows =', features_mfcc.shape[0])
print('Length of each feature =', features_mfcc.shape[1])
```

Построим график MFCC-признаков.

```
# Построение графика признаков
features_mfcc = features_mfcc.T
plt.matshow(features_mfcc)
plt.title('MFCC')
```

Извлечем свойства гребенки фильтров.

```
# Извлечем свойства гребенки фильтров
features_fb = logfbank(signal, sampling_freq)
```

Выведем параметры для гребенки фильтров.

```
# Вывод параметров для гребенки фильтров
print('\nFilter bank:\nNumber of windows =',
      features_fb.shape[0])
print('Length of each feature =', features_fb.shape[1])
```

Построим график признаков.

```
# Построение графика признаков
features_fb = features_fb.T
plt.matshow(features_fb)
plt.title('Гребенка фильтров')
plt.show()
```

Полный код примера содержится в файле `feature_extractor.py`. В процессе выполнения этого кода на экране отобразятся два графика. На первом снимке экрана отображены MFCC-признаки (рис. 12.5).

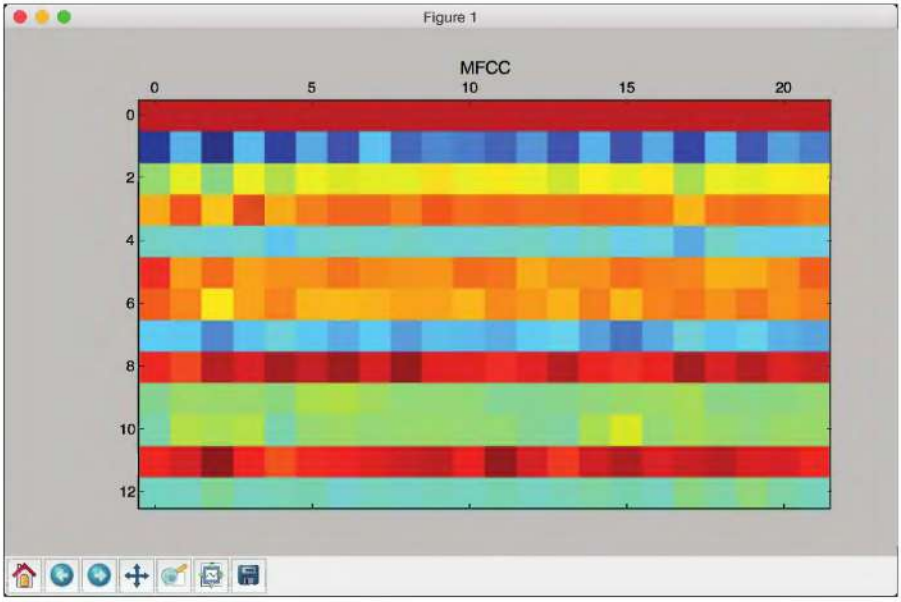

Рис. 12.5. (См. цветную вклейку; адрес указан во введении)

На втором снимке экрана отображены признаки гребенки фильтров (рис. 12.6).

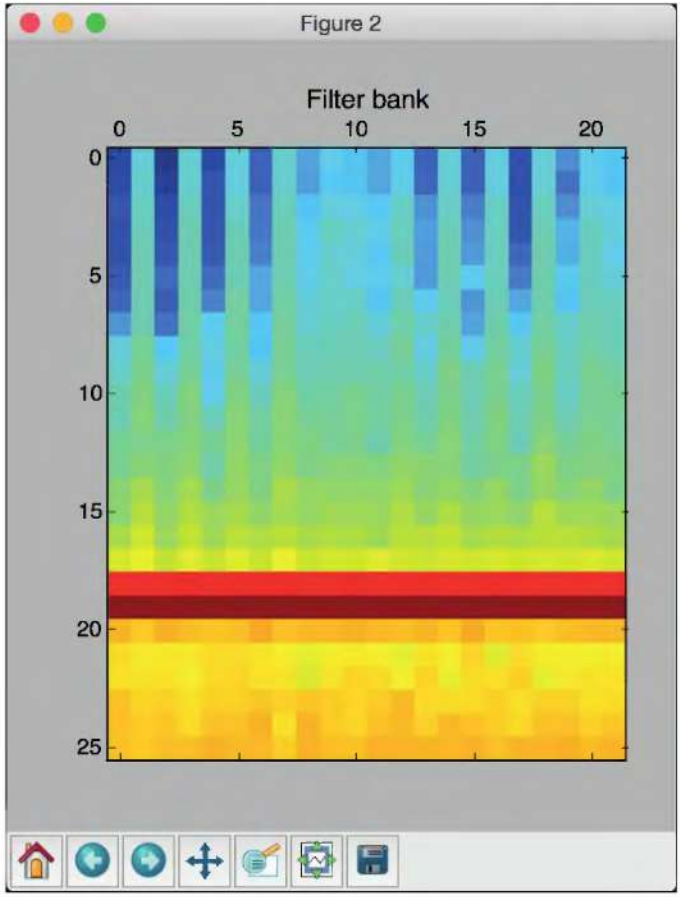

Рис. 12.6. (См. цветную вклейку; адрес указан во введении)

В окне терминала отобразится следующая информация (рис. 12.7).

```
MFCC:
Number of windows = 22
Length of each feature = 13

Filter bank:
Number of windows = 22
Length of each feature = 26
```

Рис. 12.7

Распознавание произносимых слов

После того как вы ознакомились с принципами анализа звуковых сигналов, мы можем перейти к рассмотрению методов распознавания произносимых слов. Системы распознавания речи получают аудиосигналы на входе и распознают произносимые слова. Для этой задачи мы используем скрытые марковские модели (Hidden Markov Models — HMM).

Как уже обсуждалось в предыдущей главе, HMM отлично справляются с анализом последовательных данных. Аудиосигнал — это временной ряд данных, являющийся примером последовательных данных. Исходное предположение заключается в том, что выходные результаты генерируются системой, проходящей через серию скрытых состояний. Нашей задачей является выяснение того, что собой представляют эти скрытые состояния, чтобы в нашем сигнале можно было идентифицировать слова. Если захотите окунуться глубже в эту тему, обратитесь по адресу `https://www.robots.ox.ac.uk/~vgg/rg/slides/hmm.pdf`.

Для построения нашей системы распознавания речи мы используем пакет `hmmlearn`. Подробнее об этом пакете можно узнать по адресу `http://hmmlearn.readthedocs.org/en/latest`. Чтобы установить его, выполните следующую команду в окне своего терминала:

```
$ pip3 install hmmlearn
```

Для тренировки нашей системы распознавания речи нам нужен набор данных с аудиофайлами для каждого слова. Мы будем использовать базу данных, доступную по адресу `https://code.google.com/archive/p/hmm-speech-recognition/downloads`. Чтобы упростить ее использование, в состав файлов примеров включена папка `data`, содержащая все необходимые файлы. Указанный набор данных содержит семь различных слов. С каждым словом ассоциирована папка, и каждая папка содержит 15 аудиофайлов. Мы используем 14 папок для тренировки и одну для тестирования. Обратите внимание на то, что этот набор данных в действительности очень небольшой. На практике вы будете использовать для создания систем распознавания речи наборы данных гораздо большего размера. Мы используем этот набор данных лишь для ознакомления с методами распознавания речи и демонстрации примера создания системы для распознавания произносимых слов.

Для каждого слова мы создадим HMM-модель. Все эти модели будут сохранены, чтобы впоследствии на них можно было ссылаться. Когда нам потребуется распознать слово в неизвестном аудиофайле, мы пропустим его через эти модели и выберем ту из них, которая дает наибольшую оценку. Покажем, как можно построить такую систему.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import os
import argparse
import warnings

import numpy as np
from scipy.io import wavfile

from hmmlearn import hmm
from features import mfcc
```

Определим функцию для набора входных аргументов. Мы должны указать входную папку, которая содержит аудиофайлы, необходимые для обучения нашей системы распознавания речи.

```
# Определение функции для анализа аргументов
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Trains \
            the HMM-based speech recognition system')
    parser.add_argument("--input-folder", dest="input_folder",
            required=True, help="Input folder containing the
            audio files for training")
    return parser
```

Определим класс для тренировки HMM.

```
# Определение класса для тренировки HMM
class ModelHMM(object):
    def __init__(self, num_components=4, num_iter=1000):
        self.n_components = num_components
        self.n_iter = num_iter
```

Определим тип ковариации и тип HMM.

```
        self.cov_type = 'diag'
        self.model_name = 'GaussianHMM'
```

Инициализируем переменную, в которой будем хранить модели для каждого слова.

```
        self.models = []
```

Определим модель, используя указанные параметры.

```
        self.model = hmm.GaussianHMM(n_components=
                self.n_components,
                covariance_type=self.cov_type,
                n_iter=self.n_iter)
```

Определим метод для обучения модели.

```
    # 'training_data' - это 2D-массив numpy, в котором
    # каждая строка имеет 13 измерений
    def train(self, training_data):
        np.seterr(all='ignore')
        cur_model = self.model.fit(training_data)
        self.models.append(cur_model)
```

Определим метод, оценивающий входные данные.

```
    # Выполнение HMM-модели для оценки входных данных
    def compute_score(self, input_data):
        return self.model.score(input_data)
```

Определим функцию, обеспечивающую построение модели для каждого слова в тренировочном наборе данных.

```
# Определение функции, создающей модель для каждого слова
def build_models(input_folder):
    # Инициализация переменной для сохранения всех моделей
    speech_models = []
```

Проанализируем входной каталог.

```
    # Анализ входного каталога
    for dirname in os.listdir(input_folder):
        # Получение имени подпапки
        subfolder = os.path.join(input_folder, dirname)

        if not os.path.isdir(subfolder):
            continue
```

Извлечем метку.

```
        # Извлечение метки
        label = subfolder[subfolder.rfind('/') + 1:]
```

Инициализируем переменную, предназначенную для хранения тренировочных данных.

```
        # Инициализация переменной
        X = np.array([])
```

Создадим список файлов, которые будут использоваться для тренировки моделей.

```
    # Создание списка файлов, которые будут использоваться
    # для тренировки моделей. Один из файлов в каждой папке
    # мы оставляем для тестирования
    training_files = [x for x in os.listdir(subfolder) if
                      x.endswith('.wav')][:-1]

    # Итерируем по тренировочным файлам и строим модели
    for filename in training_files:
        # Извлечение пути к текущему файлу
        filepath = os.path.join(subfolder, filename)
```

Прочитаем аудиосигнал из текущего файла.

```
        # Чтение аудиосигнала из текущего файла
        sampling_freq, signal = wavfile.read(filepath)
```

Извлечем MFCC-признаки.

```
        # Извлечение MFCC-признаков
        with warnings.catch_warnings():
            warnings.simplefilter('ignore')
            features_mfcc = mfcc(signal, sampling_freq)
```

Присоединим точку данных к переменной X.

```
        # Присоединение к переменной X
        if len(X) == 0:
            X = features_mfcc
        else:
            X = np.append(X, features_mfcc, axis=0)
```

Инициализируем HMM-модель.

```
    # Создание HMM-модели
    model = ModelHMM()
```

Обучим модель, используя тренировочные данные.

```
    # Обучение HMM
    model.train(X)
```

Сохраним модель для текущего слова.

```
    # Сохранение модели для текущего слова
    speech_models.append((model, label))
```

```
        # Сброс переменной
        model = None

    return speech_models
```

Определим функцию для тестирования тренировочного набора данных.

```
# Определение функции, тестирующей входные файлы
def run_tests(test_files):
    # Классификация входных данных
    for test_file in test_files:
        # Чтение входного файла
        sampling_freq, signal = wavfile.read(test_file)
```

Извлечем MFCC-признаки.

```
        # Извлечение MFCC-признаков
        with warnings.catch_warnings():
            warnings.simplefilter('ignore')
            features_mfcc = mfcc(signal, sampling_freq)
```

Определим переменные для хранения максимальной оценки и выходной метки.

```
        # Определение переменных
        max_score = -float('inf')
        output_label = None
```

Выполним итерации по моделям, чтобы выбрать из них наилучшую.

```
        # Прогон текущего вектора признаков через каждую
        # HMM-модель и выбор той из них, которая получила
        # наивысшую оценку
        for item in speech_models:
            model, label = item
```

Вычислим оценку и сравним ее с максимальной оценкой.

```
            score = model.compute_score(features_mfcc)
            if score > max_score:
                max_score = score
                predicted_label = label
```

Вывод результата.

```
        # Вывод предсказанного результата
        start_index = test_file.find('/') + 1
```

```
        end_index = test_file.rfind('/')
        original_label = test_file[start_index:end_index]
        print('\nOriginal: ', original_label)
        print('Predicted:', predicted_label)
```

Определим основную функцию и получим входную папку из входного параметра.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    input_folder = args.input_folder
```

Создадим HMM-модель для каждого слова из входной папки.

```
    # Создание HMM-модели для каждого слова
    speech_models = build_models(input_folder)
```

Один из файлов в каждой папке мы оставляем для тестирования. Используем этот файл для того, чтобы выяснить, насколько точна данная модель.

```
    # Тестовые файлы -- 15-й файл в каждой папке
    test_files = []
    for root, dirs, files in os.walk(input_folder):
        for filename in (x for x in files if '15' in x):
            filepath = os.path.join(root, filename)
            test_files.append(filepath)

    run_tests(test_files)
```

Полный код примера содержится в файле `speech_recognizer.py`. Убедитесь в том, что папка `data` находится в той же папке, что и файл с кодом. Выполните код с помощью приведенной ниже команды:

```
$ python3 speech_recognizer.py --input-folder data
```

После выполнения кода в окне терминала отобразится следующая информация (рис. 12.8).

Как видите, нашей системе распознавания речи удалось правильно идентифицировать все слова.

```
Original:  apple
Predicted: apple

Original:  banana
Predicted: banana

Original:  kiwi
Predicted: kiwi

Original:  lime
Predicted: lime

Original:  orange
Predicted: orange

Original:  peach
Predicted: peach

Original:  pineapple
Predicted: pineapple
```

Рис. 12.8

Резюме

Эта глава была посвящена распознаванию речи. Мы обсудили работу с голосовыми сигналами и связанные с этим понятия. Вы узнали о том, как визуализировать аудиосигналы и преобразовать их из дискретного по времени представления в частотное представление с помощью преобразования Фурье. Мы также обсудили генерацию аудиосигналов с использованием заданных параметров.

После этого мы применили изученные концепции для синтеза музыки путем склеивания звуков. Мы рассмотрели MFCC-признаки и способы их применения в реальных задачах. Кроме того, было показано, как извлекать частотные признаки из речи, и был приведен пример создания системы распознавания речи с использованием изученных средств. В следующей главе вы ознакомитесь с методами обнаружения и отслеживания объектов. Изложенные концепции будут использованы для создания системы, способной отслеживать объекты в живом видеопотоке.

13
Обнаружение и отслеживание объектов

В этой главе мы будем изучать методы обнаружения и отслеживания объектов. Мы начнем с установки OpenCV — популярной библиотеки инструментов для компьютерного зрения. Далее мы обсудим алгоритм вычисления разности между кадрами, применяемый для обнаружения движущихся элементов видео. Вы узнаете об использовании цветовых пространств для отслеживания объектов и научитесь отслеживать объекты путем вычитания фоновых изображений. Мы создадим интерактивный трекер для отслеживания объектов с помощью алгоритма CAMShift. Вы также узнаете о том, как создать трекер, отслеживающий объекты на основе анализа оптического потока. Мы обсудим проблему распознавания лиц и связанные с этим понятия, такие как каскады Хаара и интегральные изображения. Наконец, мы используем описанную методику для отслеживания глаз и определение координат взора.

К концу главы вы освоите следующие темы:

- установка библиотеки OpenCV;
- вычисление разности между кадрами;
- отслеживание объектов с помощью цветовых пространств;
- отслеживание объектов путем вычитания фоновых изображений;
- создание интерактивного трекера для отслеживания объектов с помощью алгоритма CAMShift;
- отслеживание объектов с использованием оптических потоков;
- обнаружение и отслеживание лиц;
- использование каскадов Хаара для обнаружения объектов;
- использование интегральных изображений для извлечения признаков;
- отслеживание глаз и определение координат взора.

Установка библиотеки OpenCV

В этой главе мы будем использовать библиоетку OpenCV. Более подробную информацию о ней можно получить на сайте `http://opencv.org`. Прежде чем продолжить чтение, убедитесь в том, вы установили эту библиотеку. Ниже приведены ссылки, которыми вы сможете воспользоваться для установки OpenCV 3 в Python 3 в различных операционных системах.

- Windows:

  `https://solarianprogrammer.com/2016/09/17/install-opencv-3-with-python-3-on-windows`

- Ubuntu:

  `http://www.pyimagesearch.com/2015/07/20/install-opencv-3-0-and-python-3-4-on-ubuntu`

- Mac:

  `http://www.pyimagesearch.com/2015/06/29/install-opencv-3-0-and-python-3-4-on-osx`

Вычисление разности между кадрами

Вычисление разности между кадрами — простейшая методика, которую можно использовать для идентификации движущихся элементов видео. В процессе просмотра видеопотока мы получаем массу информации на основе разницы изображений между последовательными кадрами. Рассмотрим пример того, как можно вычислить и отобразить разности между последовательными кадрами. Для работы описанного ниже кода требуется подключенная камера, поэтому предварительно убедитесь в наличии веб-камеры на вашем компьютере.

Создайте новый файл Python и импортируйте следующий пакет:

```
import cv2
```

Определим функцию, вычисляющую разность между кадрами. Начнем с вычисления разности между текущим и следующим кадрами.

```
# Вычисление разности между кадрами
def frame_diff(prev_frame, cur_frame, next_frame):
    # Разность между текущим и следующим кадрами
    diff_frames_1 = cv2.absdiff(next_frame, cur_frame)
```

Вычислим разность между текущим и предыдущим кадрами.

```
# Разность между текущим и предыдущим кадрами
diff_frames_2 = cv2.absdiff(cur_frame, prev_frame)
```

Выполним операцию побитового И для двух разностей кадров и вернем результат.

```
return cv2.bitwise_and(diff_frames_1, diff_frames_2)
```

Определим функцию, захватывающую текущий кадр из веб-камеры. Начнем с чтения кадра из объекта захвата видео.

```
# Определение функции, получающей текущий кадр из веб-камеры
def get_frame(cap, scaling_factor):
    # Чтение текущего кадра из объекта захвата видео
    _, frame = cap.read()
```

Изменим размер кадра, используя масштабный множитель, и вернем его.

```
# Изменение размера изображения
frame = cv2.resize(frame, None, fx=scaling_factor,
        fy=scaling_factor, interpolation=cv2.INTER_AREA)
```

Преобразуем изображение в градации серого и вернем его.

```
# Преобразование в градации серого
gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

return gray
```

Определим основную функцию и инициализируем объект захвата видео.

```
if __name__=='__main__':
    # Определение объекта захвата видео
    cap = cv2.VideoCapture(0)
```

Определим масштабный множитель для изменения размера изображений.

```
# Определение масштабного множителя для изображений
scaling_factor = 0.5
```

Захватим текущий кадр, следующий кадр и еще один последующий кадр.

```
# Захват текущего кадра
prev_frame = get_frame(cap, scaling_factor)
# Захват следующего кадра
cur_frame = get_frame(cap, scaling_factor)
```

```
# Захват последующего кадра
next_frame = get_frame(cap, scaling_factor)
```

Выполним бесконечный цикл до тех пор, пока пользователь не нажмет клавишу <Esc>. Начнем с вычисления разностей между кадрами.

```
# Чтение кадров из веб-камеры до тех пор, пока
# пользователь не нажмет клавишу <Esc>
while True:
    # Отображение разности между кадрами
    cv2.imshow('Перемещение объекта', frame_diff(prev_frame,
            cur_frame, next_frame))
```

Обновим переменные кадров.

```
    # Обновление переменных
    prev_frame = cur_frame
    cur_frame = next_frame
```

Захватим следующий кадр из веб-камеры.

```
    # Захват следующего кадра
    next_frame = get_frame(cap, scaling_factor)
```

Проверим, не нажал ли пользователь клавишу <Esc>. Если это действительно так, выходим из цикла.

```
    # Проверка того, не нажал ли пользователь клавишу <Esc>
    key = cv2.waitKey(10)
    if key == 27:
        break
```

После выхода из цикла следует убедиться в том, что все окна закрыты надлежащим образом.

```
# Закрытие всех окон
cv2.destroyAllWindows()
```

Полный код примера содержится в файле `frame_diff.py`. Запустив программу, вы увидите окно, отображающее живое видео. Если вы немного подвигаетесь, то увидите свой силуэт, как в представленном на рис. 13.1 примере изображения.

Белые линии на этом рисунке представляют силуэт.

Рис. 13.1

Отслеживание объектов с помощью цветовых пространств

Информация, предоставляемая разностями между кадрами, полезна, но на ее основе нельзя построить надежный трекер. Она очень чувствительна к шуму и не позволяет по-настоящему отслеживать объекты. Для создания надежного инструмента отслеживания объектов мы должны знать, какие характеристики объекта могут обеспечить необходимую точность. И здесь на помощь приходят цветовые пространства.

Изображение может быть представлено с использованием различных цветовых пространств. Наиболее популярным из них является RGB, но для отслеживания объектов оно не совсем подходит. Вместо него мы будем использовать цветовое пространство HSV. Это интуитивная цветовая модель, которая точнее соответствует восприятию цвета человеком. Более подробную информацию об этом можно получить по адресу `http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html`. Мы можем преобразовать захваченные кадры из пространства RGB в пространство HSV, а затем использовать пороговые значения цветов для отслеживания любого заданного объекта. Следует отметить, что для выбора подходящих диапазонов цветов в процессе назначения пороговых значений мы должны знать распределение цветов объекта.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import cv2
import numpy as np
```

Определим функцию, которая будет захватывать текущий кадр из веб-камеры. Начнем с чтения кадра из объекта захвата видео.

```
# Определение функции, получающей текущий кадр из веб-камеры
def get_frame(cap, scaling_factor):
    # Чтение текущего кадра из объекта захвата видео
    _, frame = cap.read()
```

Изменим размер кадра в соответствии с масштабным множителем и вернем его.

```
    # Изменение размера изображения
    frame = cv2.resize(frame, None, fx=scaling_factor,
        fy=scaling_factor, interpolation=cv2.INTER_AREA)

    return frame
```

Определим основную функцию и инициализируем объект захвата видео.

```
if __name__=='__main__':
    # Определение объекта захвата видео
    cap = cv2.VideoCapture(0)
```

Определим масштабный множитель для изменения размера изображений.

```
    # Определение масштабного множителя для изображений
    scaling_factor = 0.5
```

Выполним бесконечный цикл до тех пор, пока пользователь не нажмет клавишу <Esc>. Начнем с захвата текущего кадра.

```
    # Чтение кадров из веб-камеры до тех пор, пока
    # пользователь не нажмет клавишу <Esc>
    while True:
        # Захват текущего кадра
        frame = get_frame(cap, scaling_factor)
```

Преобразуем изображение в цветовое пространство HSV, используя встроенную функцию, доступную в библиотеке OpenCV.

```
        # Преобразование изображения в цветовое пространство HSV
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Определим приблизительный цветовой диапазон HSV для цвета человеческой кожи.

```
# Определение диапазона цветов кожи в HSV
lower = np.array([0, 70, 60])
upper = np.array([50, 150, 255])
```

Ограничим HSV-изображение для создания маски.

