Содержание

Об авторах ... 9

О рецензентах ... 10

Предисловие ... 11

Глава 1. Первые шаги к масштабируемости ... 20

Подробное объяснение термина масштабируемости ... 21
- Приведение крупномасштабных примеров ... 23
- Введение в язык Python ... 24
- Вертикальное масштабирование средствами Python ... 25
- Горизонтальное масштабирование средствами Python ... 26

Python для крупномасштабного машинного обучения ... 27
- Выбор между Python 2 и Python 3 ... 27
- Инсталляция среды Python ... 28
- Пошаговая установка ... 28
- Установка библиотек ... 29
- Способы обновления библиотек ... 31
- Научные дистрибутивы ... 32
- Введение в Jupyter ... 33

Библиотеки Python ... 37
- NumPy ... 37
- SciPy ... 37
- Pandas ... 37
- Scikit-learn ... 38

Резюме ... 44

Глава 2. Масштабируемое обучение в Scikit-learn ... 46

Внеядерное обучение ... 47
- Подвыборка как приемлемый вариант ... 48
- Оптимизация по одному прецеденту за раз ... 48
- Создание системы внеядерного обучения ... 50

Потоковая передача данных из источников ... 51
- Наборы данных для реальных дел ... 51
- Первый пример – потоковая передача набора данных Bike-sharing ... 54
- Использование инструментов ввода-вывода библиотеки pandas ... 56
- Работа с базами данных ... 57
- Особое внимание упорядочению прецедентов ... 61

Стохастическое обучение ... 63
- Пакетный градиентный спуск ... 64
- Стохастический градиентный спуск ... 67
- Реализация алгоритма SGD в библиотеке Scikit-learn ... 68
- Определение параметров обучения алгоритма SGD ... 70

Управление признаками на потоках данных ... 72
- Описание целевой переменной ... 76

Хэширование признаков ... 79
Другие элементарные преобразования ... 82
Тестирование и перекрестная проверка в потоке ... 83
Применение алгоритма SGD в деле ... 84
Резюме ... 88

Глава 3. Быстрообучающиеся реализации машин SVM ... 89
Наборы данных для самостоятельного экспериментирования ... 90
Набор данных Bike-