```
# Ограничение HSV-изображения для получения
# только цветов кожи
mask = cv2.inRange(hsv, lower, upper)
```

Выполним операцию побитового И для маски и исходного изображения.

```
# Выполнение операции побитового И для маски
# и исходного изображения
img_bitwise_and = cv2.bitwise_and(frame, frame, mask=mask)
```

Выполним медианное размытие границ для сглаживания изображения.

```
# Выполнение медианного размытия
img_median_blurred = cv2.medianBlur(img_bitwise_and, 5)
```

Отобразим входной и выходной кадры.

```
# Отображение входного и выходного кадров
cv2.imshow('Входное изображение', frame)
cv2.imshow('Выходное изображение', img_median_blurred)
```

Проверим, не нажал ли пользователь клавишу <Esc>. Если это действительно так, выходим из цикла.

```
# Проверка того, не нажал ли пользователь клавишу <Esc>
c = cv2.waitKey(5)
if c == 27:
    break
```

Выйдя из цикла, необходимо убедиться в том, что все окна закрыты надлежащим образом.

```
# Закрытие всех окон
cv2.destroyAllWindows()
```

Полный код этого примера содержится в файле `colorspaces.py`. В процессе выполнения кода на экране появятся два изображения. Окно **Входное изображение** представляет захваченный кадр (рис. 13.2).

Рис. 13.2

Во втором окне (**Выходное изображение**) отображается маска кожи (рис. 13.3).

Рис. 13.3

Отслеживание объектов путем вычитания фоновых изображений

Вычитание фона — это алгоритм, которая моделирует фоновое изображение в данном видео, а затем использует эту модель для обнаружения движущихся объектов. Этот подход широко применяется в программах сжатия видео, а также в системах видеонаблюдения. Он отлично справляется со своими задачами в тех случаях, когда требуется обнаруживать движущиеся объекты в пределах статической сцены. Работа данного алгоритма в основном заключается в обнаружении фонового изображения, построении модели для него и последующем вычитании фона из текущего кадра для получения переднего плана. Этот передний план и соответствует движущимся объектам.

Здесь одним из главных этапов является создание модели фона. Это не то же самое, что вычисление разности между последовательными кадрами. Фактически мы моделируем фон и обновляем его в реальном времени, что превращает этот процесс в адаптивный алгоритм, способный приспосабливаться к скользящей базовой линии. Вот почему он работает намного лучше, чем вычитание кадров.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import cv2
import numpy as np
```

Определим функцию, которая будет захватывать текущий кадр.

```
# Определение функции, захватывающей текущий кадр из веб-камеры
def get_frame(cap, scaling_factor):
    # Чтение текущего кадра из объекта захвата видео
    _, frame = cap.read()
```

Изменим размер изображения и вернем измененное изображение.

```
    # Изменение размера изображения
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    return frame
```

Определим основную функцию и инициализируем объект захвата видео.

```
if __name__=='__main__':
    # Определение объекта захвата видео
    cap = cv2.VideoCapture(0)
```

Определим объект вычитания фона.

```
# Определение объекта вычитания фона
bg_subtractor = cv2.createBackgroundSubtractorMOG2()
```

Определим историю и скорость обучения. В отношении того, что такое "история" (history), комментарий говорит сам за себя.

```
# Определим количество предыдущих кадров, которые следует
# использовать для обучения. Этот фактор управляет скоростью
# обучения алгоритма. Под скоростью обучения подразумевается
# скорость, с которой ваша модель будет учиться распознавать
# фон. Чем выше значение параметра 'history', тем ниже
# скорость обучения. Вы можете поэкспериментировать с этим
# значением, чтобы увидеть, как оно влияет на результат.
history = 100

# Определение скорости обучения
learning_rate = 1.0/history
```

Выполним бесконечный цикл до тех пор, пока пользователь не нажмет клавишу <Esc>. Начнем с захвата текущего кадра.

```
# Чтение кадров из веб-камеры до тех пор,
# пока пользователь не нажмет клавишу <Esc>
while True:
    # Захват текущего кадра
    frame = get_frame(cap, 0.5)
```

Вычислим маску, используя объект вычитания фона, определенный ранее.

```
    # Вычисление маски
    mask = bg_subtractor.apply(frame, learningRate=learning_rate)
```

Преобразуем маску из градаций серого в RGB.

```
    # Преобразование изображения из градаций серого
    # в пространство RGB
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
```

Выведем входное и выходное изображения.

```
    # Вывод изображений
    cv2.imshow('Выходное изображение', frame)
    cv2.imshow('Выходное изображение', mask & frame)
```

Проверим, не нажал ли пользователь клавишу <Esc>. Если это действительно так, выходим из цикла.

```
# Проверка того, не нажал ли пользователь клавишу <Esc>
c = cv2.waitKey(10)
if c == 27:
    break
```

Выйдя из цикла, убедимся в том, что объект захвата видео сброшен, а все окна закрыты надлежащим образом.

```
# Сброс объекта захвата видео
cap.release()
# Закрытие всех окон
cv2.destroyAllWindows()
```

Полный код примера содержится в файле `background_subtraction.py`. В процессе выполнения кода вы увидите окно, в котором отображается живое видео. Если вы немного подвигаетесь, то сможете частично увидеть себя (рис. 13.4).

Рис. 13.4

Как только вы прекратите движение, изображение начнет затухать, поскольку теперь вы станете частью фона. Алгоритм будет воспринимать ваше изображение как фон и соответствующим образом обновлять модель (рис. 13.5).

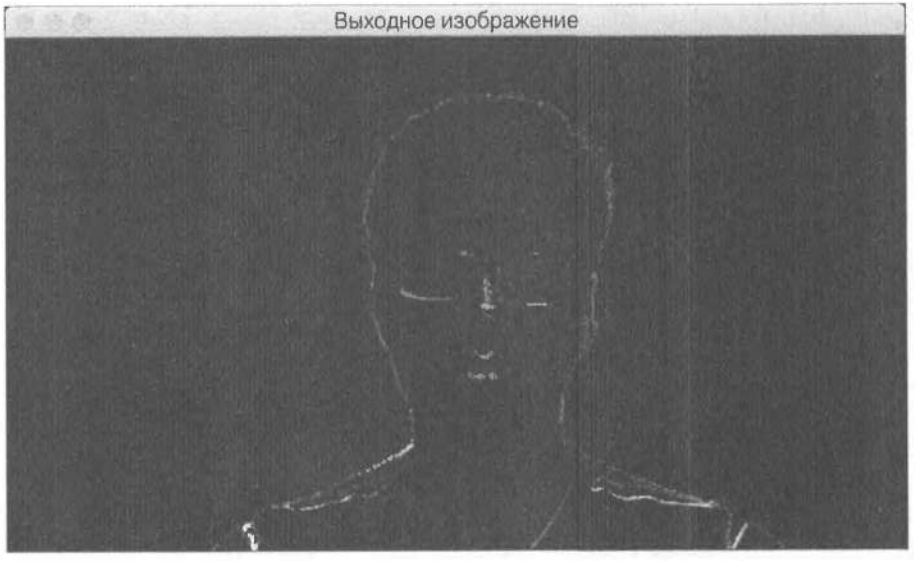

Рис. 13.5

Если вы будете оставаться неподвижным, изображение продолжит затухать (рис. 13.6).

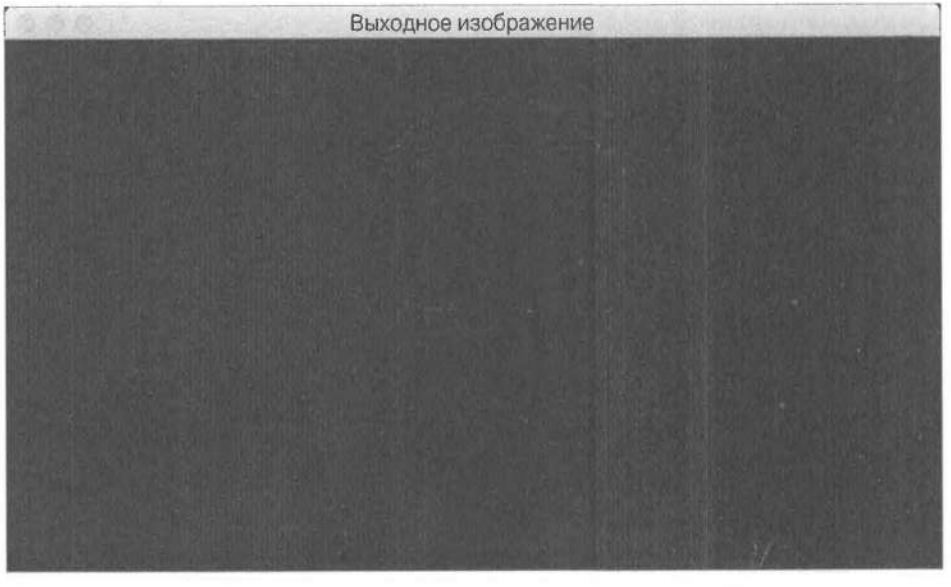

Рис. 13.6

Процесс затухания указывает на то, что текущая сцена становится частью модели фона.

Создание интерактивного трекера объектов с помощью алгоритма CAMShift

Отслеживание, основанное на использовании цветовых пространств, позволяет контролировать цветные объекты, но сначала мы должны определить цвет. Это существенное ограничение! Давайте рассмотрим, каким образом можно выделить объект в живом видео, а затем отслеживать его с помощью трекера. Именно здесь на помощь приходит алгоритм CAMShift (Continuously Adaptive Mean Shift — непрерывный адаптивный сдвиг среднего). В основном он представляет собой адаптивную версию алгоритма сдвига среднего.

Чтобы понимать, как работает алгоритм CAMShift, следует сначала разобраться в том, как работает процесс сдвига. Предположим, нас интересует какая-то область заданного кадра. Мы выбрали эту область, поскольку она содержит объект, представляющий для нас интерес. Мы хотим отслеживать этот объект и потому ограничили контуром его приблизительные границы, которые и определяют "интересующую нас область". Мы хотим, чтобы наш трекер отслеживал этот объект по мере его перемещения в видео.

Для этого мы выбираем набор точек на основе гистограммы цветов данной области и вычисляем позицию их центроида. Если положение данного центроида совпадает с геометрическим центром этой области, то мы знаем, что объект не двигался. Если это не так, то мы знаем, что объект переместился. Это означает, что мы должны переместить также ограничивающий контур. Перемещение центроида непосредственно указывает нам на направление перемещения объекта. Мы должны переместить наш ограничительный прямоугольник таким образом, чтобы новый центроид стал его геометрическим центром. Мы делаем это для каждого кадра и отслеживаем объект в реальном времени. Отсюда и название алгоритма — "сдвиг среднего", поскольку "среднее" (т.е. центроид) смещается, и мы используем этот факт для отслеживания объекта.

Рассмотрим, как это связано с алгоритмом CAMShift. В алгоритме сдвига среднего есть проблема, заключающаяся в том, что размеры объекта не должны изменяться со временем. Как только мы очерчиваем ограничительный прямоугольник, его размеры остаются неизменными независимо от того, насколько близко или далеко объект находится от камеры. Именно поэтому мы и должны применять алгоритм CAMShift, поскольку он позволяет подгонять размеры ограничительного прямоугольника под размеры объекта. Для получения более подробной информации по этому вопросу посетите веб-страницу `http://docs.opencv.org/3.1.0/db/df8/tutorial_py_meanshift.html`. Рассмотрим конкретный пример создания такого трекера.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import cv2
import numpy as np
```

Определим класс, который будет обрабатывать всю функциональность, связанную с отслеживанием объектов.

```
# Определение класса, содержащего всю функциональность,
# необходимую для отслеживания объектов
class ObjectTracker(object):
    def __init__(self, scaling_factor=0.5):
        # Инициализация объекта захвата видео
        self.cap = cv2.VideoCapture(0)
```

Захватим текущий кадр.

```
        # Захват кадра из веб-камеры
        _, self.frame = self.cap.read()
```

Зададим масштабный множитель.

```
        # Масштабный множитель для захваченного изображения
        self.scaling_factor = scaling_factor
```

Изменим размер изображения.

```
        # Изменение размера изображения
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)
```

Создадим окно для отображения результата.

```
        # Создание окна для отображения кадра
        cv2.namedWindow('Object Tracker')
```

Установим функцию обратного вызова, позволяющую отслеживать перемещение мыши.

```
        # Установка функции обратного вызова, отслеживающей
        # события мыши
        cv2.setMouseCallback('Object Tracker', self.mouse_event)
```

Инициализируем переменные для отслеживания прямоугольной рамки выбора.

```
        # Инициализация переменной, связанной с ограничительным
        # прямоугольником выбранной области
```

```
        self.selection = None
        # Инициализация переменной, связанной с начальной
        # позицией
        self.drag_start = None
        # Инициализация переменной, связанной с состоянием
        # отслеживания
        self.tracking_state = 0
```

Определим функцию для отслеживания событий мыши.

```
    # Определение метода для отслеживания событий мыши
    def mouse_event(self, event, x, y, flags, param):
        # Преобразование координат X и Y в 16-битовые
        # целые числа NumPy
        x, y = np.int16([x, y])
```

Нажатие левой кнопки мыши указывает на то, что пользователь начал вычерчивать ограничительный прямоугольник.

```
        # Проверка нажатия кнопки мыши
        if event == cv2.EVENT_LBUTTONDOWN:
            self.drag_start = (x, y)
            self.tracking_state = 0
```

Если пользователь в настоящий момент перетаскивает мышь для установки размера выделенной прямоугольной области, то отследим ширину и высоту этой области.

```
        # Проверка того, не начал ли пользователь выделять область
        if self.drag_start:
            if flags & cv2.EVENT_FLAG_LBUTTON:
                # Извлечение размеров кадра
                h, w = self.frame.shape[:2]
```

Установим начальные значения координат X и Y прямоугольника.

```
                # Получение начальной позиции
                xi, yi = self.drag_start
```

Получим максимальную и минимальную координату, чтобы определить направление, в котором пользователь перетаскивает мышь, вычерчивая прямоугольник.

```
                # Получение максимальной и минимальной координаты
                x0, y0 = np.maximum(0, np.minimum([xi, yi], [x, y]))
                x1, y1 = np.minimum([w, h], np.maximum([xi, yi],
                        [x, y]))
```

Сбросим переменную selection.

```
# Сброс переменной selection
self.selection = None
```

Завершим выделение прямоугольной области.

```
# Завершение выделения прямоугольной области
if x1-x0 > 0 and y1-y0 > 0:
    self.selection = (x0, y0, x1, y1)
```

Если выделение завершено, устанавливаем флаг, указывающий на то, что мы должны отслеживать движение объекта, находящегося в выделенной прямоугольной области.

```
else:
    # Если выделение завершено, начать отслеживание
    self.drag_start = None
    if self.selection is not None:
        self.tracking_state = 1
```

Определим метод для отслеживания объекта.

```
# Метод, начинающий отслеживание объекта
def start_tracking(self):
    # Итерируем до тех пор, пока пользователь не нажмет
    # клавишу <Esc>
    while True:
        # Захват кадра из веб-камеры
        _, self.frame = self.cap.read()
```

Изменим размер кадра.

```
# Изменение размера входного кадра
self.frame = cv2.resize(self.frame, None,
        fx=self.scaling_factor,
        fy=self.scaling_factor,
        interpolation=cv2.INTER_AREA)
```

Создадим копию кадра. Впоследствии она нам понадобится.

```
# Создание копии кадра
vis = self.frame.copy()
```

Преобразуем цветовое пространство кадра из RGB в HSV.

```
# Преобразование кадра в цветовое пространство HSV
hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)
```

Создадим маску на основании предварительно установленных пороговых значений.

```
# Создание маски на основании предварительно
# установленных пороговых значений
mask = cv2.inRange(hsv, np.array((0., 60., 32.)),
                   np.array((180., 255., 255.)))
```

Проверим, выделил ли пользователь область.

```
# Проверка выделения пользователем области
if self.selection:
    # Извлечение координат выделенного прямоугольника
    x0, y0, x1, y1 = self.selection

    # Извлечем окно отслеживания
    self.track_window = (x0, y0, x1-x0, y1-y0)
```

Извлечем интересующие нас области из HSV-изображения, а также маску. Вычислим на основании этой информации гистограмму области изображения.

```
# Извлечение интересующей нас области
hsv_roi = hsv[y0:y1, x0:x1]
mask_roi = mask[y0:y1, x0:x1]

# Вычисление гистограммы интересующей нас области
# HSV-изображения с использованием маски
hist = cv2.calcHist([hsv_roi], [0], mask_roi,
                    [16], [0, 180])
```

Нормализуем гистограмму.

```
# Нормализация и переформирование гистограммы
cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX);
self.hist = hist.reshape(-1)
```

Извлечем интересующую нас область из исходного кадра.

```
# Извлечение интересующей нас области из кадра
vis_roi = vis[y0:y1, x0:x1]
```

Вычислим результат применения побитовой операции НЕ к интересующей нас области. Это делается исключительно в целях ее отображения.

```
# Вычисление негативного изображения
# (исключительно в целях отображения)
cv2.bitwise_not(vis_roi, vis_roi)
vis[mask == 0] = 0
```

Проверим, находится ли система в состоянии отслеживания.

```
# Проверка того, находится ли система
# в состоянии "отслеживание"
if self.tracking_state == 1:
    # Сброс переменной selection variable
    self.selection = None
```

Вычислим проекцию гистограммы на просвет.

```
# Вычисление проекции гистограммы на просвет
hsv_backproj = cv2.calcBackProject([hsv],
        [0], self.hist, [0, 180], 1)
```

Применим операцию побитового И к гистограмме и маске.

```
# Вычисление результата применения операции
# побитового И к проекции гистограммы на
# просвет и маске
hsv_backproj &= mask
```

Определим критерий прекращения работы трекера.

```
# Определение критерия для прекращения
# работы трекера
term_crit = (cv2.TERM_CRITERIA_EPS |
        cv2.TERM_CRITERIA_COUNT, 10, 1)
```

Применим алгоритм CAMShift к проекции гистограммы на просвет.

```
# Применение алгоритма CAMShift к 'hsv_backproj'
track_box, self.track_window =
        cv2.CamShift(hsv_backproj,
        self.track_window, term_crit)
```

Вычертим эллипс вокруг объекта и отобразим его.

```
# Вычерчивание эллипса вокруг объекта
cv2.ellipse(vis, track_box, (0, 255, 0), 2)
# Отображение живого видео
cv2.imshow('Трекер объекта', vis)
```

Выходим из цикла, если пользователь нажал клавишу <Esc>.

```
# Прекратить, если пользователь нажал клавишу <Esc>
c = cv2.waitKey(5)
if c == 27:
    break
```

После выхода из цикла необходимо убедиться в том, что все окна закрыты надлежащим образом.

```
# Закрытие всех окон
cv2.destroyAllWindows()
```

Определим основную функцию и начнем отслеживание.

```
if __name__ == '__main__':
    # Запуск трекера
    ObjectTracker().start_tracking()
```

Полный код примера содержится в файле `camshift.py`. После запуска программы откроется окно, в котором отображается живое видео, поступающее от веб-камеры.

Возьмите и удерживайте в руке какой-нибудь объект, а затем очертите прямоугольник вокруг него. Вычерчивая прямоугольник, отведите указатель мыши подальше от его конечной позиции. Изображение будет выглядеть примерно так, как показано на рис. 13.7.

Рис. 13.7

Завершив выделение, переместите указатель мыши в другую позицию, чтобы зафиксировать прямоугольник. Это событие запустит процесс отслеживания (рис. 13.8).

Перемещайте объект в различных направлениях, чтобы убедиться в том, что он отслеживается (рис. 13.9).

Все работает так, как и ожидалось. Вы можете перемещать объект в разные стороны, чтобы наблюдать за тем, как он отслеживается в режиме реального времени.

Рис. 13.8

Рис. 13.9

Отслеживание объектов с использованием оптических потоков

Оптический поток — весьма популярная методика, широко применяемая в компьютерном зрении. Для отслеживания объектов в ней используются *особые (признаковые) точки* (feature points) изображений. Индивидуальные особые точки отслеживаются в последовательных кадрах видео в режиме реального времени. Когда мы обнаруживаем набор особых точек в данном кадре, мы отслеживаем его, вычисляя векторы смещений и отображая движение особых точек в видеопоследовательности. Эти векторы смещений

известны как *векторы движения* (motion vectors). Существует множество методов вычисления оптического потока, но наиболее популярным из них является метод Лукаса–Канаде. Вот ссылка на оригинальную статью, в которой описан этот метод:

```
http://cseweb.ucsd.edu/classes/sp02/cse252/lucaskanade81.pdf
```

Первый шаг состоит в извлечении особых точек из текущего кадра. Для каждой извлекаемой точки создается фрагмент размером 3×3 пикселя с особой точкой в центре. Мы предполагаем, что все точки в каждом фрагменте движутся одинаково. Размер этого окна можно регулировать в зависимости от ситуации.

Для каждого фрагмента ищется соответствие в его окрестности в предыдущем кадре. Наилучшее соответствие выбирается на основании метрики ошибки. Размеры области поиска превышают 3×3, поскольку мы просматриваем ряд различных фрагментов размером 3×3, чтобы выбрать тот, который является ближайшим к текущему фрагменту. Как только мы находим ближайший аналогичный фрагмент, путь между центральной точкой текущего фрагмента и соответствующим ему фрагментом предыдущего кадра становится вектором движения. Аналогичным образом мы вычисляем векторы движения для всех других фрагментов.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import cv2
import numpy as np
```

Определим функцию, запускающую трекинг с использованием оптического потока. Начнем с инициализации объекта захвата видео и масштабного множителя.

```
# Определим функцию для отслеживания объекта
def start_tracking():
    # Инициализация объекта захвата видео
    cap = cv2.VideoCapture(0)

    # Определение масштабного множителя для кадров
    scaling_factor = 0.5
```

Определим количество отслеживаемых и пропускаемых кадров.

```
    # Количество отслеживаемых кадров
    num_frames_to_track = 5

    # Шаг пропуска
    num_frames_jump = 2
```

Инициализируем переменные, связанные с путями отслеживания и индексом кадра.

```
# Инициализация переменных
tracking_paths = []
frame_index = 0
```

Определим параметры отслеживания, такие как размер окна, максимальный уровень и критерий прекращения отслеживания.

```
# Определение параметров отслеживания
tracking_params = dict(winSize = (11, 11), maxLevel = 2,
        criteria = (cv2.TERM_CRITERIA_EPS |
                    cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

Итерируем в бесконечном цикле до тех пор, пока пользователь не нажмет клавишу <Esc>. Начнем с захвата текущего кадра и изменения его размеров.

```
# Итерирование до тех пор, пока пользователь
# не нажмет клавишу <Esc>
while True:
    # Захват текущего кадра
    _, frame = cap.read()

    # Изменение размеров кадра
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)
```

Преобразуем кадр из RGB в градации серого.

```
    # Преобразование в градации серого
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Создадим копию кадра.

```
    # Создание копии кадра
    output_img = frame.copy()
```

Проверим, превышает ли длина отслеживаемых путей нуль.

```
    if len(tracking_paths) > 0:
        # Получение изображений
        prev_img, current_img = prev_gray, frame_gray
```

Организуем особые точки.

```
        # Организация особых точек
        feature_points_0 = np.float32([tp[-1] for tp in
                tracking_paths]).reshape(-1, 1, 2)
```

Вычислим оптический поток на основании предыдущего и текущего изображений, используя особые точки и параметры отслеживания.

```
# Вычисление оптического потока
feature_points_1, _, _ = cv2.calcOpticalFlowPyrLK(
        prev_img, current_img, feature_points_0,
        None, **tracking_params)
# Вычисление обратного оптического потока
feature_points_0_rev, _, _ =
        cv2.calcOpticalFlowPyrLK(
            current_img, prev_img, feature_points_1,
            None, **tracking_params)
# Вычисление разности между прямым
# и обратным оптическими потоками
diff_feature_points = abs(feature_points_0 -
        feature_points_0_rev).reshape(-1, 2).max(-1)
```

Извлечем подходящие особые точки.

```
# Извлечение подходящих точек
good_points = diff_feature_points < 1
```

Инициализируем переменную для новых путей отслеживания.

```
# Инициализация переменной
new_tracking_paths = []
```

Итерируем по всем подходящим особым точкам и вычерчиваем окружности вокруг них.

```
# Итерации по всем подходящим особым точкам
for tp, (x, y), good_points_flag in
        zip(tracking_paths,
        feature_points_1.reshape(-1, 2),
        good_points):
    # Продолжение, если флаг не равен true
    if not good_points_flag:
        continue
```

Присоединим координаты X и Y, убеждаясь в том, что количество кадров, которые мы должны отслеживать, не превышено.

```
    # Присоединение координат X и Y и проверка того,
    # не превышает ли длина списка пороговое значение
    tp.append((x, y))
    if len(tp) > num_frames_to_track:
        del tp[0]

    new_tracking_paths.append(tp)
```

Вычертим окружность вокруг точки. Обновим пути отслеживания и вычертим линии, используя новые пути для отображения движения.

```
# Вычерчивание окружности вокруг особых точек
cv2.circle(output_img, (x, y), 3, (0, 255, 0), -1)

# Обновление путей отслеживания
tracking_paths = new_tracking_paths

# Вычерчивание линий
cv2.polylines(output_img, [np.int32(tp) for tp in
              tracking_paths], False, (0, 150, 0))
```

Войдем в блок условия после пропуска определенного ранее количества кадров.

```
# Вход в блок 'if' после пропуска подходящего количества кадров
if not frame_index % num_frames_jump:
    # Создание маски и вычерчивание окружностей
    mask = np.zeros_like(frame_gray)
    mask[:] = 255
    for x, y in [np.int32(tp[-1]) for tp in tracking_paths]:
        cv2.circle(mask, (x, y), 6, 0, -1)
```

Вычислим подходящие особые точки (признаки), подлежащие отслеживанию, используя встроенную функцию с такими параметрами, как маска, максимальное количество углов, уровень качества, минимальное расстояние и размер блока.

```
# Вычисление подходящих признаков для отслеживания
feature_points = cv2.goodFeaturesToTrack(frame_gray,
        mask = mask, maxCorners = 500, qualityLevel = 0.3,
        minDistance = 7, blockSize = 7)
```

В случае, если особые точки существуют, присоединить их к путям отслеживания.

```
# Проверка существования особых точек; если они
# существуют, присоединить их к путям отслеживания
if feature_points is not None:
    for x, y in np.float32(feature_points).reshape(-1, 2):
        tracking_paths.append([(x, y)])
```

Обновим две переменные, связанные с индексом кадра и предыдущим изображением в градациях серого.

```
# Обновление переменных
frame_index += 1
prev_gray = frame_gray
```

Отобразим результат.

```
# Отображение результата
cv2.imshow('Оптический поток', output_img)
```

Проверим, не нажал ли пользователь клавишу <Esc>. Если это действительно так, выходим из цикла.

```
# Проверка того, не нажал ли пользователь клавишу <Esc>
c = cv2.waitKey(1)
if c == 27:
    break
```

Определим основную функцию и начнем отслеживание. Сразу же после того, как трекер прекратит работу, необходимо убедиться в том, что все окна закрыты надлежащим образом.

```
if __name__ == '__main__':
    # Запуск трекера
    start_tracking()

    # Закрытие всех окон
    cv2.destroyAllWindows()
```

Полный код примера содержится в файле `optical_flow.py`. После запуска этого кода откроется окно, в котором отображается живое видео, поступающее от веб-камеры, вместе с особыми точками (рис. 13.10).

Рис. 13.10

Если вы немного переместитесь, то увидите линии, отображающие движение особых точек (рис. 13.11).

Рис. 13.11

Если вы переместитесь в противоположном направлении, то соответствующим образом изменится и направление линий (рис. 13.12).

Рис. 13.12

Обнаружение и отслеживание лиц

Обнаружение лиц (face detection) — это определение местоположения лица на изображении. Это понятие часто путают с *распознаванием* лиц (face recognition) — процессом идентификации личности. В типичных биометри-

ческих системах используют как обнаружение, так и распознавание лиц. Сначала с помощью техники обнаружения лиц находят местоположение лица на изображении, а затем с помощью техники распознавания лиц устанавливают личность человека. В этом разделе рассматривается автоматизация обнаружения и отслеживания лиц в живом видеопотоке.

Использование каскадов Хаара для обнаружения лиц

Для обнаружения лиц в видео мы будем использовать *каскады Хаара*. В данном случае каскады Хаара — это каскадные классификаторы, основанные на признаках Хаара. Этот метод обнаружения объектов был впервые предложен *Полом Виолой* и *Майклом Джонсом* в их основополагающей исследовательской статье, опубликованной в 2001 году. Вы сможете ознакомиться с ней по следующему адресу:

```
https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
```

В этой статье описан эффективный метод машинного обучения, который может быть использован для обнаружения любого объекта.

В данном методе применяется усиленный каскад простых классификаторов. Этот каскад используется для создания общего классификатора, работающего с высокой степенью точности. Уместность применения каскада обусловлена тем, что он позволяет обойти проблему построения простого одностадийного классификатора столь же высокой точности, поскольку решение этой проблемы сопряжено с большими вычислительными трудностями.

Рассмотрим пример, в котором нужно обнаружить, скажем, такой объект, как теннисный мяч. Для создания подобного детектора объектов нужна система, которую можно обучить распознаванию теннисного мяча. Такая система должна уметь определять, есть ли на данном изображении теннисный мяч. Мы должны обучить систему, используя множество изображений теннисного мяча. Нам также понадобятся изображения, на которых теннисный мяч отсутствует. Это позволит системе научиться отличать его от других объектов.

Мы собираемся создать точную модель, поэтому она будет сложной. Следовательно, мы не сможем выполнять ее в режиме реального времени. Слишком простая модель будет неточной. В мире машинного обучения с необходимостью достижения подобного компромисса между скоростью и точностью приходится сталкиваться довольно часто. Метод Виолы–Джонса преодолевает эту проблему за счет создания набора простых классификаторов. Затем эти классификаторы каскадируются, образуя единый классификатор, характеризующийся надежностью и высокой точностью.

Сейчас мы узнаем, как применить эти возможности для обнаружения лиц. Создание системы машинного обучения для обнаружения лиц мы начнем с извлечения признаков. Алгоритмы машинного обучения будут использовать эти признаки для того, чтобы понять, что собой представляет лицо. И здесь нам на помощь приходят *признаки Хаара*. Это просто суммы и разности фрагментов изображения, и их вычисление фактически не представляет трудностей. Чтобы метод оставался надежным в отношении масштабирования, мы повторяем всю процедуру для изображений различного размера. Если вы хотите узнать больше об этом методе в рамках учебного руководства, воспользуйтесь следующей ссылкой:

```
http://www.cs.ubc.ca/~lowe/425/slides/13-ViolaJones.pdf
```

Как только признаки извлечены, мы пропускаем их через наш каскад простых классификаторов. Мы проверяем различные прямоугольные подобласти изображения и отбрасываем те, которые не содержат изображений лица. Это позволяет быстро получить ответ. Вычисление признаков удается ускорить за счет использования так называемых интегральных изображений.

Использование интегральных изображений для извлечения признаков

Для вычисления признаков Хаара нам нужно вычислить суммы и разности многих подобластей изображения. Эти операции должны выполняться с использованием различных масштабов изображения, что требует интенсивных вычислений. Чтобы создаваемая система могла работать в режиме реального времени, мы используем интегральные изображения. Обратимся к рис. 13.13.

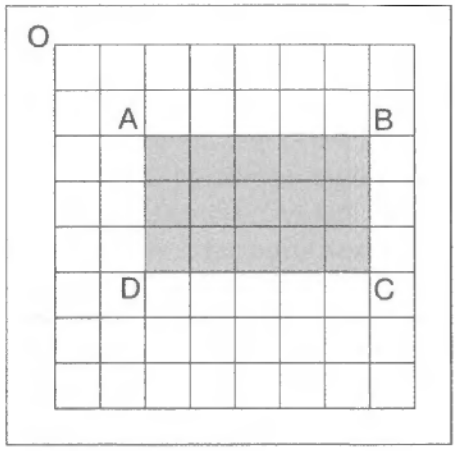

Рис. 13.13

Если нам необходимо просуммировать пиксели в прямоугольной области ABCD этого изображения, то вовсе необязательно проходить по всем пикселям, принадлежащим данному прямоугольнику. Пусть OP обозначает область прямоугольника, определенного с помощью его левого верхнего угла O и точки P, расположенной на другом конце диагонали. Тогда для расчета площади прямоугольной области ABCD можно воспользоваться следующей формулой:

$$\textit{Площадь прямоугольника } ABCD = OC - (OB + OD - OA)$$

Что особенного в этой формуле? Если вы заметили, нам не пришлось организовывать никаких циклов или пересчитывать какие-либо прямоугольные области. Все значения справа от знака равенства уже доступны, поскольку они вычисляются на предыдущих стадиях программы. Мы непосредственно используем их для вычисления площади данного прямоугольника. Рассмотрим пример создания системы обнаружения лиц.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import cv2
import numpy as np
```

Загрузим файл каскада Хаара, соответствующий обнаружению лиц.

```
# Загрузка файла каскада Хаара
face_cascade = cv2.CascadeClassifier(
        'haar_cascade_files/haarcascade_frontalface_default.xml')

# Проверка корректности загрузки файла каскада
if face_cascade.empty():
    raise IOError('Unable to load the face cascade classifier xml file')
```

Инициализация объекта захвата видео.

```
# Инициализируем объект захвата видео
cap = cv2.VideoCapture(0)

# Определение масштабного множителя
scaling_factor = 0.5
```

Выполняем бесконечный цикл до тех пор, пока пользователь не нажмет клавишу <Esc>. Захватываем текущий кадр.

```
# Итерируем до тех пор, пока пользователь не нажмет
# клавишу <Esc>
while True:
    # Захват текущего кадра
    _, frame = cap.read()
```

Изменим размер кадра.

```
# Изменение размера кадра
frame = cv2.resize(frame, None,
    fx=scaling_factor, fy=scaling_factor,
    interpolation=cv2.INTER_AREA)
```

Преобразуем изображение в градации серого.

```
# Преобразование в градации серого
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Запустим детектор лиц для изображения в градациях серого.

```
# Выполнение детектора лиц для изображения
# в градациях серого
face_rects = face_cascade.detectMultiScale(gray, 1.3, 5)
```

Итерируем по обнаруженным лицам и вычерчиваем прямоугольники вокруг них.

```
# Вычерчивание прямоугольника вокруг лица
for (x,y,w,h) in face_rects:
    cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 3)
```

Отобразим результат.

```
# Отображение результата
cv2.imshow('Детектор лиц', frame)
```

Проверим, не нажал ли пользователь клавишу <Esc>. Если это действительно так, выходим из цикла.

```
# Проверка того, не нажал ли пользователь клавишу <Esc>
c = cv2.waitKey(1)
if c == 27:
    break
```

После выхода из цикла следует убедиться в том, что объект захвата кадра сброшен, а все окна закрыты надлежащим образом.

```
# Освобождение объекта захвата видео
cap.release()

# Закрытие всех окон
cv2.destroyAllWindows()
```

Полный код примера содержится в файле `face_detector.py`. Выполнив код, вы увидите примерно следующее (рис. 13.14).

Рис. 13.14

Отслеживание глаз и определение координат взора

Отслеживание глаз работает во многом аналогично обнаружению лиц. Вместо файла каскада для лиц будем использовать файл каскада для глаз. Создайте новый файл Python и импортируйте следующие пакеты.

```
import cv2
import numpy as np
```

Загрузим файлы каскадов Хаара для лиц и глаз.

```
# Загрузка файлов каскадов Хаара для лиц и глаз
face_cascade = cv2.CascadeClassifier('haar_cascade_files/
        haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haar_cascade_files/
        haarcascade_eye.xml')

# Проверка корректности загрузки файла каскада лица
if face_cascade.empty():
    raise IOError('Unable to load the face cascade classifier
                   xml file')
```

```
# Проверка корректности загрузки файла каскада лиц
if eye_cascade.empty():
    raise IOError('Unable to load the eye cascade classifier
                   xml file')
```

Инициализация объекта захвата видео и определение масштабного множителя.

```
# Инициализация объекта захвата видео
cap = cv2.VideoCapture(0)

# Определение масштабного множителя
ds_factor = 0.5
```

Выполняем бесконечный цикл до тех пор, пока пользователь не нажмет клавишу <Esc>.

```
# Итерируем до тех пор, пока пользователь не нажмет клавишу 'Esc'
while True:
    # Захват текущего кадра
    _, frame = cap.read()
```

Изменим размер кадра.

```
    # Изменение размера кадра
    frame = cv2.resize(frame, None, fx=ds_factor, fy=ds_factor,
                       interpolation=cv2.INTER_AREA)
```

Преобразуем кадр из RGB в градации серого.

```
    # Преобразование в градации серого
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Запустим детектор лиц.

```
    # Выполнение детектора лиц для изображения в градациях серого
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

Для каждого обнаруженного лица выполняем детектор глаз в пределах этой области.

```
    # Выполнение детектора глаз для каждого обнаруженного лица
    for (x,y,w,h) in faces:
        # Извлечение интересующей нас области изображения лица
        # в градациях серого
        roi_gray = gray[y:y+h, x:x+w]
```

Извлечем интересующую нас область и выполним детектор глаз.

```
# Извлечение интересующей нас области цветного
# изображения лица
roi_color = frame[y:y+h, x:x+w]

# Выполнение детектора глаз в интересующей нас области
# изображения в градациях серого
eyes = eye_cascade.detectMultiScale(roi_gray)
```

Вычертим окружности вокруг глаз и отобразим результат.

```
# Вычерчивание окружностей вокруг глаз
for (x_eye,y_eye,w_eye,h_eye) in eyes:
    center = (int(x_eye + 0.5*w_eye), int(y_eye +
              0.5*h_eye))
    radius = int(0.3 * (w_eye + h_eye))
    color = (0, 255, 0)
    thickness = 3
    cv2.circle(roi_color, center, radius, color, thickness)
# Отобразить вывод
cv2.imshow('Детектор глаз', frame)
```

Если пользователь нажал клавишу <Esc>, выйти из цикла.

```
# Проверка того, не нажал ли пользователь клавишу <Esc>
c = cv2.waitKey(1)
if c == 27:
    break
```

После выхода из цикла следует убедиться в том, что объект захвата кадра сброшен, а все окна закрыты надлежащим образом.

```
# Освобождение объекта захвата видео
cap.release()

# Закрытие всех окон
cv2.destroyAllWindows()
```

Полный код примера содержится в файле `eye_detector.py`. Выполнив код, вы увидите примерно следующее (рис. 13.15).

Рис. 13.15

Резюме

Эта глава была посвящена обнаружению и отслеживанию объектов. Было показано, как установить библиотеку OpenCV с поддержкой Python в различных операционных системах. Вы узнали о методе вычитания кадров и его применении для обнаружения движущихся частей в видео. Мы обсудили отслеживание изображений человеческой кожи с помощью цветовых пространств. Мы также рассмотрели метод вычитания фоновых изображений и показали, как его использовать для отслеживания объектов в статических сценах. Мы создали интерактивный трекер объектов, используя алгоритм CAMShift.

Вы узнали о том, как создать трекер объектов на основе оптического потока. Мы обсудили методы обнаружения лиц и рассказали о том, что такое каскады Хаара и интегральные изображения. Эти методы были использованы для создания детектора и трекера глаз. В следующей главе мы рассмотрим искусственные нейронные сети и применим их для создания системы оптического распознавания символов.

14
Искусственные нейронные сети

Эта глава посвящена искусственным нейронным сетям. Мы начнем с краткого введения в искусственные нейронные сети и описания процесса установки соответствующей библиотеки. Мы обсудим, что такое перцептроны и как построить классификатор на их основе. Мы расскажем об одно- и многослойных нейронных сетях. Вы узнаете о том, как использовать нейронные сети для создания векторных квантизаторов. Мы будем анализировать последовательные данные с помощью рекуррентных нейронных сетей. После этого мы прменим искусственные нейронные сети для создания системы оптического распознавания символов.

К концу главы вы освоите следующие темы:

- введение в искусственные нейронные сети;
- создание классификатора на основе перцептрона;
- построение однослойной нейронной сети;
- построение многослойной нейронной сети;
- создание векторного квантизатора;
- анализ последовательных данных с помощью рекуррентных нейронных сетей;
- визуализация символов с использованием базы данных оптического распознавания символов;
- создание системы оптического распознавания символов.

Введение в искусственные нейронные сети

Одной из фундаментальных предпосылок в области искусственного интеллекта является предположение о возможности создания машин, способных выполнять задачи, обычно требующие приложения человеческого интеллекта. Мозг человека обладает поразительной способностью обучаться новому.

Почему бы не использовать модель человеческого мозга для создания машины? *Искусственная нейронная сеть* — это модель, предназначенная для имитации процессов обучения, протекающих в мозге человека.

Искусственные нейронные сети проектируются таким образом, чтобы они могли распознавать базовые образы (закономерности, устойчивые взаимосвязи, скрытые в данных) и учиться на них. Они могут применяться для решения различных задач, таких как классификация, регрессия, сегментация данных и др. Прежде чем предоставить данные нейронной сети, мы должны преобразовать их в числовую форму. Например, мы можем иметь дело с данными самой разной природы, включая визуальные и текстовые данные, временные ряды и т.п. Нам приходится принимать решения относительно того, каким образом следует представлять задачи, чтобы они были понятны нейронным сетям.

Создание нейронной сети

Процесс обучения человека носит иерархический характер. В нейронной сети нашего мозга этот процесс осуществляется в несколько стадий, для каждой из которых характерна своя степень гранулярности. На одних стадиях идет обучение простым вещам, на других — более сложным. В качестве примера рассмотрим визуальное распознавание объекта. Когда мы смотрим на ящик, то сначала мы просто идентифицируем такие его элементы, как углы и ребра. На следующей стадии идентифицируется форма ящика, а на последующей — что собой представляет объект. Этот процесс происходит по-разному для различных задач, но, вероятно, идея вам понятна. Создавая такую иерархию, человеческий мозг быстро разделяет понятия и идентифицирует данный объект.

Чтобы имитировать процесс обучения человека, при построении искусственных нейронных сетей используют слои нейронов. Идея этих нейронов подсказана биологическими процессами, о которых шла речь выше. Каждый слой искусственной нейронной сети представляет собой множество независимых нейронов. Каждый нейрон некоторого слоя соединен с нейронами смежного слоя.

Тренировка нейронной сети

Если мы имеем дело с N-мерными входными данными, то входной слой будет состоять из N нейронов. Если среди наших обучающих (тренировочных) данных выделяются M различных классов, то выходной слой будет состоять из M нейронов. Слои, заключенные между входным и выходным слоями, называются *скрытыми*. Простая нейронная сеть состоит из пары слоев, а глубокая нейронная сеть — из множества слоев.

Рассмотрим случай, когда мы хотим использовать нейронную сеть для классификации данных. Первым шагом является сбор подходящих тренировочных данных и их маркирование. Каждый нейрон действует как простая функция, и нейронная сеть тренируется до тех пор, пока ошибка не станет меньше некоторого заданного значения. В качестве ошибки в основном используют разницу между предсказанным и фактическим выходами. Исходя из того, насколько велика ошибка, нейронная сеть корректирует саму себя и повторно обучается до тех пор, пока не приблизится к решению. Чтобы узнать больше о нейронных сетях, воспользуйтесь следующей ссылкой:

`http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html`

В этой главе мы будем использовать библиотеку NeuroLab, более подробную информацию о которой можно получить на сайте `https://pythonhosted.org/neurolab`. Чтобы установить эту библиотеку, выполните в окне своего терминала следующую команду:

```
$ pip3 install neurolab
```

Установив библиотеку, можете переходить к чтению следующего раздела.

Создание классификатора на основе перцептрона

Перцептрон — строительный кирпичик искусственной нейронной сети. Он представляет собой одиночный нейрон, который получает входные данные (сигнал), выполняет над ними вычисления и выдает выходной сигнал. Для принятия решений перцептрон применяет простую функцию. Предположим, мы имеем дело с N-мерной входной точкой данных. Перцептрон вычисляет взвешенную сумму N чисел, после чего добавляет к ним константу для получения выходного результата. Эта константа называется *смещением* нейрона. Интересно отметить, что столь простые перцептроны используются для проектирования очень сложных глубоких нейронных сетей. Рассмотрим пример создания классификатора на основе перцептрона с использованием библиотеки NeuroLab.

Создайте новый файл Python и импортируйте следующие файлы.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Загрузим входные данные из текстового файла `data_perceptron.txt`. Каждая его строка содержит числа, разделенные пробелом, из которых первые два числа — признаки, а последнее — метка (маркер).

```
# Загрузка входных данных
text = np.loadtxt('data_perceptron.txt')
```

Разделим текст на точки данных и метки.

```
# Разделение точек данных и меток
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))
```

Построим график точек данных.

```
# Построение графика входных данных
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
```

Определим максимальное и минимальное значения, которые могут достигаться в каждом измерении.

```
# Определение максимального и минимального значений
# для каждого измерения
dim1_min, dim1_max, dim2_min, dim2_max = 0, 1, 0, 1
```

Поскольку данные разделены на два класса, для представления выходного результата требуется всего один бит. В результате выходной слой будет содержать только один нейрон.

```
# Количество нейронов в выходном слое
num_output = labels.shape[1]
```

В нашем наборе точки данных — двумерные. Определим перцептрон с двумя входными нейронами, по одному для каждого измерения.

```
# Определение перцептрона с двумя входными нейронами (поскольку
# входные данные — двумерные)
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
perceptron = nl.net.newp([dim1, dim2], num_output)
```

Обучим перцептрон с помощью тренировочных данных.

```
# Тренировка перцептрона с использованием наших данных
error_progress = perceptron.train(data, labels, epochs=100,
                                  show=20, lr=0.03)
```

Отобразим график процесса обучения, используя метрику ошибки.

```
# Построение графика процесса обучения
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
plt.grid()

plt.show()
```

Полный код примера содержится в файле `perceptron_classifier.py`. В процессе выполнения кода на экране отобразятся два графика. На первом снимке экрана представлены входные точки данных (рис. 14.1).

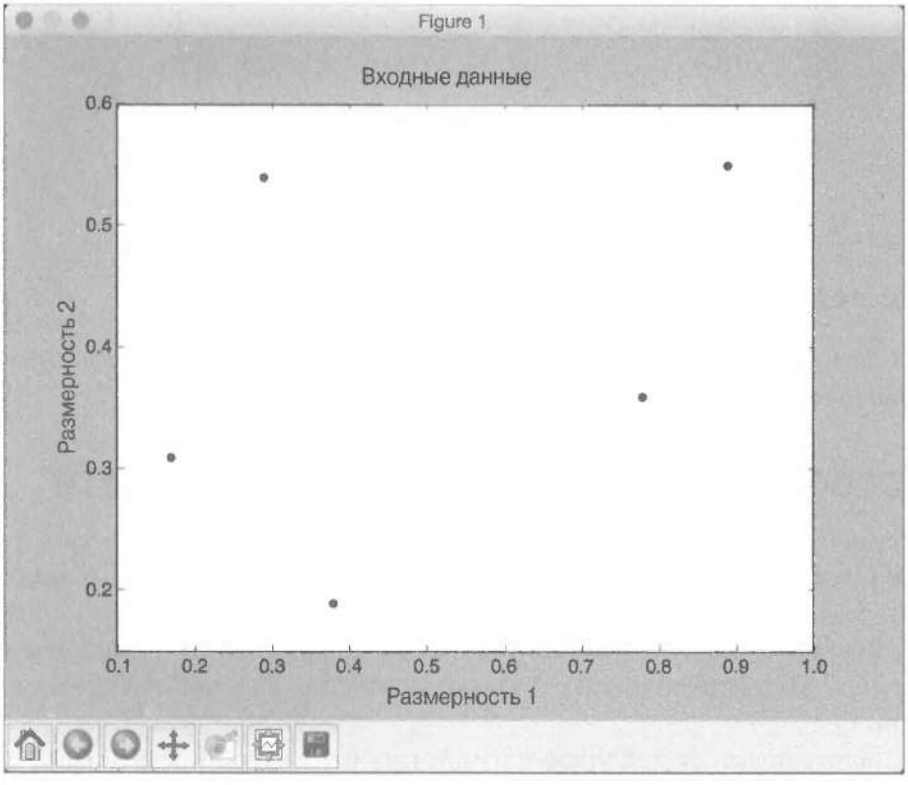

Рис. 14.1

На втором снимке экрана представлена степень продвижения процесса обучения, оцениваемая по величине ошибки (рис. 14.2).

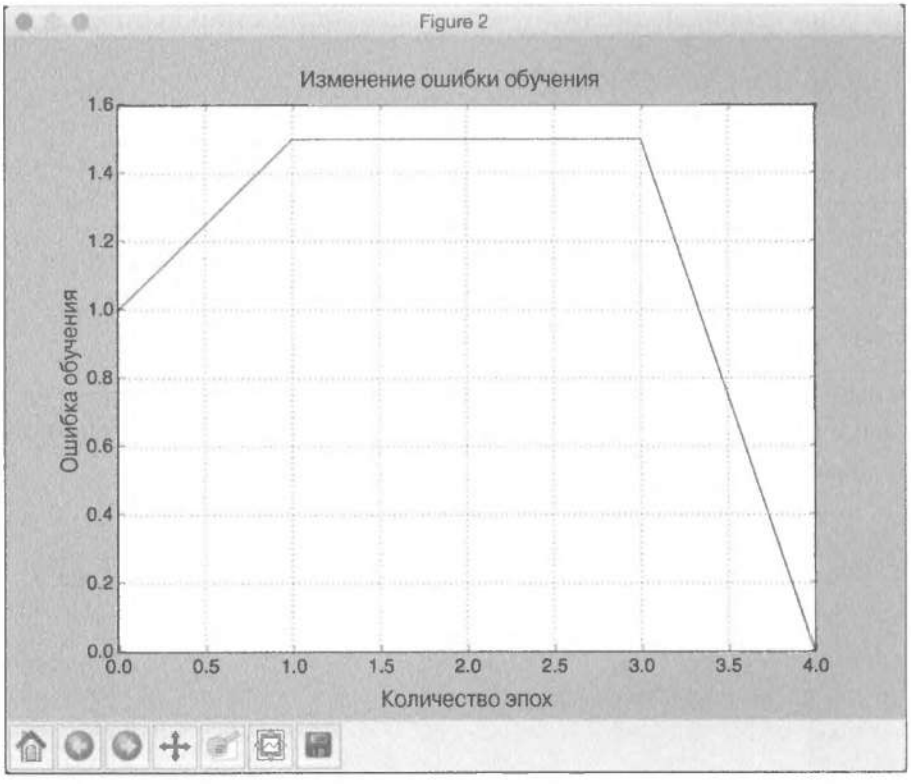

Рис. 14.2

Обратите внимание на то, что в соответствии с предыдущим снимком экрана ошибка снижается до 0 в конце четвертой эпохи.

Построение однослойной нейронной сети

Перцептрон — неплохая отправная точка, но его возможности ограничены. Следующий шаг заключается в том, чтобы заставить набор нейронов действовать как одно целое и посмотреть, чего удастся достигнуть. Давайте создадим однослойную нейронную сеть, которая состоит из независимых нейронов, воздействующих на входные данные для получения выходного результата.

Создайте новый файл Python и импортируйте следующие файлы.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Мы будем использовать входные данные из файла `data_simple_nn.txt`. Каждая строка этого файла содержит четыре числа. Первые два числа обра-

зуют точку данных, а остальные два являются метками. Почему нам нужно использовать два числа в качестве меток? Потому что в нашем наборе данных имеются четыре различных класса, для представления которых нужны два бита. Пойдем дальше и загрузим данные.

```
# Загрузка входных данных
text = np.loadtxt('data_simple_nn.txt')
```

Разделим данные на точки данных и метки.

```
# Разделение данных на точки данных и метки
data = text[:, 0:2]
labels = text[:, 2:]
```

Построим график входных данных.

```
# Построение графика входных данных
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
```

Извлечем минимальное и максимальное значения для каждого измерения (нам не нужно указывать их в коде, как это делалось в предыдущем разделе).

```
# Минимальное и максимальное значения для каждого измерения
dim1_min, dim1_max = data[:,0].min(), data[:,0].max()
dim2_min, dim2_max = data[:,1].min(), data[:,1].max()
```

Определим количество нейронов в выходном слое.

```
# Определение количества нейронов в выходном слое
num_output = labels.shape[1]
```

Определим однослойную нейронную сеть, используя заданные выше параметры.

```
# Определение однослойной нейронной сети
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)
```

Обучим нейронную сеть с помощью тренировочных данных.

```
# Обучение нейронной сети
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)
```

Построим график продвижения процесса обучения.

```
# Построение графика продвижения процесса обучения
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
plt.grid()

plt.show()
```

Определим выборочные тестовые точки данных и запустим для них нейронную сеть.

```
# Выполнение классификатора на тестовых точках данных
print('\nTest results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

Полный код примера содержится в файле `simple_neural_network.py`. В процессе выполнения этого кода на экране отобразятся два графика. На первом снимке экрана представлены входные точки данных (рис. 14.3).

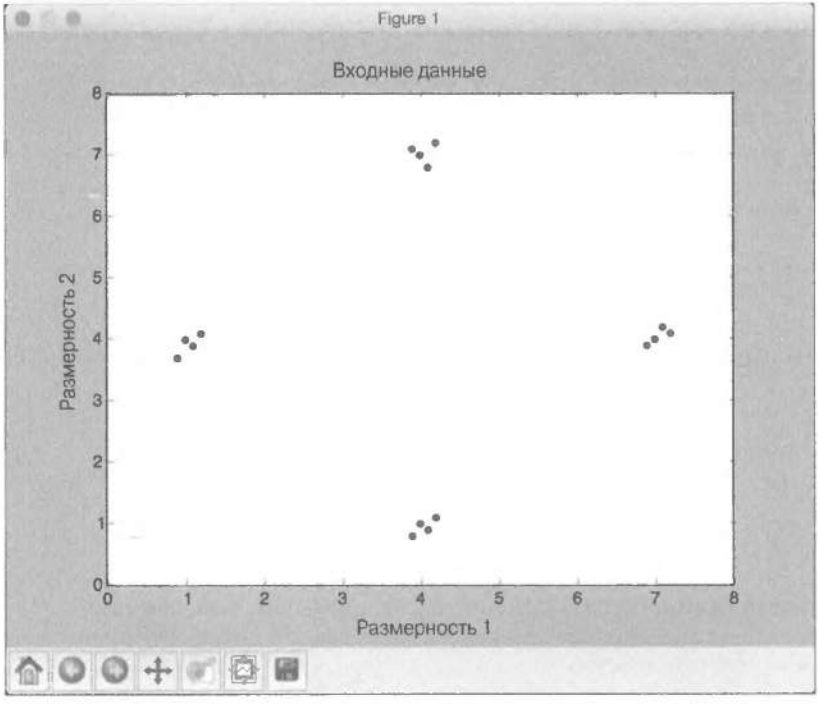

Рис. 14.3

На втором снимке экрана представлен график продвижения процесса обучения (рис. 14.4).

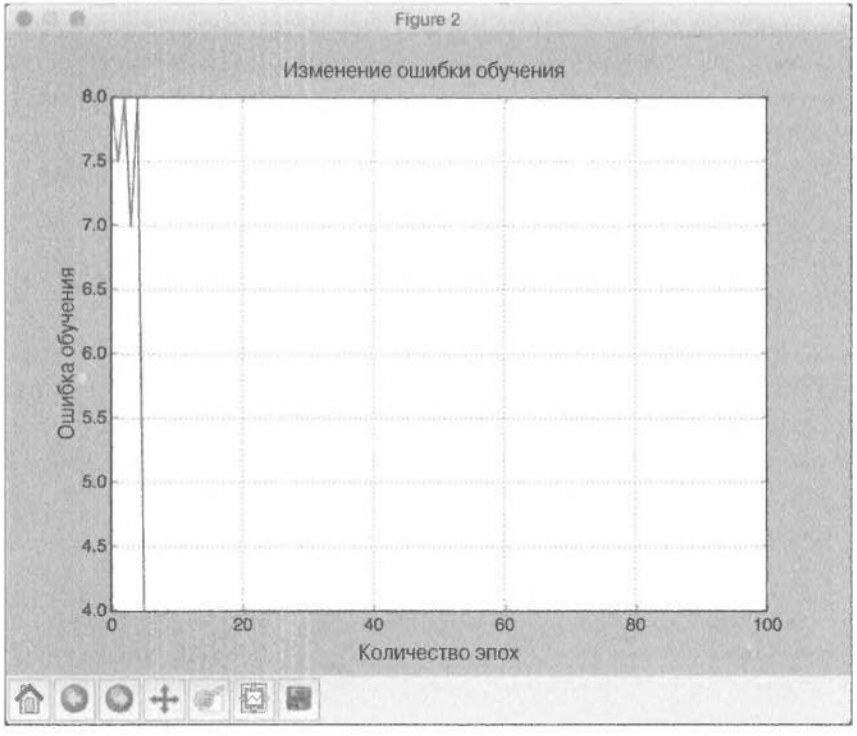

Рис. 14.4

Закрыв окна с этими графиками, вы увидите в окне терминала следующий вывод (рис. 14.5).

```
Epoch: 20; Error: 4.0;
Epoch: 40; Error: 4.0;
Epoch: 60; Error: 4.0;
Epoch: 80; Error: 4.0;
Epoch: 100; Error: 4.0;
The maximum number of train epochs is reached

Test results:
[0.4, 4.3] --> [ 0.  0.]
[4.4, 0.6] --> [ 1.  0.]
[4.7, 8.1] --> [ 1.  1.]
```

Рис. 14.5

Отыскав эти тестовые точки данных на двумерном графике, вы сможете убедиться в том, что результаты были предсказаны корректно.

Построение многослойной нейронной сети

Для получения более высокой точности мы должны предоставить нейронной сети большую свободу. Это означает, что нейронная сеть должна иметь более одного слоя для извлечения базовых закономерностей, существующих среди тестовых данных. Давайте создадим многослойную нейронную сеть, которая нам это обеспечит.

Создайте новый файл Python и импортируйте следующие файлы.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

В двух предыдущих разделах было продемонстрировано использование нейронной сети в качестве классификатора. В этом разделе будет показано, как использовать нейронную сеть в качестве регрессора. Сгенерируем выборочный набор точек данных, используя уравнение $y = 3x^2 + 5$, а затем нормализуем данные.

```
# Генерация тренировочных данных
min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 3 * np.square(x) + 5
y /= np.linalg.norm(y)
```

Переформируем приведенные выше переменные для создания тренировочного набора данных.

```
# Создание данных и меток
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)
```

Построим график входных данных.

```
# Построение графика входных данных
plt.figure()
plt.scatter(data, labels)
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
```

Определим многослойную нейронную сеть с двумя скрытыми слоями. Вы можете спроектировать нейронную сеть любым другим желаемым способом.

В данном случае у нас будет 10 нейронов в первом слое и 6 нейронов во втором слое. Наша задача заключается в предсказании одного значения, поэтому выходной слой будет содержать всего один нейрон.

```
# Определение многослойной нейронной сети с двумя скрытыми
# слоями. Первый скрытый слой состоит из десяти нейронов.
# Второй скрытый слой состоит из шести нейронов.
# Выходной слой состоит из одного нейрона.
nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])
```

Установим метод градиентного спуска в качестве обучающего алгоритма.

```
# Задание градиентного  спуска в качестве обучающего алгоритма
nn.trainf = nl.train.train_gd
```

Обучим нейронную сеть, используя сгенерированный ранее тренировочный набор данных.

```
# Тренировка нейронной сети
error_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)
```

Запустим нейронную сеть для тренировочных точек данных.

```
# Выполнение нейронной сети на тренировочных данных
output = nn.sim(data)
y_pred = output.reshape(num_points)
```

Построим график продвижения процесса обучения.

```
# Построение графика ошибки обучения
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
```

Построим график предсказанных результатов.

```
# Построение графика результатов
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred =
nn.sim(x_dense.reshape(x_dense.size,1)).reshape(x_dense.size)

plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Фактические и прогнозные значения')

plt.show()
```

Полный код примера содержится в файле `multilayer_neural_network.py`. В процессе выполнения этого кода на экране отобразятся три графика. На первом снимке экрана представлены входные данные (рис. 14.6).

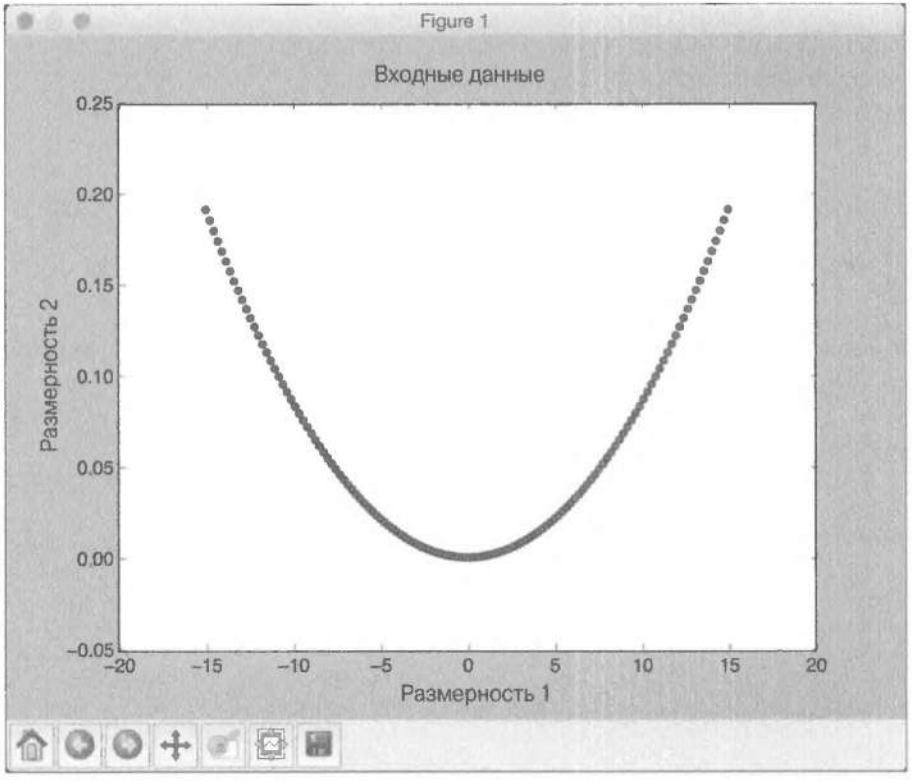

Рис. 14.6

На втором снимке экрана представлен график продвижения процесса обучения (рис. 14.7).

На третьем снимке экрана представлен график предсказанных результатов, наложенный на график входных данных (рис. 14.8).

Как видим, предсказанные результаты следуют общей тенденции. Если вы продолжите обучение сети и уменьшите ошибку, то увидите, что соответствие предсказанных результатов кривой исходных данных будет характеризоваться еще более высокой точностью.

В окне терминала отобразится следующая информация (рис. 14.9).

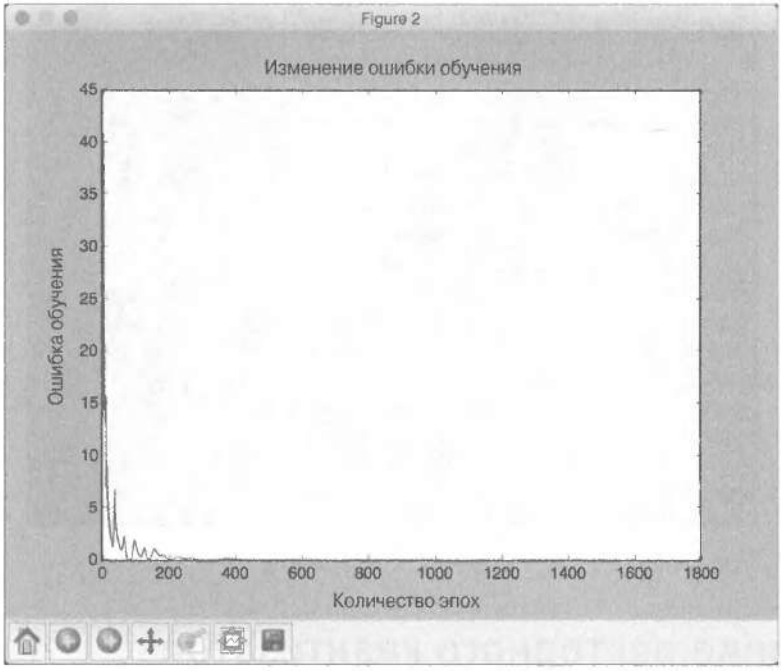

Рис. 14.7

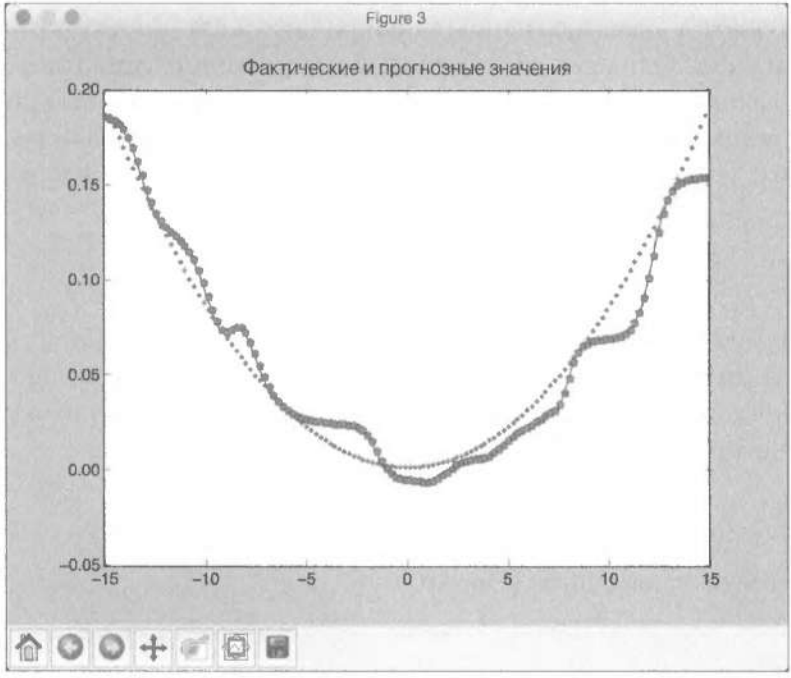

Рис. 14.8

```
Epoch: 100; Error: 1.9247718251621995;
Epoch: 200; Error: 0.15723294798079526;
Epoch: 300; Error: 0.021680213116912858;
Epoch: 400; Error: 0.1381761995539017;
Epoch: 500; Error: 0.04392553381948737;
Epoch: 600; Error: 0.02975401597014979;
Epoch: 700; Error: 0.014228560930227126;
Epoch: 800; Error: 0.03460207842970052;
Epoch: 900; Error: 0.035934053149433196;
Epoch: 1000; Error: 0.025833284445815966;
Epoch: 1100; Error: 0.013672412879982398;
Epoch: 1200; Error: 0.01776586425692384;
Epoch: 1300; Error: 0.04310242610384976;
Epoch: 1400; Error: 0.03799681296096611;
Epoch: 1500; Error: 0.02467030041520845;
Epoch: 1600; Error: 0.010094873168855236;
Epoch: 1700; Error: 0.01210866043021068;
The goal of learning is reached
```

Рис. 14.9

Создание векторного квантизатора

Векторная квантизация — это метод квантизации, в котором входные данные представляются фиксированным количеством представительных точек. Этот подход эквивалентен округлению чисел. Он широко применяется в ряде областей, таких как распознавание изображений, семантический анализ и наука о данных. Рассмотрим конкретный пример использования искусственных нейронных сетей для построения векторного квантизатора.

Создайте новый файл Python и импортируйте следующие файлы.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Загрузим входные данные из файла `data_vector_quantization.txt`. Каждая строка этого файла содержит шесть чисел. Первые два числа образуют точку данных, тогда как последние четыре — прямое кодирование метки. Всего имеются четыре класса данных.

```
# Загрузка входных данных
text = np.loadtxt('data_vector_quantization.txt')
```

Разделим текст на данные и метки.

```
# Разделение текста на данные и метки
data = text[:, 0:2]
labels = text[:, 2:]
```

Определим нейронную сеть с двумя слоями: десять нейронов во входном слое и четыре в выходном.

```
# Определение нейронной сети с двумя слоями:
# десять нейронов во входном слое и четыре в выходном
num_input_neurons = 10
num_output_neurons = 4
weights = [1/num_output_neurons] * num_output_neurons
nn = nl.net.newlvq(nl.tool.minmax(data), num_input_neurons, weights)
```

Обучим нейронную сеть, используя тренировочный набор данных.

```
# Обучение нейронной сети
_ = nn.train(data, labels, epochs=500, goal=-1)
```

Создадим сетку точек для визуализации кластеров.

```
# Создание входной сетки данных
xx, yy = np.meshgrid(np.arange(0, 10, 0.2),
        np.arange(0, 10, 0.2))
xx.shape = xx.size, 1
yy.shape = yy.size, 1
grid_xy = np.concatenate((xx, yy), axis=1)
```

Выполним вычисления над сеткой точек данных с помощью нейронной сети.

```
# Выполнение вычислений над входной сеткой данных
grid_eval = nn.sim(grid_xy)
```

Извлечем четыре класса.

```
# Определение четырех классов
class_1 = data[labels[:,0] == 1]
class_2 = data[labels[:,1] == 1]
class_3 = data[labels[:,2] == 1]
class_4 = data[labels[:,3] == 1]
```

Извлечем сетки, соответствующие этим четырем классам.

```
# Define X-Y grids for all the 4 classes
grid_1 = grid_xy[grid_eval[:,0] == 1]
grid_2 = grid_xy[grid_eval[:,1] == 1]
grid_3 = grid_xy[grid_eval[:,2] == 1]
grid_4 = grid_xy[grid_eval[:,3] == 1]
```

Построим график результирующих данных.

```
# Построение графика результирующих данных
plt.plot(class_1[:,0], class_1[:,1], 'ko',
         class_2[:,0], class_2[:,1], 'ko',
```

```
        class_3[:,0], class_3[:,1], 'ko',
        class_4[:,0], class_4[:,1], 'ko')
plt.plot(grid_1[:,0], grid_1[:,1], 'm.',
        grid_2[:,0], grid_2[:,1], 'bx',
        grid_3[:,0], grid_3[:,1], 'c^',
        grid_4[:,0], grid_4[:,1], 'y+')
plt.axis([0, 10, 0, 10])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Векторная квантизация')

plt.show()
```

Полный код примера содержится в файле `vector_quantizer.py`. В процессе выполнения этого кода на экране отобразится следующий график, представляющий входные точки данных и границы между кластерами (рис. 14.10).

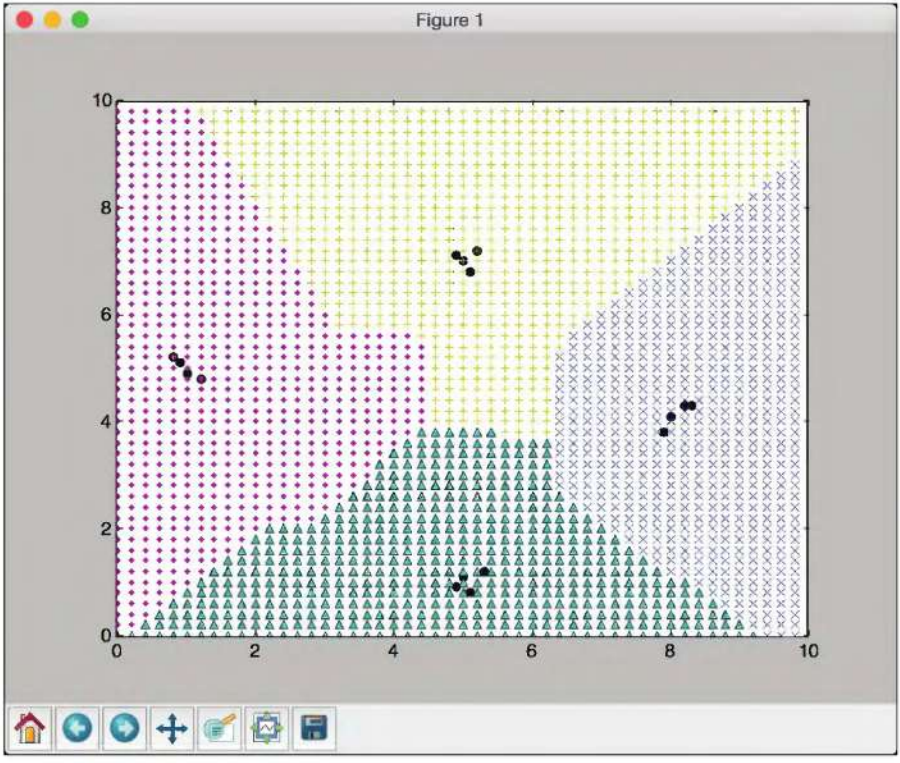

Рис. 14.10. (См. цветную вклейку; адрес указан во введении)

В окне терминала отобразится следующая информация (рис. 14.11).

```
Epoch: 100; Error: 0.0;
Epoch: 200; Error: 0.0;
Epoch: 300; Error: 0.0;
Epoch: 400; Error: 0.0;
Epoch: 500; Error: 0.0;
The maximum number of train epochs is reached
```

Рис. 14.11

Анализ последовательных данных с помощью рекуррентных нейронных сетей

До сих пор мы имели дело со статическими данными. Однако искусственные нейронные сети могут пригодиться и при построении моделей последовательных данных. В частности, рекуррентные нейронные сети отлично справляются с моделированием таких данных. Пожалуй, именно последовательные данные в виде временных рядов встречаются чаще всего в нашем мире. Больше о рекуррентных нейронных сетях можно узнать по следующему адресу:

```
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-
part-1-introduction-to-rnns
```

Работая с временными рядами данных, мы не можем просто использовать универсальные модели обучения. Для создания надежных моделей необходимо учитывать временные зависимости между данными. Давайте рассмотрим, как это делается.

Создайте новый файл Python и импортируйте следующие файлы.

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
```

Определим функцию, генерирующую синусоидальные сигналы. Начнем с определения четырех синусоидальных волн.

```
def get_data(num_points):
    # Создание синусоидальных волн
    wave_1 = 0.5 * np.sin(np.arange(0, num_points))
    wave_2 = 3.6 * np.sin(np.arange(0, num_points))
    wave_3 = 1.1 * np.sin(np.arange(0, num_points))
    wave_4 = 4.7 * np.sin(np.arange(0, num_points))
```

Создадим переменные амплитуды синусоидальных сигналов.

```
    # Создание переменных амплитуд
    amp_1 = np.ones(num_points)
    amp_2 = 2.1 + np.zeros(num_points)
    amp_3 = 3.2 * np.ones(num_points)
    amp_4 = 0.8 + np.zeros(num_points)
```

Создадим суммарный волновой сигнал.

```
    wave = np.array([wave_1, wave_2, wave_3,
                     wave_4]).reshape(num_points * 4, 1)
    amp = np.array([[amp_1, amp_2, amp_3,
                    amp_4]]).reshape(num_points * 4, 1)

    return wave, amp
```

Определим функцию, визуализирующую выходной сигнал нейронной сети.

```
# Визуализация выходного сигнала
def visualize_output(nn, num_points_test):
    wave, amp = get_data(num_points_test)
    output = nn.sim(wave)
    plt.plot(amp.reshape(num_points_test * 4))
    plt.plot(output.reshape(num_points_test * 4))
```

Определим основную функцию и создадим волновой сигнал.

```
if __name__=='__main__':
    # Создание выборочных данных
    num_points = 40
    wave, amp = get_data(num_points)
```

Создадим рекуррентную нейронную сеть с двумя слоями.

```
    # Создание рекуррентной нейронной сети с двумя слоями
    nn = nl.net.newelm([[-2, 2]], [10, 1], [nl.trans.TanSig(),
                       nl.trans.PureLin()])
```

Зададим функции инициализации для каждого слоя.

```
    # Задание функций инициализации для каждого слоя
    nn.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
    nn.layers[1].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
    nn.init()
```

Обучим нейронную сеть.

```
    # Обучение рекуррентной нейронной сети
    error_progress = nn.train(wave, amp, epochs=1200, show=100,
                              goal=0.01)
```

Прогоним данные через нейронную сеть.

```
# Прогонка тренировочных данных через сеть
output = nn.sim(wave)
```

Построим график результатов.

```
# Построение графика результатов
plt.subplot(211)
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка (MSE)')

plt.subplot(212)
plt.plot(amp.reshape(num_points * 4))
plt.plot(output.reshape(num_points * 4))
plt.legend(['Факт', 'Прогноз'])
```

Проверим работу нейронной сети, используя неизвестные тестовые данные.

```
# Тестирование нейронной сети на неизвестных данных
plt.figure()

plt.subplot(211)
visualize_output(nn, 82)
plt.xlim([0, 300])

plt.subplot(212)
visualize_output(nn, 49)
plt.xlim([0, 300])

plt.show()
```

Полный код примера содержится в файле `recurrent_neural_network.py`. В процессе выполнения этого кода на экране отобразятся два графика. В верхней части первого снимка экрана представлено продвижение процесса обучения, а в нижней — предсказанный результат, наложенный на суммарный входной сигнал (рис. 14.12).

В верхней части второго снимка экрана отображается результат имитации волнового сигнала нейронной сетью для увеличенной длины сигнала, а в нижней — для уменьшенной длины (рис. 14.13).

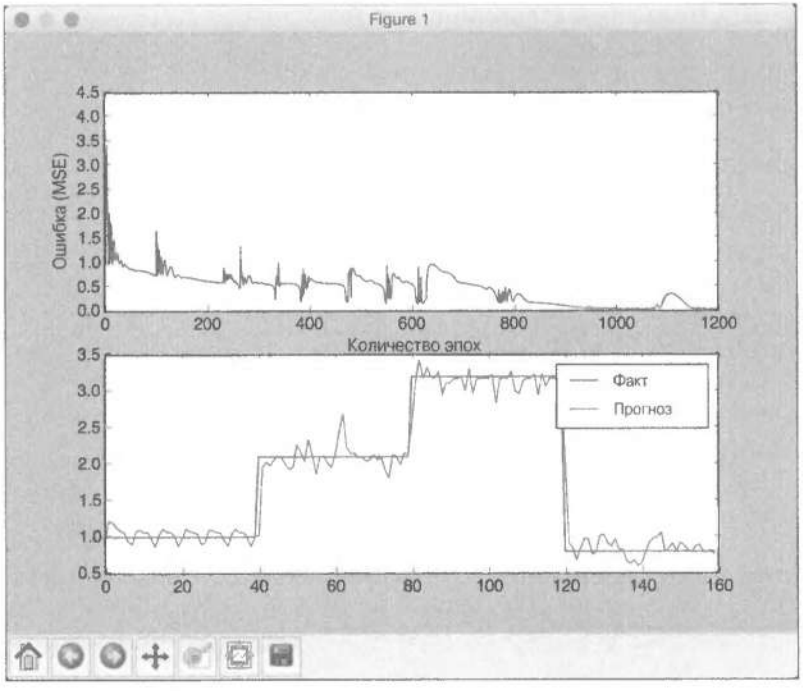

Рис. 14.12

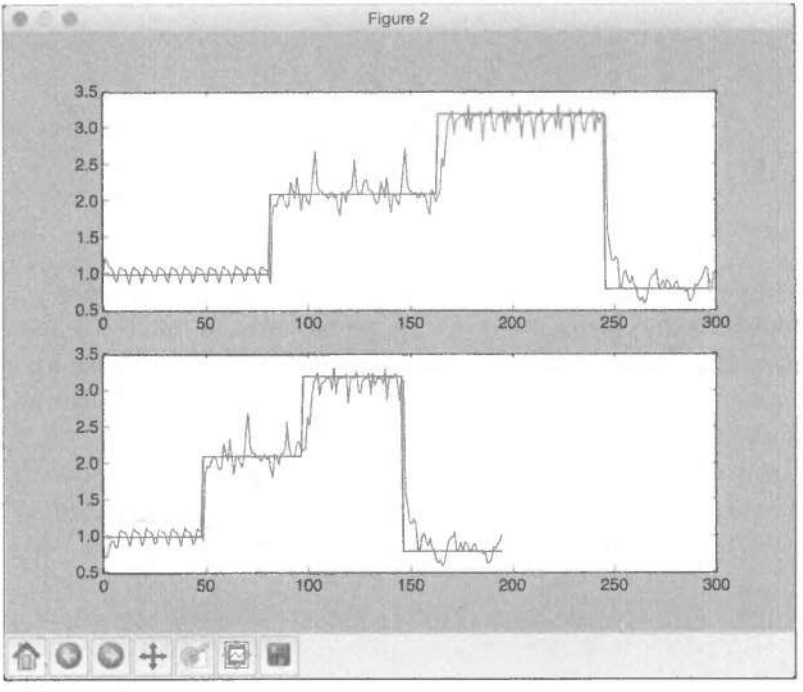

Рис. 14.13

В окне терминала отобразится следующая информация (рис. 14.14).

```
Epoch: 100; Error: 0.7378753203612153;
Epoch: 200; Error: 0.6276459886666788;
Epoch: 300; Error: 0.586316536629095;
Epoch: 400; Error: 0.7246461052491963;
Epoch: 500; Error: 0.7244266943409208;
Epoch: 600; Error: 0.5650581389122635;
Epoch: 700; Error: 0.5798180931911314;
Epoch: 800; Error: 0.19557566610789826;
Epoch: 900; Error: 0.10837074465396046;
Epoch: 1000; Error: 0.04330852391940663;
Epoch: 1100; Error: 0.3073835343028226;
Epoch: 1200; Error: 0.034685278416163604;
The maximum number of train epochs is reached
```

Рис. 14.14

Визуализация символов с использованием базы данных оптического распознавания символов

Искусственные нейронные сети могут задействовать оптическое распознавание символов. Возможно, это один из наиболее известных примеров его использования. **Оптическое распознавание символов** (Optical Character Recognition — OCR) — это процесс распознавания рукописных символов в изображениях. Прежде чем браться за создание этой модели, вам следует ознакомиться с соответствующим набором данных. Мы будем использовать набор данных, доступный по адресу `http://ai.stanford.edu/~btaskar/ocr`. Вы должны загрузить файл `letter.data`. Для удобства этот файл включен в состав примеров, прилагаемых к книге. Загрузим данные и визуализируем символы.

Создайте новый файл Python и импортируйте следующие файлы.

```
import os
import sys

import cv2
import numpy as np
```

Определим входной файл, содержащий OCR-данные.

```
# Определение входного файла
input_file = 'letter.data'
```

Определим параметры визуализации и другие параметры, необходимые для загрузки данных из этого файла.

```
# Определение параметров визуализации
img_resize_factor = 12
start = 6
end = -1
height, width = 16, 8
```

Выполним итерации по строкам файла до тех пор, пока пользователь не нажмет клавишу <Esc>. В этом файле строки разделены символами табуляции. Прочитаем каждую строку и умножим ее на 255.

```
# Итерирование до тех пор, пока пользователь не нажмет
# клавишу <Esc>
with open(input_file, 'r') as f:
    for line in f.readlines():
        # Чтение данных
        data = np.array([255 * float(x) for x in
                         line.split('\t')[start:end]])
```

Переформируем одномерный массив в двумерный.

```
        # Переформирование данных в двумерный массив
        img = np.reshape(data, (height, width))
```

Масштабируем изображение для визуализации.

```
        # Масштабирование изображения
        img_scaled = cv2.resize(img, None, fx=img_resize_factor,
                                fy=img_resize_factor)
```

Выведем изображение.

```
        # Вывод изображения
        cv2.imshow('Image', img_scaled)
```

Проверим, не нажал ли пользователь клавишу <Esc>. Если это действительно так, выйти из цикла.

```
        # Проверка того, не нажал ли пользователь клавишу <Esc>
        c = cv2.waitKey()
        if c == 27:
            break
```

Полный код примера содержится в файле `character_visualizer.py`. В процессе выполнения этого кода на экране отобразится окно с изображением символа. Нажимая клавишу пробела, вы сможете просматривать дополнительные символы. Вот так выглядит символ `o` (рис. 14.15).

А вот так выглядит символ `i` (рис. 14.16).

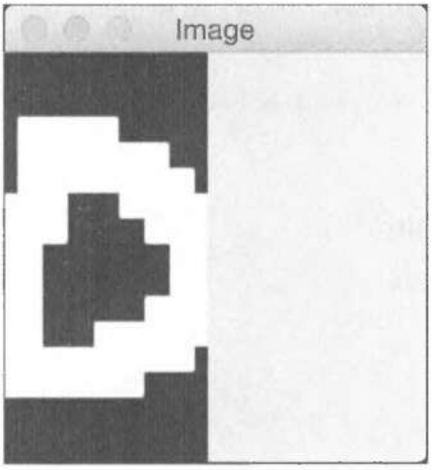

Рис. 14.15

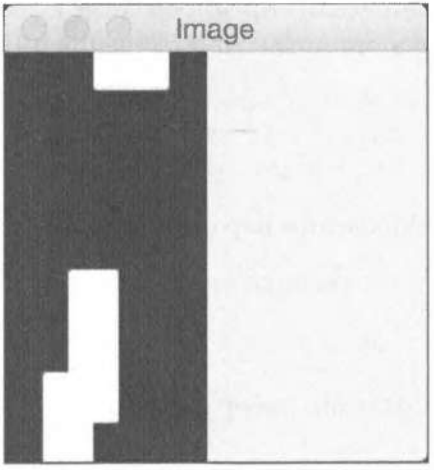

Рис. 14.16

Создание системы оптического распознавания символов

Теперь, когда вы уже знаете, как работать с данными такого типа, мы можем перейти к созданию системы оптического распознавания символов с помощью искусственной нейронной сети.

Создайте новый файл Python и импортируйте следующие файлы.

```
import numpy as np
import neurolab as nl
```

Определим входной файл.

```
# Определение входного файла
input_file = 'letter.data'
```

Определим количество загружаемых точек данных.

```
# Определим количество точек данных, подлежащих
# загрузке из входного файла
num_datapoints = 50
```

Определим строку, содержащую все различные символы.

```
# Строка, содержащая все различные символы
orig_labels = 'omandig'
```

Извлечем количество различных классов.

```
# Вычисление количества различных классов
num_orig_labels = len(orig_labels)
```

Определим тренировочные и тестовые данные. Мы будем использовать их в пропорции 90% для обучения и 10% для тестирования.

```
# Определение параметров тренировочных и тестовых данных
num_train = int(0.9 * num_datapoints)
num_test = num_datapoints - num_train
```

Определим параметры извлечения данных.

```
# Определение параметров извлечения данных
start = 6
end = -1
```

Создадим набор данных.

```
# Создание набора данных
data = []
labels = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        # Разбиение текущей строки по символам табуляции
        list_vals = line.split('\t')
```

Если метка отсутствует в нашем списке меток, пропустить ее.

```
        # Проверка того, присутствует ли данная метка в нашем
        # базовом списке меток. Если она отсутствует,
        # мы ее пропускаем.
        if list_vals[1] not in orig_labels:
            continue
```

Извлечем текущую метку и присоединим ее к основному списку.

```
        # Извлечение текущей метки и ее присоединение
        # к основному списку
        label = np.zeros((num_orig_labels, 1))
        label[orig_labels.index(list_vals[1])] = 1
        labels.append(label)
```

Извлечем символьный вектор и присоединим его к основному списку.

```
        # Извлечение символьного вектора и его присоединение
        # к основному списку.
        cur_char = np.array([float(x) for x in list_vals[start:end]])
        data.append(cur_char)
```

Создав набор данных, выходим из цикла.

```
        # Выход из цикла, как только создание требуемого
        # набора данных завершено
        if len(data) >= num_datapoints:
            break
```

Преобразуем списки в массивы numpy.

```
# Преобразование данных и меток в массивы NumPy
data = np.asfarray(data)
labels = np.array(labels).reshape(num_datapoints, num_orig_labels)
```

Извлечем количество измерений.

```
# Извлечение количества измерений
num_dims = len(data[0])
```

Создадим нейронную сеть и зададим метод градиентного спуска в качестве обучающего алгоритма.

```
# Создание нейронной сети
nn = nl.net.newff([[0, 1] for _ in range(len(data[0]))],
                  [128, 16, num_orig_labels])

# Задание градиентного спуска в качестве обучающего алгоритма
nn.trainf = nl.train.train_gd
```

Обучим нейронную сеть.

```
# Обучение сети
error_progress = nn.train(data[:num_train,:],
        labels[:num_train,:], epochs=10000, show=100, goal=0.01)
```

Предскажем результат для тестовых данных.

```
# Предсказание результата для тестовых данных
print('\nTesting on unknown data:')
predicted_test = nn.sim(data[num_train:, :])
for i in range(num_test):
    print('\nOriginal:', orig_labels[np.argmax(labels[i])])
    print('Predicted:', orig_labels[np.argmax(predicted_test[i])])
```

Полный код примера содержится в файле ocr.py. В процессе выполнения этого кода в окне терминала отобразится следующая информация (рис. 14.17).

```
Epoch: 100; Error: 80.75182001223291;
Epoch: 200; Error: 49.823887961230206;
Epoch: 300; Error: 26.624261963923217;
Epoch: 400; Error: 31.131906412329677;
Epoch: 500; Error: 30.589610928772494;
Epoch: 600; Error: 23.129959531324324;
Epoch: 700; Error: 15.561849160600984;
Epoch: 800; Error: 9.52433563455828;
Epoch: 900; Error: 1.4032941634688987;
Epoch: 1000; Error: 1.1584148924740179;
Epoch: 1100; Error: 0.844934060039839;
Epoch: 1200; Error: 0.646187646028962;
Epoch: 1300; Error: 0.48881681329304894;
Epoch: 1400; Error: 0.4005475591737743;
Epoch: 1500; Error: 0.34145887283532067;
Epoch: 1600; Error: 0.29871068426249625;
Epoch: 1700; Error: 0.2657577763744411;
Epoch: 1800; Error: 0.23921810237252988;
Epoch: 1900; Error: 0.2172060084455509;
Epoch: 2000; Error: 0.19856823374761018;
Epoch: 2100; Error: 0.18253521958793384;
Epoch: 2200; Error: 0.16855895648078095;
```

Рис. 14.17

Вычисления будут выполняться в течение 10000 эпох. Завершающая часть вывода будет выглядеть так (рис. 14.18).

Как видите, в трех случаях получены правильные ответы. Если вы используете более крупный набор данных и дольше потренируете сеть, то получите более высокую точность.

Резюме

Эта глава была посвящена искусственным нейронным сетям. В ней рассказывалось о том, как создать и обучить нейронную сеть, что такое перцептроны и как создать классификатор на их основе. Мы обсудили одно- и многослойные нейронные сети. Вы узнали о том, как использовать нейронные сети для создания векторного квантизатора. Мы проанализировали последовательные данные с помощью рекуррентных нейронных сетей. После этого мы создали систему оптического распознавания символов с помощью искусственной нейронной сети. В следующей главе вы узнаете об обучении с подкреплением и познакомитесь с методами создания интеллектуальных агентов обучения.

```
Epoch: 9500; Error: 0.032460181065798295;
Epoch: 9600; Error: 0.027044816600106478;
Epoch: 9700; Error: 0.022026328910164213;
Epoch: 9800; Error: 0.018353324233938713;
Epoch: 9900; Error: 0.01578969259136868;
Epoch: 10000; Error: 0.014064205770213847;
The maximum number of train epochs is reached

Testing on unknown data:

Original: o
Predicted: o

Original: m
Predicted: n

Original: m
Predicted: m

Original: a
Predicted: d

Original: n
Predicted: n
```

Рис. 14.18

15
Обучение с подкреплением

Эта глава посвящена обучению с подкреплением. Мы рассмотрим различия между этим видом обучения и обучением с учителем. Будут показаны реальные примеры обучения с подкреплением, вы узнаете, как оно применяется, и познакомитесь с соответствующими понятиями. Чтобы продемонстрировать, как все это работает, мы создадим специальную среду в Python. Изложенные концепции будут использованы далее для создания агента обучения.

К концу главы вы освоите следующие темы:

- основы обучения с подкреплением;
- обучение с подкреплением и обучение с учителем;
- реальные примеры обучения с подкреплением;
- строительные блоки обучения с подкреплением;
- создание программной среды;
- создание агента обучения.

Основы обучения с подкреплением

Концепция обучения имеет фундаментальное значение для искусственного интеллекта. Мы хотим, чтобы машины понимали процесс обучения и тем самым могли самостоятельно действовать в этом направлении. Люди учатся посредством наблюдения и взаимодействия с окружающей их средой. Оказавшись в незнакомом месте, вы быстро осматриваетесь и замечаете все, что происходит вокруг вас. Никто вас не учит тому, что вам следует сделать. Вы просто наблюдаете за окружением и взаимодействуете с ним. Устанавливая связь с окружением, мы стараемся собрать как можно больше информации о причинах и следствиях всевозможных событий, фиксируем результаты различных действий и учимся тому, какие действия необходимо предпринимать для получения того или иного результата.

Все это повсеместно происходит в нашей жизни. Мы накапливаем знания об окружении и учимся реагировать на различные события. Рассмотрим еще один пример, в котором центральной фигурой является оратор. Всякий раз, когда хорошие ораторы выступают на публике, они отдают себе отчет в том, как публика будет реагировать на каждое их слово. Если аудитория никак не реагирует, оратор немедленно перестраивает свою речь таким образом, чтобы она заинтересовала слушателей. Как видим, оратор пытается повлиять на окружение, изменяя свое поведение. Можно сказать, что оратор обучается на результатах своего взаимодействия с аудиторией для того, чтобы предпринимать действия, способные обеспечить достижение определенной цели. Это одна из наиболее фундаментальных идей в области искусственного интеллекта, на использовании которой основаны многие разработки. Говоря об обучении с подкреплением, мы должны всегда помнить об этом.

Обучение с подкреплением — это процесс обучения тому, что следует делать, и привязка ситуаций к определенным действиям с целью максимизации выгоды. В большинстве парадигм машинного обучения агенту обучения сообщается, какие действия должны быть предприняты для достижения определенных результатов. В случае обучения с подкреплением агент не получает никаких сообщений об этом. Вместо этого он должен самостоятельно обнаружить, какие действия принесут наибольшую выгоду, испытывая их поочередно. Обычно его действия влияют как на непосредственную выгоду, так и на последующую ситуацию. Это означает, что последствия действий будут сказываться также на результирующей выгоде всех остальных действий.

Чтобы осознать суть обучения с подкреплением, вы должны понять, что в данном случае мы определяем не метод обучения, а саму проблему обучения. Поэтому можно сказать, что любой метод обучения, позволяющий решить нашу задачу, является методом обучения с подкреплением. Обучение с подкреплением имеет две особенности: использование метода проб и ошибок и отложенная выгода. Агент обучения использует эти две особенности для того, чтобы учиться на последствиях своих действий.

Обучение с подкреплением и обучение с учителем

В настоящее время на обучении с подкреплением сфокусированы многочисленные исследования. Может показаться, что оно немного напоминает обучение с учителем, но это не так. Процесс обучения с учителем — это обучение на помеченных данных, предоставляемых нами. Несмотря на всю полезность этой методики, ее недостаточно для того, чтобы начать обучение на результатах взаимодействия с окружением. Если мы хотим спроектировать

машину, которая могла бы уверенно перемещаться на неразведанной территории, то этот вид обучения ничем не сможет нам помочь, ведь мы не располагаем никакими заранее известными тренировочными данными. Нам нужен агент, который мог бы учиться на собственном опыте взаимодействия с неизвестной территорией. Реальную помощь в подобных условиях может оказать обучение с подкреплением.

Рассмотрим исследовательскую часть, когда агент должен взаимодействовать с новым окружением для того, чтобы обучаться. Как много он может исследовать? Мы даже не знаем, насколько велико окружение, и в большинстве случаев невозможно исследовать все вероятные действия. Так как же должен действовать агент? Должен ли он учиться на ограниченном опыте или ждать, пока не исследует дополнительные возможности, прежде чем что-то предпринимать? Это одна из основных проблем обучения с подкреплением. В интересах получения наибольшей выгоды агент должен отдавать предпочтение действиям, которые уже были испытаны и результат которых известен. Но для того чтобы обнаружить наиболее оптимальные действия, он должен продолжить испытание новых действий, которые еще не были выбраны. На протяжении многих лет исследователи интенсивно изучали вопросы достижения компромисса между исследованием и использованием известных возможностей, и эта тема все еще не потеряла своей актуальности.

Реальные примеры обучения с подкреплением

Обратимся к ситуациям, в которых с обучением с подкреплением можно столкнуться в реальной жизни. Это поможет вам понять, как оно работает и какие приложения могут быть созданы на основе использования этой концепции.

- **Игры.** Возьмем настольную игру, например го или шахматы. Чтобы определить наилучший ход, игроки должны учесть множество факторов. Количество возможных ходов настолько велико, что выполнить их перебор методом грубой силы просто невозможно. Если бы мы создавали машину для такой игры с использованием традиционных методик, то должны были бы задать большое количество правил для охвата всех возможностей. Обучение с подкреплением полностью обходит эту проблему. Мы не должны вручную определять какие-либо правила. Агент обучения учится самостоятельно в процессе игры.
- **Робототехника.** Рассмотрим робота, задачей которого является обследование незнакомого здания. Он должен убедиться в том, что ему

хватит заряда батарей для возврата на базу. Робот должен самостоятельно сделать выбор относительно того, должен ли он принимать решения, достигая компромисса между объемом собранной информации и возможностью вернуться на базу.

- **Промышленные контроллеры.** Рассмотрим случай планирования движения лифтов. Хороший планировщик затратит наименьшее количество энергии и обслужит максимальное количество людей. В задачах подобного рода агенты обучения с подкреплением могут научиться тому, как это обеспечить, в имитированном окружении. Затем они могут применить приобретенные знания для того, чтобы обеспечить оптимальный режим планирования.
- **Развитие детей.** Для того чтобы научиться ходить, новорожденным требуется несколько месяцев. Они приобретают это умение благодаря многократным попыткам, пока не научатся держать баланс.

Во всех этих примерах можно заметить кое-что общее: все они включают взаимодействие с окружением. Агент обучения стремится достигнуть определенной цели даже в условиях неопределенности в отношении окружения. Действия агента изменят будущее состояние окружения. Это влияет на круг возможностей, доступных в более поздние моменты времени, по мере того как агент обучения взаимодействует с данным окружением.

Строительные блоки обучения с подкреплением

После того как вы ознакомились с рядом примеров, мы можем углубиться в рассмотрение строительных блоков систем обучения с подкреплением. Помимо взаимодействия агента с окружением следует учитывать также другие факторы (рис. 15.1).

Типичный агент обучения с подкреплением проходит через следующие этапы.

- Существует набор состояний, связанных с агентом и средой. В заданный момент времени агент проверяет входное состояние для получения сведений относительно окружения.
- Существуют стратегии принятия решений относительно того, какие действия должны предприниматься. Эти стратегии играют роль функций принятия решений. Нужное действие определяется на основании информации о входном состоянии с использованием этих стратегий.
- Агент предпринимает действие на основании предыдущего этапа.

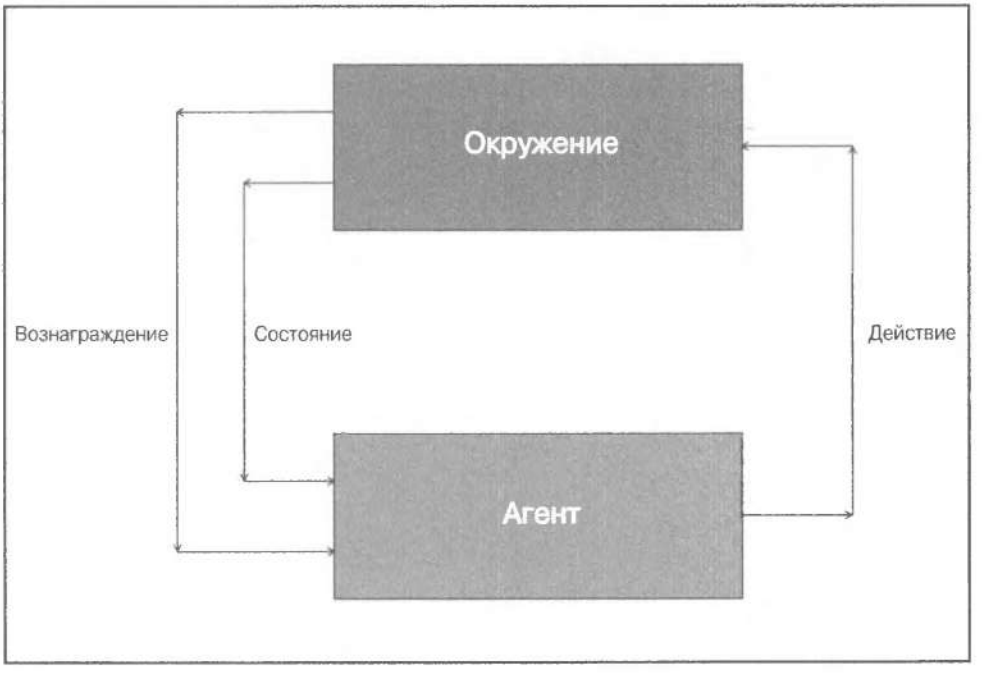

Рис. 15.1

- В ответ на это действие окружение реагирует определенным образом. Агент получает *подкрепление*, также называемое *вознаграждением*, от окружения.
- Агент записывает информацию об этом вознаграждении. Важно отметить, что вознаграждение соответствует конкретной паре "состояние–действие".

Системы обучения с подкреплением могут выполнять одновременно несколько вещей: обучаться, осуществляя поиск методом проб и ошибок, изучать модель окружения, в котором они действуют, а затем использовать эту модель для планирования следующих шагов.

Создание окружения

Для создания агентов обучения с подкреплением мы будем использовать пакет OpenAI Gym. Подробнее об этом пакете можно узнать на сайте `https://gym.openai.com`. Вы сможете установить его с помощью утилиты `pip`, выполнив следующую команду в окне терминала:

```
$ pip3 install gym
```

По адресу https://github.com/openai/gym#installation приведены различные советы и рекомендации касательно инсталляции. Установив пакет, мы можем пойти дальше и приступить к написанию кода.

Создадим новый файл Python и импортируем следующие пакеты.

```
import argparse
import gym
```

Определим функцию для анализа входных аргументов. Мы будем иметь возможность указать тип окружения, которое хотим использовать.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Run an environment')
    parser.add_argument('--input-env', dest='input_env',
            required=True, choices=['cartpole', 'mountaincar',
            'pendulum', 'taxi', 'lake'],
            help='Specify the name of the environment')
    return parser
```

Определим основную функцию и проанализируем входные аргументы.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    input_env = args.input_env
```

Создадим отображение входных аргументов на имена окружений, определенных в пакете OpenAI Gym.

```
    name_map = {'cartpole': 'CartPole-v0',
                'mountaincar': 'MountainCar-v0',
                'pendulum': 'Pendulum-v0',
                'taxi': 'Taxi-v1',
                'lake': 'FrozenLake-v0'}
```

Создадим окружение на основании входного аргумента и сбросим его состояние.

```
    # Создание окружения и сброс его состояния
    env = gym.make(name_map[input_env])
    env.reset()
```

Итерируем 1000 раз, предпринимая действие на каждом шаге.

```
    # Итерирование 1000 раз
    for _ in range(1000):
        # Визуализация окружения
        env.render()
        # Выполнение случайного действия
        env.step(env.action_space.sample())
```

Полный код примера содержится в файле `run_environment.py`. Если хотите узнать, как запускать этот код, выполните его, указав `help` в качестве аргумента командной строки (рис. 15.2).

```
$ python3 run_environment.py --help
usage: run_environment.py [-h] --input-env
                          {cartpole,mountaincar,pendulum,taxi,lake}

Run an environment

optional arguments:
  -h, --help            show this help message and exit
  --input-env {cartpole,mountaincar,pendulum,taxi,lake}
                        Specify the name of the environment
```

Рис. 15.2

Запустим этот код с аргументом `cartpole` (обратный маятник). Выполните следующую команду в окне терминала:

```
$ python3 run_environment.py --input-env cartpole
```

После запуска программы откроется окно, в котором отображается обратный маятник, движущийся вправо относительно вас. Его начальная позиция показана на рис. 15.3.

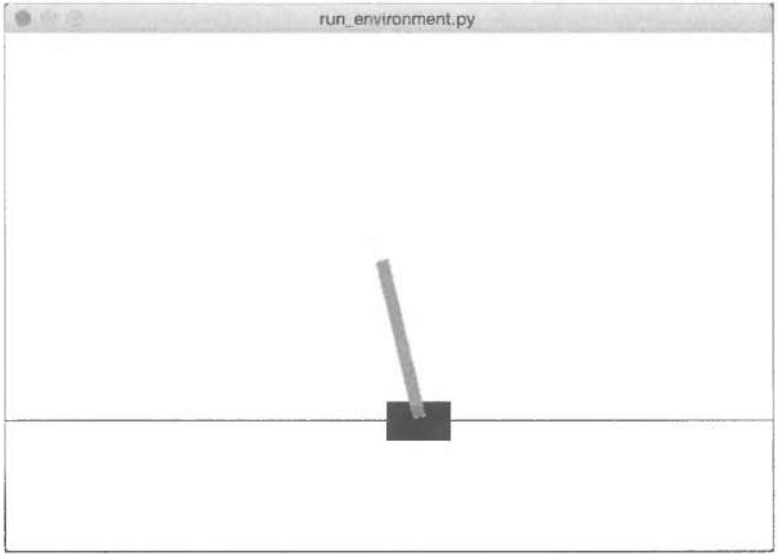

Рис. 15.3

Примерно через секунду вы увидите его движущимся так, как показано на рис. 15.4.

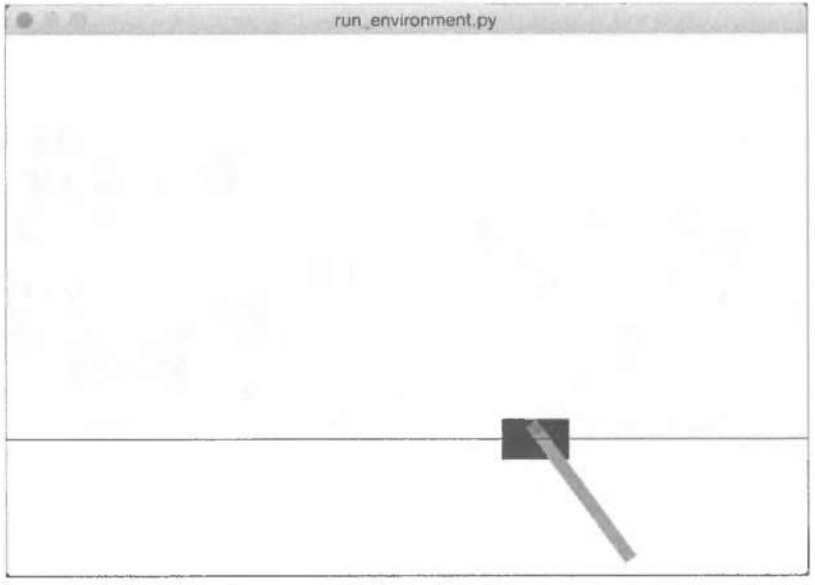

Рис. 15.4

Ближе к концу выполнения программы вы увидите маятник выходящим за пределы окна (рис. 15.5).

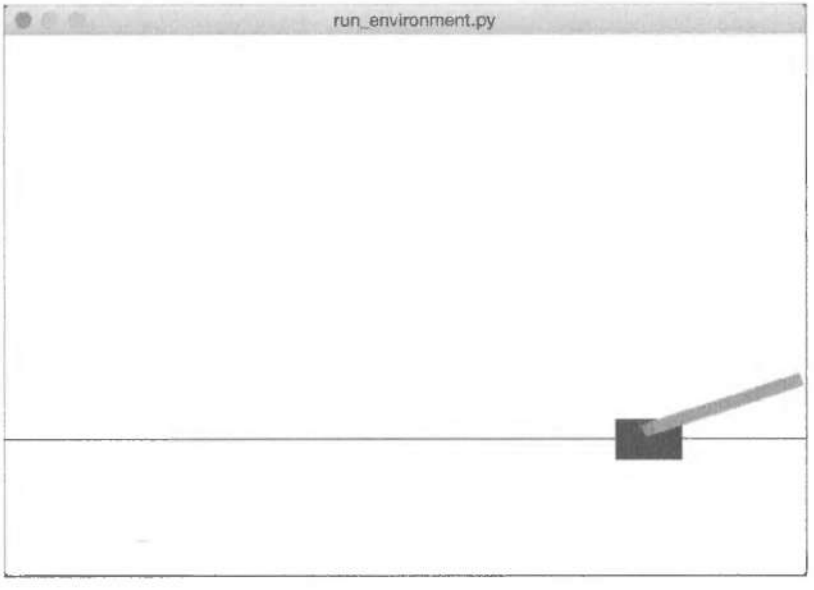

Рис. 15.5

Запустим этот код с аргументом `mountain car` (горный автомобиль). Выполните следующую команду в окне терминала:

```
$ python3 run_environment.py --input-env mountaincar
```

Первоначальная картинка будет выглядеть так (рис. 15.6).

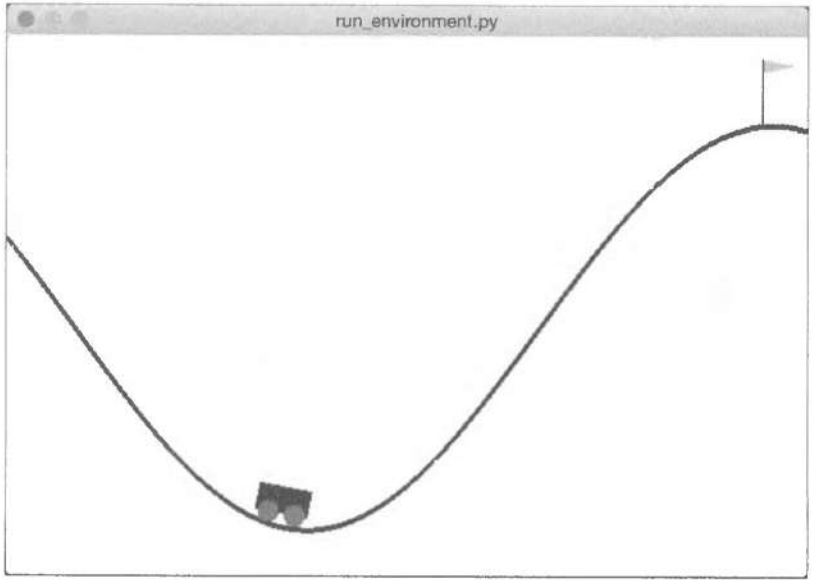

Рис. 15.6

Выждав несколько секунд, пока код выполняется, вы увидите, что автомобиль раскачивается все больше, стремясь достигнуть флажка (рис. 15.7).

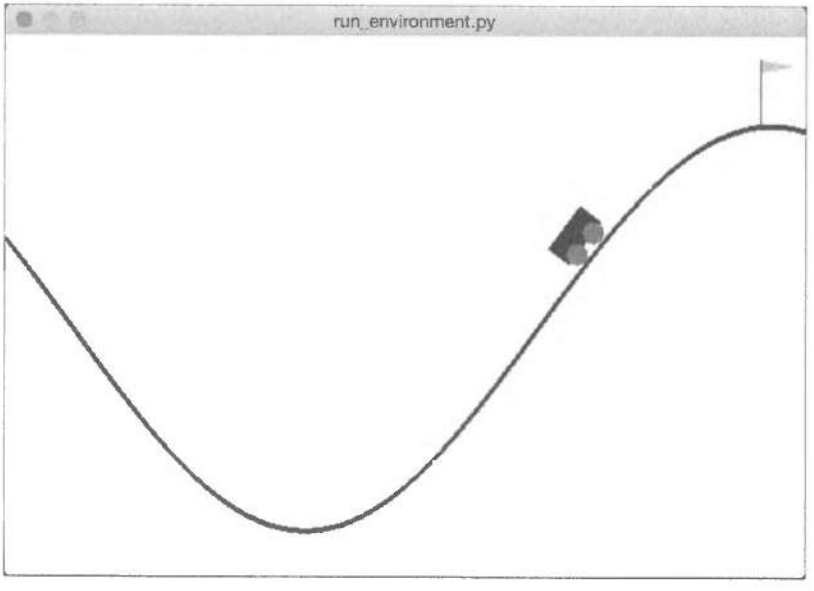

Рис. 15.7

Автомобиль будет продолжать попытки разогнаться, как показано на рис. 15.8.

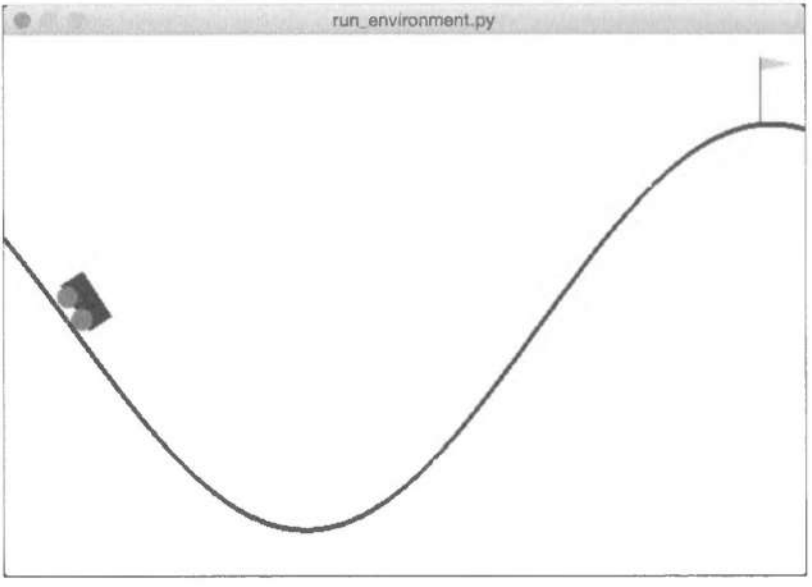

Рис. 15.8

Создание агента обучения

Приступим к созданию агента обучения, способного достигать цели. Он будет этому учиться. Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse
import gym
```

Определим функцию для анализа входных аргументов.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Run an environment')
    parser.add_argument('--input-env', dest='input_env',
            required=True,
            choices=['cartpole', 'mountaincar', 'pendulum'],
            help='Specify the name of the environment')
    return parser
```

Проанализируем входные аргументы.

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    input_env = args.input_env
```

Создадим отображение входных аргументов на имена окружений, определенных в пакете OpenAI Gym.

```
name_map = {'cartpole': 'CartPole-v0',
            'mountaincar': 'MountainCar-v0',
            'pendulum': 'Pendulum-v0'}
```

Создадим окружение на основании входного аргумента.

```
# Создание окружения
env = gym.make(name_map[input_env])
```

Начнем итерации, сбросив состояние окружения.

```
# Запуск итерирования
for _ in range(20):
    # Сброс состояния окружения
    observation = env.reset()
```

После каждого сброса состояния итерируем 100 раз. Начнем с визуализации окружения.

```
    # Итерирование 100 раз
    for i in range(100):
        # Визуализация окружения
        env.render()
```

Выведем результат текущего наблюдения и предпримем действие, исходя из доступного для него пространства.

```
        # Вывод текущего наблюдения
        print(observation)

        # Совершение действия
        action = env.action_space.sample()
```

Извлечем результаты выполнения текущего действия.

```
        # Извлечение наблюдения, вознаграждения, состояния
        # и другой информации, полученной в результате
        # выполнения данного действия
        observation, reward, done, info = env.step(action)
```

Проверим, достигнута ли цель.

```
        # Проверка достижения цели
        if done:
            print('Episode finished after {}
                  timesteps'.format(i+1))
        break
```

Полный код примера содержится в файле `balancer.py`. Если хотите узнать, как запускать этот код, выполните его, указав `help` в качестве аргумента командной строки (рис. 15.9).

```
$ python3 balancer.py --help
usage: balancer.py [-h] --input-env {cartpole,mountaincar,pendulum}

Run an environment

optional arguments:
  -h, --help            show this help message and exit
  --input-env {cartpole,mountaincar,pendulum}
                        Specify the name of the environment
```

Рис. 15.9

Запустим этот код с аргументом `cartpole` (обратный маятник). Выполните следующую команду в окне терминала:

```
$ python3 balancer.py --input-env cartpole
```

После запуска программы откроется окно, в котором отображается балансирующий обратный маятник (рис. 15.10).

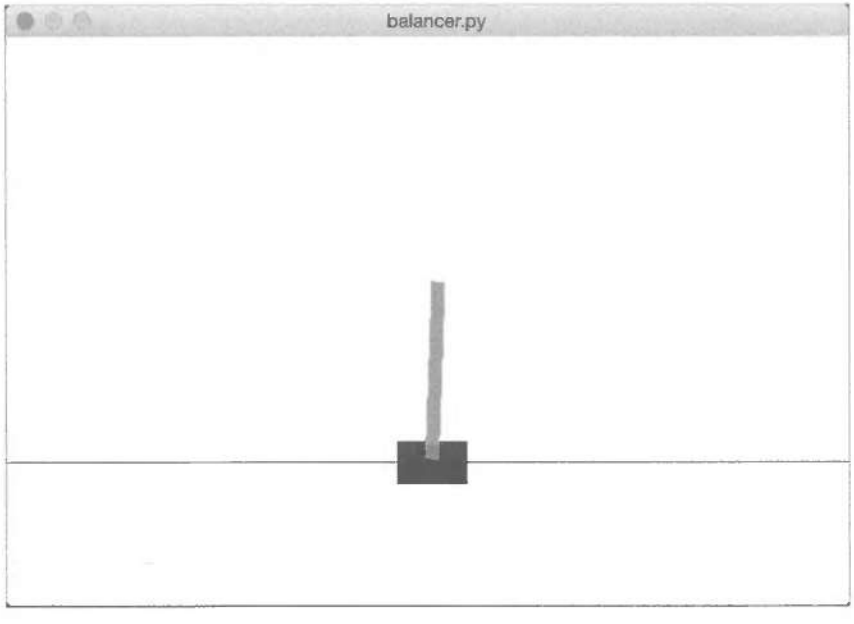

Рис. 15.10

Выждав несколько секунд, пока выполняется код, вы увидите, что маятник по-прежнему сохраняет устойчивую позицию (рис. 15.11).

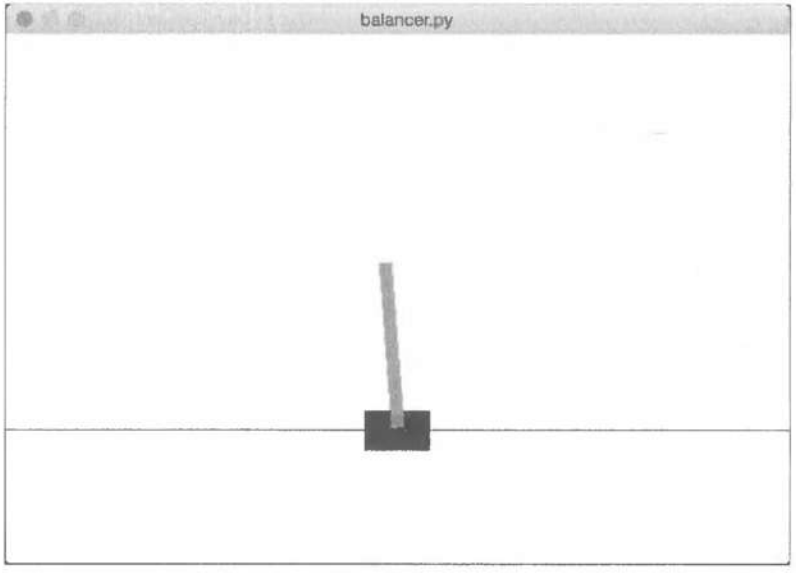

Рис. 15.11

В окне терминала отобразится большой объем информации. Ее примерный вид для одного из эпизодов приведен на рис. 15.12.

```
[ 0.01704777  0.03379922 -0.01628054  0.02868271]
[ 0.01772375 -0.16108552 -0.01570689  0.31618481]
[ 0.01450204  0.03425659 -0.00938319  0.01859014]
[ 0.01518717 -0.16072954 -0.00901139  0.30829785]
[ 0.01197258 -0.35572194 -0.00284543  0.59812526]
[ 0.00485814 -0.16056029  0.00911707  0.30454742]
[ 0.00164694 -0.35581098  0.01520802  0.60009165]
[-0.00546928 -0.16090505  0.02720986  0.31223756]
[-0.00868738 -0.35640386  0.03345461  0.61337594]
[-0.01581546 -0.55197696  0.04572213  0.91640525]
[-0.026855   -0.3575021   0.06405023  0.63843544]
[-0.03400504 -0.16332896  0.07681894  0.36659087]
[-0.03727162 -0.3594537   0.08415076  0.68247294]
[-0.04446069 -0.5556372   0.09780022  1.00041801]
[-0.05557344 -0.75192055  0.11780858  1.32214352]
[-0.07061185 -0.55846765  0.14425145  1.06853119]
[-0.0817812  -0.36551752  0.16562207  0.82437502]
[-0.08909155 -0.56247052  0.18210957  1.16423244]
[-0.10034096 -0.75943464  0.20539422  1.50803784]
Episode finished after 19 timesteps
```

Рис. 15.12

Для завершения других эпизодов потребуется другое количество шагов. Чтобы убедиться в этом, прокрутите содержимое окна терминала.

Резюме

Эта глава была посвящена системам обучения с подкреплением. Мы рассмотрели основы обучения с подкреплением и показали, как его можно настроить. Были проанализированы различия между обучением с подкреплением и обучением с учителем. Также были приведены примеры обучения с подкреплением, взятые из реальной жизни.

Мы обсудили строительные блоки систем обучения с подкреплением и такие сопутствующие понятия, как агент обучения, окружение, стратегия обучения, вознаграждение и пр. Все это было использовано нами для создания агента обучения с подкреплением.

16
Глубокое обучение и сверточные нейронные сети

Эта глава посвящена **глубокому обучению и сверточным нейронным сетям (CNN)**. За последние несколько лет CNN привлекли к себе огромное внимание, особенно в области распознавания изображений. Мы обсудим архитектуру CNN и типы их внутренних слоев. Также будет продемонстрировано использование пакета TensorFlow. Мы создадим линейный регрессор на основе перцептрона. Кроме того, будет показано, как создать классификатор изображений с помощью однослойной нейронной сети. После этого мы создадим классификатор изображений, используя CNN.

К концу главы вы освоите следующие темы:

- что такое сверточные нейронные сети (CNN);
- архитектура CNN;
- типы слоев CNN;
- создание линейного регрессора на основе перцептрона;
- создание классификатора изображений с использованием однослойной нейронной сети;
- создание классификатора изображений с использованием сверточной нейронной сети.

Что такое сверточные нейронные сети

В последних двух главах было продемонстрировано, как работают нейронные сети. Нейронные сети состоят из нейронов, характеризующихся весами и смещениями. Эти веса и смещения подстраиваются в процессе улучшения модели обучения. Каждый нейрон получает набор входных данных

(сигналов), подвергает их некоторой обработке и вырабатывает выходные данные (сигналы).

Нейронная сеть, состоящая из множества слоев, называется *глубокой нейронной сетью*. Направление исследований в области искусственного интеллекта, занимающееся глубокими нейронными сетями, называется *глубоким обучением*.

Одним из главных недостатков обычных нейронных сетей является то, что они игнорируют структуру входных данных. Прежде чем входные данные передаются сети, все они преобразуются в одномерный массив. Такой подход приемлем для обычных данных, но в случае изображений ситуация усложняется.

Рассмотрим изображения в градациях серого. Эти изображения являются 2D-структурами, и мы знаем, что пространственное расположение пикселей несет в себе массу скрытой информации. Проигнорировав эту информацию, мы потеряем множество базовых закономерностей (шаблонов). И здесь на выручку приходят **сверточные нейронные сети (CNN)**. При обработке изображений CNN учитывают их 2D-структуру.

CNN также состоят из нейронов, имеющих веса и смещения. Эти нейроны воспринимают данные, обрабатывают их и выдают выходные данные. Задача нейронной сети заключается в обеспечении перехода от необработанных данных во входном слое к корректному классу в выходном слое. CNN отличаются от обычных нейронных сетей типом используемых слоев и способом обработки входных данных. Предполагается, что входные данные являются изображениями, что позволяет извлекать специфические только для них свойства. Это делает обработку изображений с помощью CNN более эффективной.

Архитектура CNN

Работая с обычными нейронными сетями, мы должны преобразовывать входные данные в одиночный вектор. Он выступает в качестве входного сигнала нейронной сети, который проходит через ее слои. В этих слоях каждый нейрон соединен со всеми нейронами предыдущего слоя. Следует также отметить, что нейроны внутри любого слоя не соединяются между собой. Они связаны лишь с нейронами соседних слоев. Последний слой сети является выходным и представляет окончательный выходной сигнал.

Если мы попытаемся применить эту структуру к изображениям, то очень быстро потеряем любую возможность работать с ней. В качестве примера рассмотрим набор данных, состоящий из RGB-изображений размером 256×256. Поскольку эти изображения являются трехканальными, каждое из них имеет $256 * 256 * 3 = 196608$ весов, и это только для одного нейрона!

Каждый слой содержит не один нейрон, поэтому с увеличением количества нейронов количество весов будет очень быстро возрастать. Это означает, что модель будет обладать огромным количеством параметров, подлежащих настройке в процессе обучения, что значительно усложняет выполнение необходимых вычислений, требующих больших затрат времени. Соединение каждого нейрона одного слоя с каждым из нейронов предыдущего слоя, называемое *полносвязностью*, явно нас не устраивает.

При обработке данных структура изображений учитывается CNN явным образом. Нейроны в CNN организованы в трех измерениях: ширина, высота и глубина. Каждый нейрон текущего слоя соединен с небольшим фрагментом выхода предыдущего слоя, что можно уподобить применению к входному изображению фильтра размером $N \times N$. Это контрастирует с полносвязным слоем, в котором каждый нейрон соединен со всеми нейронами предыдущего слоя.

Поскольку одиночный фильтр не в состоянии уловить все нюансы изображения, мы делаем это M раз для уверенности в том, что все детали будут схвачены. Эти M фильтров действуют в качестве экстракторов признаков. Если вы посмотрите на выходные результаты фильтров, то увидите, что они извлекают такие признаки, как ребра, углы и т.п. Сказанное относится к начальным слоям CNN. По мере углубления в сеть последующие слои извлекают все более высокоуровневые признаки.

Типы слоев CNN

Теперь, когда вы познакомились с архитектурой CNN, мы можем рассмотреть типы слоев, используемых для построения этих сетей. Обычно используются следующие типы слоев.

- **Входной слой.** Этот слой принимает необработанные данные изображений в том виде, в каком они есть.
- **Сверточный слой.** Этот слой вычисляет свертки между нейронами и различными фрагментами входного изображения. Если вам необходимо освежить свои знания о свертке изображений, воспользуйтесь следующей ссылкой:

  `http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Ludwig_ImageConvolution.pdf`

  Сверточный слой в основном вычисляет точечное (скалярное) произведение весов и небольших фрагментов выходных данных предыдущего слоя.

- **Слой скорректированных линейных блоков (Rectified Linear Unit — ReLU).** Этот слой применяет функцию активации к выходному сигналу предыдущего слоя. Обычно используют функции наподобие *max(0, x)*. Этот слой необходим для добавления нелинейности в сеть, чтобы она допускала обобщение на любой тип функций.
- **Объединяющий слой.** Этот слой семплирует выходные данные предыдущего слоя для получения структуры меньшей размерности. Такое объединение способствует сохранению лишь наиболее существенных частей информации по мере прохождения сети. Для этой цели часто используют объединение типа Max, которому соответствует выбор максимального значения в данном окне размером *K*×*K*.
- **Полносвязный слой.** Этот слой вычисляет выходные оценки в последнем слое. Результирующий выходной сигнал имеет размеры *1x1xL*, где *L* — количество классов в тренировочном наборе данных.

По мере удаления от входного слоя и приближения к выходному слою входное изображение трансформируется из пиксельных значений в окончательные оценки классов. Были предложены различные архитектуры CNN, и активные исследования в этой области продолжаются. Точность и надежность модели зависят от многих факторов: типа слоев, глубины сети, взаимного расположения слоев различного типа внутри сети, выбора функций активации для каждого слоя, тренировочных данных и т.п.

Создание линейного регрессора на основе перцептрона

В этом разделе будет показано, как создать линейную регрессионную модель на основе перцептронов. Вы уже сталкивались с линейной регрессией в предыдущих главах, но далее речь пойдет о создании линейной регрессионной модели с использованием нейронных сетей.

Мы будем использовать библиотеку TensorFlow. Это популярный набор средств глубокого обучения, который широко применяется для построения различных прикладных систем. Прежде чем продолжить чтение, установите TensorFlow. Инструкции по инсталляции приведены по адресу `https://www.tensorflow.org/get_started/os_setup`. Убедившись в том, что библиотека успешно установлена, создайте новый файл Python и импортируйте следующие пакеты.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

Мы будем генерировать определенные точки данных и смотреть, насколько нам удается подогнать под них модель. Определим количество генерируемых данных.

```
# Определение количества генерируемых точек
num_points = 1200
```

Определим параметры, которые будут применяться для генерации данных. Мы будем использовать модель прямой линии: $y = mx + c$.

```
# Генерация данных на основе уравнения y = mx + c
data = []
m = 0.2
c = 0.5
for i in range(num_points):
    # Генерация 'x'
    x = np.random.normal(0.0, 0.8)
```

Сгенерируем шум, варьирующий данные.

```
    # Генерация шума
    noise = np.random.normal(0.0, 0.04)
```

Вычислим значение y с помощью уравнения.

```
    # Вычисление 'y'
    y = m*x + c + noise
    data.append([x, y])
```

Завершив итерирование, разделим данные на входные и выходные переменные.

```
# Разделение 'x' и 'y'
x_data = [d[0] for d in data]
y_data = [d[1] for d in data]
```

Построим график данных.

```
# Построение графика сгенерированных данных
plt.plot(x_data, y_data, 'ro')
plt.title('Входные данные')
plt.show()
```

Сгенерируем веса и смещения для перцептрона. Для весов мы используем генератор случайных чисел с равномерным законом распределения, а смещения зададим равными нулю.

```
# Генерация весов и смещений
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
```

Определим уравнения, используя переменные TensorFlow.

```
# Определение уравнения для 'y'
y = W * x_data + b
```

Определим функцию потерь, которую можно будет использовать в процессе обучения. Оптимизатор будет пытаться минимизировать ее значение.

```
# Определение способа вычисления потерь
loss = tf.reduce_mean(tf.square(y - y_data))
```

Определим оптимизатор, использующий метод градиентного спуска, и передадим ему функцию потерь.

```
# Определение оптимизатора на основе метода градиентного спуска
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

Все переменные созданы, но пока что мы их не инициализировали. Сделаем это.

```
# Инициализация всех переменных
init = tf.initialize_all_variables()
```

Начнем сеанс работы с TensorFlow, запустив его с помощью инициализатора.

```
# Начало сеанса работы с TensorFlow и его запуск
sess = tf.Session()
sess.run(init)
```

Начнем процесс обучения.

```
# Начало итерирования
num_iterations = 10
for step in range(num_iterations):
    # Запуск сеанса
    sess.run(train)
```

Выведем данные о продвижении процесса обучения. По мере увеличения количества выполненных итераций параметр потерь непрерывно снижается.

```
    # Вывод информации о продвижении процесса обучения
    print('\nITERATION', step+1)
    print('W =', sess.run(W)[0])
    print('b =', sess.run(b)[0])
    print('loss =', sess.run(loss))
```

Построим график сгенерированных данных и наложим на него предсказанную модель. В данном случае моделью является прямая линия.

```
# Построение графика входных данных
plt.plot(x_data, y_data, 'ro')

# Построение графика предсказанной выходной линии
plt.plot(x_data, sess.run(W) * x_data + sess.run(b))
```

Зададим параметры графика.

```
# Задание параметров графика
plt.xlabel('Размерность 0')
plt.ylabel('Размерность 1')
plt.title('Итерация ' + str(step+1) + ' из ' + str(num_iterations))
plt.show()
```

Полный код примера содержится в файле `linear_regression.py`. В процессе выполнения этого кода откроется окно, отображающее входные данные (рис. 16.1).

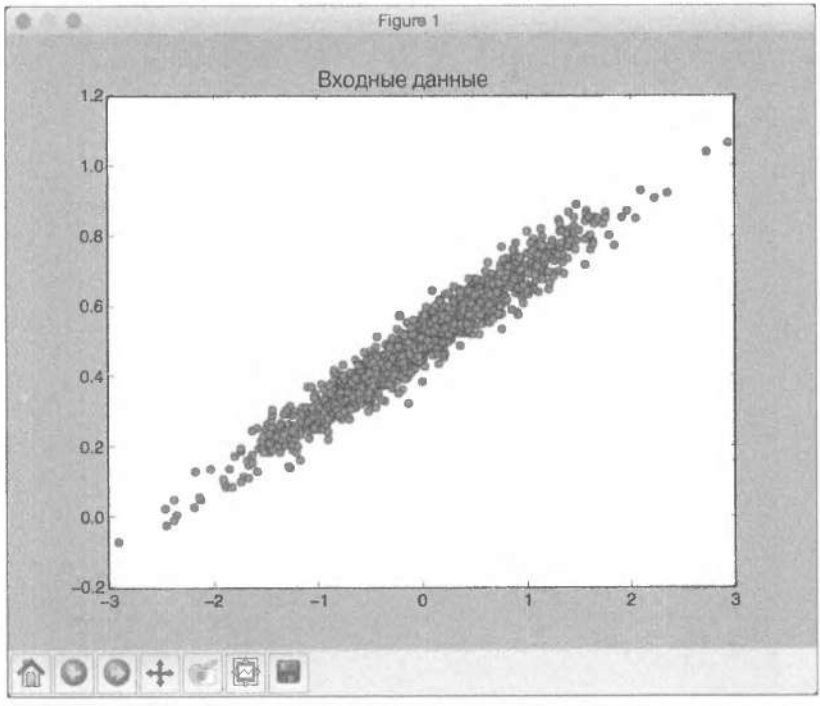

Рис. 16.1

Закрыв это окно, вы сможете наблюдать за ходом процесса обучения. Первая итерация будет выглядеть примерно так (рис. 16.2).

Как видите, между линией и данными наблюдается полное несовпадение. Закройте это окно, чтобы перейти к следующей итерации (рис. 16.3).

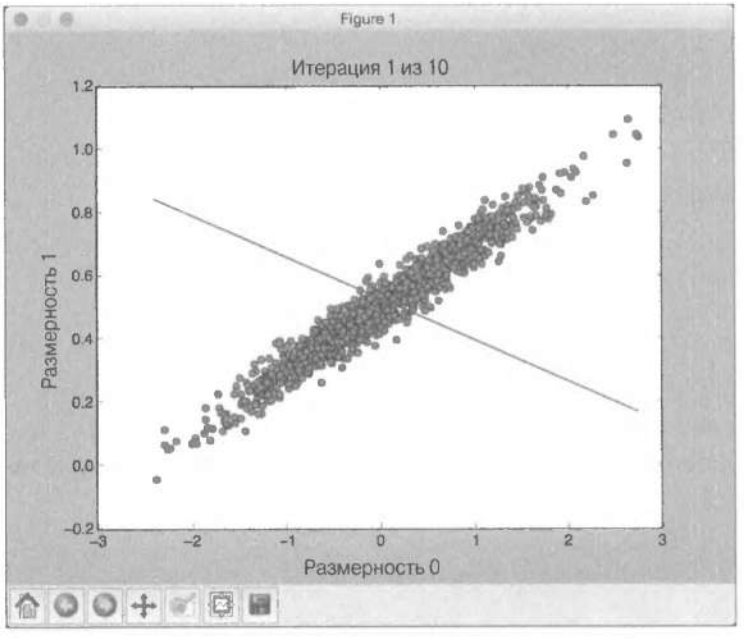

Рис. 16.2

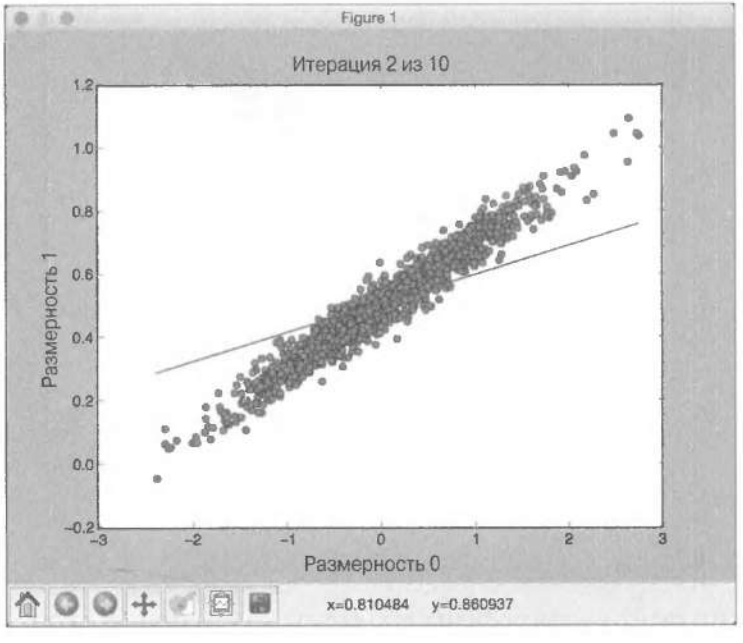

Рис. 16.3

На этот раз линия располагается ближе к данным, но отклонение все еще велико. Закройте это окно и продолжите итерации (рис. 16.4).

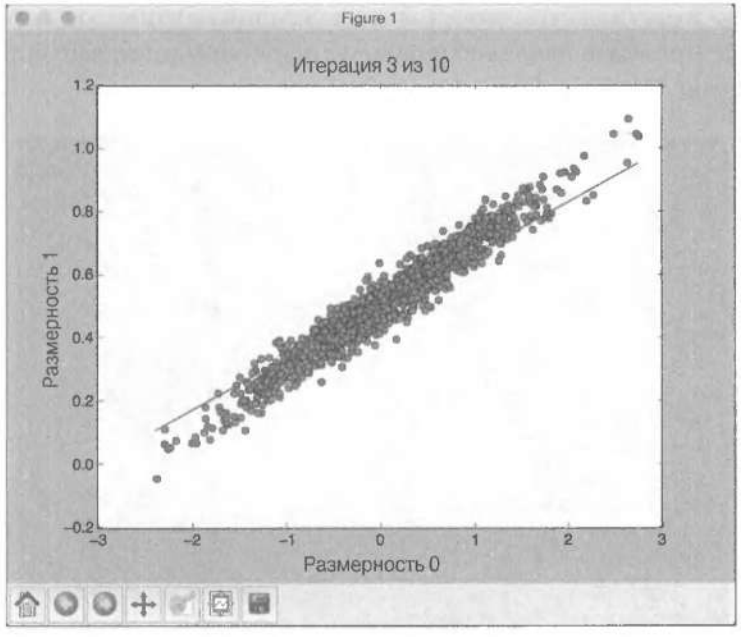

Рис. 16.4

Нетрудно заметить, что линия приблизилась к реальной модели. По мере продолжения итерационного процесса модель будет постепенно улучшаться. Восьмая итерация будет выглядеть примерно так (рис. 16.5).

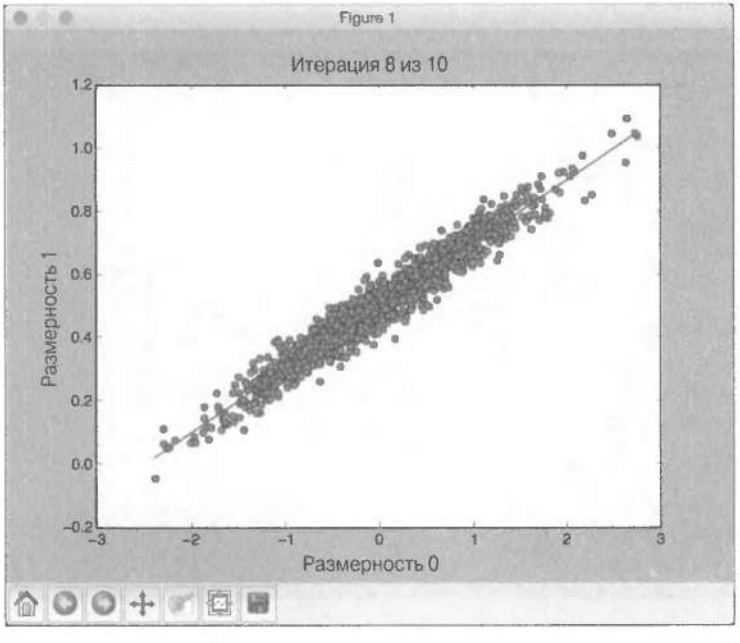

Рис. 16.5

Нетрудно заметить, что теперь линия намного лучше согласуется с данными. Ниже приведен начальный фрагмент информации, выводимой в окне терминала (рис. 16.6).

```
ITERATION 1
W = -0.130961
b = 0.53005
loss = 0.0760343

ITERATION 2
W = 0.0917911
b = 0.508959
loss = 0.00960302

ITERATION 3
W = 0.164665
b = 0.502555
loss = 0.00250165

ITERATION 4
W = 0.188492
b = 0.500459
loss = 0.0017425
```

Рис. 16.6

По завершении процесса обучения в окне терминала отобразится следующая информация (рис. 16.7).

```
ITERATION 7
W = 0.199662
b = 0.499477
loss = 0.00165175

ITERATION 8
W = 0.199934
b = 0.499453
loss = 0.00165165

ITERATION 9
W = 0.200023
b = 0.499445
loss = 0.00165164

ITERATION 10
W = 0.200052
b = 0.499443
loss = 0.00165164
```

Рис. 16.7

Создание классификатора изображений на основе однослойной нейронной сети

Рассмотрим пример построения однослойной нейронной сети с использованием TensorFlow и создания классификатора изображений на ее основе. Для построения нашей системы мы используем базу данных изображений MNIST, содержащую образы рукописных цифр. Нашей целью является создание классификатора, способного корректно идентифицировать цифру в каждом изображении.

Создайте новый файл Python и импортируйте следующие пакеты.

```
import argparse

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

Определим функцию для анализа входных аргументов.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Build a
        classifier using MNIST data')
    parser.add_argument('--input-dir', dest='input_dir', type=str,
        default='./mnist_data', help='Directory for storing data')
    return parser
```

Определим основную функцию и проанализируем входные аргументы.

```
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
```

Извлечем данные изображений MNIST. С помощью флага `one_hot` мы указываем, что будем использовать *прямое* (one-hot) кодирование в наших признаках. Это означает, что если мы имеем `n` классов, то признаком конкретной точки данных будет служить массив длины `n`. Каждый элемент этого массива соответствует определенному классу. Указание класса будет осуществляться посредством установки соответствующего индекса в `1`, а всех остальных индексов — в `0`.

```
    # Получение данных MNIST
    mnist = input_data.read_data_sets(args.input_dir, one_hot=True)
```

Изображения в указанной базе данных имеют размер *28×28*. Для создания входного слоя мы должны преобразовать изображение в одномерный массив.

```
# Изображения имеют размер 28×28, поэтому создается
# входной слой с 784 нейронами (28x28=784)
x = tf.placeholder(tf.float32, [None, 784])
```

Создадим однослойную нейронную сеть с весами и смещениями. В базе данных имеются 10 различных цифр. Количество нейронов во входном слое равно *784*, а количество нейронов в выходном слое — *10*.

```
# Создание слоя с весами и смещениями. Всего имеется
# 10 различных цифр, поэтому выходной слой должен
# иметь 10 классов
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

Создадим уравнение, которое будет использоваться для обучения.

```
# Создание уравнения для 'y': y = W*x + b
y = tf.matmul(x, W) + b
```

Определим функцию потерь и оптимизатор, использующий метод градиентного спуска.

```
# Определим энтропийные потери и оптимизатор на основе
# метода градиентного спуска
y_loss = tf.placeholder(tf.float32, [None, 10])
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y,
                      y_loss))
optimizer = tf.train.GradientDescentOptimizer(0.5).minimize(loss)
```

Инициализируем все переменные.

```
# Инициализация всех переменных
init = tf.initialize_all_variables()
```

Создадим сеанс TensorFlow и запустим его.

```
# Создание сеанса
session = tf.Session()
session.run(init)
```

Запустим процесс обучения. В ходе обучения будут использоваться фрагменты изображений: оптимизатор будет выполняться для текущего фрагмента и переходить к обработке очередного фрагмента на следующей итерации. На каждой итерации первым шагом является получение очередного фрагмента изображения, используемого для обучения.

```
# Запуск обучения
num_iterations = 1200
```

```
batch_size = 90
for _ in range(num_iterations):
    # Получение очередного фрагмента изображения
    x_batch, y_batch = mnist.train.next_batch(batch_size)
```

Выполним оптимизатор для данного фрагмента изображения.

```
    # Обучение на данном фрагменте изображения
    session.run(optimizer, feed_dict = {x: x_batch, y_loss:
                y_batch})
```

Как только процесс обучения завершится, вычислим точность, используя тестовый набор данных.

```
# Вычисление точности с использованием тестовых данных
predicted = tf.equal(tf.argmax(y, 1), tf.argmax(y_loss, 1))
accuracy = tf.reduce_mean(tf.cast(predicted, tf.float32))
print('\nAccuracy =', session.run(accuracy, feed_dict = {
      x: mnist.test.images, y_loss: mnist.test.labels}))
```

Полный код примера содержится в файле `single_layer.py`. Когда вы запустите программу, она загрузит данные в подпапку `mnist_data` текущей папки. Это режим по умолчанию. Если вы хотите использовать другую папку, укажите ее с помощью входного аргумента. Выполнив данный код, вы получите в окне терминала следующий вывод (рис. 16.8).

```
Extracting ./mnist_data/train-images-idx3-ubyte.gz
Extracting ./mnist_data/train-labels-idx1-ubyte.gz
Extracting ./mnist_data/t10k-images-idx3-ubyte.gz
Extracting ./mnist_data/t10k-labels-idx1-ubyte.gz

Accuracy = 0.921
```

Рис. 16.8

Отсюда следует, что точность построенной модели составляет 92,1%.

Создание классификатора изображений на основе сверточной нейронной сети

Работу классификатора, рассмотренного в предыдущем разделе, нельзя считать достаточно удовлетворительной. Получение точности 92,1% на наборе данных MNIST — относительно простая задача. В этом разделе будет продемонстрировано, что использование сверточных нейронных сетей (CNN) позволяет достигнуть намного более высокой точностью. Мы создадим классификатор изображений, используя тот же набор данных, но на основе CNN, а не однослойной нейронной сети.

Создадим новый файл Python и импортируем следующие пакеты.

```
import argparse

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

Определим функцию для анализа входных аргументов.

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Build a CNN
             classifier using MNIST data')
    parser.add_argument('--input-dir', dest='input_dir',
                        type=str, default='./mnist_data',
                        help='Directory for storing data')
    return parser
```

Определим функцию, создающую значения весов в каждом слое.

```
def get_weights(shape):
    data = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(data)
```

Определим функцию, создающую значения смещений в каждом слое.

```
def get_biases(shape):
    data = tf.constant(0.1, shape=shape)
    return tf.Variable(data)
```

Определим функцию, создающую слой на основе входной формы.

```
def create_layer(shape):
    # Получение весов и смещений
    W = get_weights(shape)
    b = get_biases([shape[-1]])

    return W, b
```

Определим функцию для выполнения 2D-свертки.

```
def convolution_2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

Определим функцию, выполняющую операцию объединения типа Max для размерности 2×2.

```
def max_pooling(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
            strides=[1, 2, 2, 1], padding='SAME')
```

Определим основную функцию и проанализируем входные аргументы.

```
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
```

Извлечем данные изображений MNIST.

```
    # Получение данных MNIST
    mnist = input_data.read_data_sets(args.input_dir, one_hot=True)
```

Создадим входной слой с 784 нейронами.

```
    # Изображения имеют размер 28x28, поэтому создаем входной
    # слой, содержащий 784 нейрона (28x28=784)
    x = tf.placeholder(tf.float32, [None, 784])
```

Мы будем использовать сверточную нейронную сеть, использующую преимущества двумерной структуры изображений. Поэтому преобразуем `x` в четырехмерный тензор, второе и третье измерения которого соответствуют размерам изображения.

```
    # Переформирование 'x' в четырехмерный тензор
    x_image = tf.reshape(x, [-1, 28, 28, 1])
```

Создадим первый сверточный слой, который будет извлекать 32 признака для каждого из фрагментов размером 5×5.

```
    # Определение первого сверточного слоя
    W_conv1, b_conv1 = create_layer([5, 5, 1, 32])
```

Выполним свертку изображения с использованием тензора весов, вычисленного на предыдущем шаге, а затем прибавим к нему тензор смещений. После этого мы применим к полученному результату функцию активации **ReLU** (Rectified Linear Unit — скорректированный линейный блок, или "выпрямитель").

```
    # Свертка изображения с помощью тензора весов, добавление
    # тензора смещений и применение функции ReLU
    h_conv1 = tf.nn.relu(convolution_2d(x_image, W_conv1) + b_conv1)
```

Применим оператор `max_pooling` размерностью 2×2 к результату, полученному на предыдущем шаге.

```
    # Применение оператора max_pooling
    h_pool1 = max_pooling(h_conv1)
```

Создадим второй сверточный слой, вычисляющий 64 признака для каждого из блоков размером 5×5.

```
# Определим второй сверточный слой
W_conv2, b_conv2 = create_layer([5, 5, 32, 64])
```

Выполним свертку результата предыдущего слоя с использованием тензора весов, вычисленного на предыдущем шаге, и прибавим к нему тензор смещений. После этого мы должны применить к результату функцию ReLU.

```
# Свертка результата предыдущего слоя с помощью тензора весов,
# добавление тензора смещений и применение функции ReLU
h_conv2 = tf.nn.relu(convolution_2d(h_pool1, W_conv2) + b_conv2)
```

Применим оператор `max_pooling` размерностью 2×2 к результату, полученному на предыдущем шаге.

```
# Применение оператора max_pooling
h_pool2 = max_pooling(h_conv2)
```

Теперь изображение уменьшено до размера 7×7. Создадим полносвязный слой с 1024 нейронами.

```
# Определение полносвязного слоя
W_fc1, b_fc1 = create_layer([7 * 7 * 64, 1024])
```

Переформируем результат предыдущего слоя.

```
# Переформирование результата предыдущего слоя
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
```

Умножим результат предыдущего слоя на тензор весов полносвязного слоя, добавим тензор смещений и применим к полученному результату функцию ReLU.

```
# Умножение результата предыдущего слоя на тензор весов,
# добавление тензора смещений и применение функции ReLU
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

Чтобы предотвратить переобучение сети, мы должны создать *исключающий слой* (dropout layer). Создадим заполнитель TensorFlow для значений, задающих вероятность того, что результат нейрона будет оставлен при исключении нейронов из слоя.

```
# Определение исключающего слоя с использованием
# вероятностного заполнителя для всех нейронов
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Определим считывающий слой с десятью выходными нейронами, соответствующими десяти классам в нашем наборе данных. Вычислим результат.

```
# Определение считывающего слоя (выходной слой)
W_fc2, b_fc2 = create_layer([1024, 10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

Определим функцию потерь и функцию оптимизатора.

```
# Определение энтропийных потерь и оптимизатора
y_loss = tf.placeholder(tf.float32, [None, 10])
loss = tf.reduce_mean(tf.nn.
       softmax_cross_entropy_with_logits(y_conv, y_loss))
optimizer = tf.train.AdamOptimizer(1e-4).minimize(loss)
```

Определим способ вычисления точности.

```
# Определение способа вычисления точности
predicted = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_loss, 1))
accuracy = tf.reduce_mean(tf.cast(predicted, tf.float32))
```

Создадим и запустим сеанс после инициализации переменных.

```
# Создание и запуск сеанса
sess = tf.InteractiveSession()
init = tf.initialize_all_variables()
sess.run(init)
```

Запустим процесс обучения.

```
# Запуск обучения
num_iterations = 21000
batch_size = 75
print('\nTraining the model....')
for i in range(num_iterations):
    # Получение следующего блока изображений
    batch = mnist.train.next_batch(batch_size)
```

Выведем данные о повышении точности через каждые 50 итераций.

```
    # Вывод данных о ходе процесса
    if i % 50 == 0:
        cur_accuracy = accuracy.eval(feed_dict = {
                x: batch[0], y_loss: batch[1],
                keep_prob: 1.0})
        print('Iteration', i, ', Accuracy =', cur_accuracy)
```

Выполним оптимизатор для текущего блока.

```
# Обучение на текущем блоке
optimizer.run(feed_dict = {x: batch[0], y_loss:
              batch[1], keep_prob: 0.5})
```

По завершении процесса обучения вычислим точность, используя тестовый набор данных.

```
# Вычисление точности с использованием тестовых данных
print('Test accuracy =', accuracy.eval(feed_dict = {
      x: mnist.test.images, y_loss: mnist.test.labels,
      keep_prob: 1.0}))
```

Полный код примера содержится в файле `cnn.py`. Выполнив этот код, вы получите следующий вывод в окне терминала (рис. 16.9).

```
Extracting ./mnist_data/train-images-idx3-ubyte.gz
Extracting ./mnist_data/train-labels-idx1-ubyte.gz
Extracting ./mnist_data/t10k-images-idx3-ubyte.gz
Extracting ./mnist_data/t10k-labels-idx1-ubyte.gz

Training the model....
Iteration 0 , Accuracy = 0.0533333
Iteration 50 , Accuracy = 0.813333
Iteration 100 , Accuracy = 0.8
Iteration 150 , Accuracy = 0.906667
Iteration 200 , Accuracy = 0.84
Iteration 250 , Accuracy = 0.92
Iteration 300 , Accuracy = 0.933333
Iteration 350 , Accuracy = 0.866667
Iteration 400 , Accuracy = 0.973333
Iteration 450 , Accuracy = 0.933333
Iteration 500 , Accuracy = 0.906667
Iteration 550 , Accuracy = 0.853333
Iteration 600 , Accuracy = 0.973333
Iteration 650 , Accuracy = 0.973333
Iteration 700 , Accuracy = 0.96
Iteration 750 , Accuracy = 0.933333
```

Рис. 16.9

По мере увеличения количества выполненных итераций точность повышается (рис. 16.10).

Полученные нами результаты свидетельствуют о том, что сверточные нейронные сети обеспечивают намного более высокую точность по сравнению с простыми нейронными сетями.

```
Iteration 2900 , Accuracy = 0.973333
Iteration 2950 , Accuracy = 1.0
Iteration 3000 , Accuracy = 0.973333
Iteration 3050 , Accuracy = 1.0
Iteration 3100 , Accuracy = 0.986667
Iteration 3150 , Accuracy = 1.0
Iteration 3200 , Accuracy = 1.0
Iteration 3250 , Accuracy = 1.0
Iteration 3300 , Accuracy = 1.0
Iteration 3350 , Accuracy = 1.0
Iteration 3400 , Accuracy = 0.986667
Iteration 3450 , Accuracy = 0.946667
Iteration 3500 , Accuracy = 0.973333
Iteration 3550 , Accuracy = 0.973333
Iteration 3600 , Accuracy = 1.0
Iteration 3650 , Accuracy = 0.986667
Iteration 3700 , Accuracy = 1.0
Iteration 3750 , Accuracy = 1.0
Iteration 3800 , Accuracy = 0.986667
Iteration 3850 , Accuracy = 0.986667
Iteration 3900 , Accuracy = 1.0
```

Рис. 16.10

Резюме

Эта глава была посвящена глубокому обучению и сверточным нейронным сетям (CNN). Мы обсудили, что собой представляют CNN, зачем они нужны и какова их архитектура. Вы узнали о различных типах слоев, используемых в CNN. Вы познакомились с библиотекой TensorFlow, которую мы использовали для создания линейного регрессора на основе перцептрона. Было показано, как создать классификатор изображений на основе однослойной нейронной сети, после чего мы создали классификатор изображений на основе CNN.

Предметный указатель

A

AdaBoost 103

B

Bag of Words 276
BFS 186

C

CAMShift 357
CMA-ES 221
CNN 422, 433
  типы слоев 423
CRF 313
CSP 187

D

DFS 186

E

easyAI 249
EM-алгоритм 126

F

F-мера 85

G

GMM 124
GPS 35

H

HMM 309, 337
HSV 349

I

IPL 36

M

MAP-алгоритм 126
matplotlib 41
MFCC 333
MiniMax 247
MNIST 431

N

NegaMax 248
NeuroLab 381
NLP 267
NumPy 40

O

OCR 399
OpenAI Gym 411
OpenCV 346

P

Pandas 296
Python 39

R

ReLU 424, 435

S

scikit-learn 41
SciPy 40
SVM 64

T

TensorFlow 424
tf-idf 280

U

UCS 186

А

Агент обучения 408
  создание 416
Алгоритм
  A* 200
  MiniMax 247
  NegaMax 248

SSS* 259
генетический 211
распространения сходства 130
сдвига среднего 357
эволюционный 211, 230
Альфа-бета-отсечение 248
Аналитическая модель 38
Ансамблевое обучение 79
Аудиосигнал 324, 337
генерирование 329

Б

Бинаризация 48

В

Вектор движения 365
Векторная квантизация 392
Временной ряд 296
срез 300
статистики 305
Выброс 50
Вычитание фона 353

Г

Гауссовская модель 124
Генеалогическое дерево 170
Генетический алгоритм 211
Генетическое программирование 30, 230
Глубокое обучение 422

Д

Декларативное программирование 164
Дерево принятия решений 81
Дискретное косинусное
преобразование 334

Е

Евклидово расстояние 151

Ж

Жадный поиск 189

З

Зависимая переменная 69
Задача с ограничениями 187

И

Имитация отжига 188
Императивное программирование 164
Интеллектуальный агент 37
Информированный поиск 186
Исключение среднего 48
Искусственный интеллект 22
логический 28
Исчерпывающий поиск 246
Итератор целей 169

К

Каскад Хаара 371
Качество 58
Классификатор 381
Классификация 46
Кластеризация 112
оценка качества 121
Когнитивное моделирование 33
Кодирование меток 51
Коллаборативная фильтрация 155
Комбинаторный поиск 246
Компьютерное зрение 25
Коэффициент MFCC 333
Критериальная переменная 69
Кроссовер 213

Л

Латентное размещение Дирихле 291
Лексема 269
Лемматизация 272
Линейный регрессор 424
Логистическая регрессия 52
Логистическая функция 52
Логическое программирование 163
Локальный оптимум 213
Локальный поиск 187

М

Максимальный зазор 64
Маскированная переменная 168
Масштабирование 49
Матрица неточностей 61
Машина опорных векторов 64
Машинное обучение 28, 45

Метка 51
Метод
  K ближайших соседей 140, 143
  k-средних 112
  Виолы–Джонса 371
  грубой силы 246
  крутого восхождения 188
  Лукаса–Канаде 365
  наименьших абсолютных
    отклонений 50
  наименьших квадратов 50
  сдвига среднего 116
Мутация 213

Н

Наивный байесовский классификатор 57
Независимая переменная 69
Неинформированный поиск 186
Нейронная сеть 380
  глубокая 422
  многослойная 388
  однослойная 384, 431
  рекуррентная 395
  сверточная 422, 433
  тренировка 380
Неконтролируемое обучение 112
Непараметрический алгоритм 116
Нормализация 50

О

Обнаружение лиц 370
Обобщенная линейная модель 52
Обработка естественного языка 267
Обучающий конвейер 138
Обучение
  без учителя 46, 111
  с подкреплением 407
  с учителем 46, 408
Объектно-ориентированное
    программирование 164
Опорный вектор 64
Оптический поток 364
Оптическое распознавание символов 399
Отсечение 248
Отслеживание
  глаз 375
  объектов 349
Оценка сходства 150
  по Пирсону 151
Оценочная функция 212
Ошибка I рода 61

П

Пакет
  cvxopt 296
  deap 214
  gensim 269
  hmmlearn 296, 337
  logpy 167
  matplotlib 317
  nltk 268
  numpy 329
  pattern 269
  pystruct 296
  python_speech_features 334
  simpleai 190
  sklearn 41, 75, 137
  sympy 167
  Tkinter 53
Парадигма 163
Параметрическая модель 126
Перцептрон 381, 424
Полносвязность 423
Полнота 59
Полупараметрическая модель 125
Популяция 213
Последовательные данные 295, 395
Правило 166
Предельно случайный лес 86
Предиктор 69
Преобразование Фурье 327
Признак 48
  Хаара 372
Прогнозатор категорий 280
Простое число 169
Процедурное программирование 164
Прямое кодирование 431

Р

Разделяющая гиперплоскость 64
Распознавание
  образов 28
  речи 26, 323, 337
Рациональный агент 34
Реверсивный (обратный) поиск
    изображений 25
Регрессия 69
  логистическая 52
  символическая 230
Регрессор 70
  AdaBoost 103
  многомерный 73
Рекомбинация 213
Рекомендательная система 150, 158
Речевой признак 333
Робототехника 27

С

Сверточная нейронная сеть 422, 433
Сдвиг среднего 357
Сентимент-анализ 286
Сеточный поиск 100
Сигмоида 52
Силуэтная оценка 121
Символическая регрессия 230
Символическое программирование 164
Скрытая марковская модель 309, 337
Случайный лес 85
Смешанная модель 124
Смещение нейрона 381
Спонтанное обучение 112
Срез 300
Статистика tf-idf 280
Стемминг 271
Стоимостный анализ 188
Стохастический поиск 188
Стратегия отсечения 247

Т

Тематическое моделирование 290
Терм-документная матрица 277
Тест Тьюринга 30
Токенизация 269
Точность 59

У

Универсальный решатель задач 35
Унификация 166
Уровень доверительности 91
Условное случайное поле 313

Ф

Факт 166
Функциональное программирование 164
Функция
  активации 435
  оценочная 212
  приспособленности 212

Ц

Цветовое пространство 349
Целевая функция 188
Центроид 112

Ч

Чанкинг 274
Частотность слов 280
Частотный интервал 326

Ш

Ширина окна 118

Э

Эволюционный алгоритм 211, 230
Эвристика 30
Эвристический поиск 186
Экспертная система 26
Энтропия 81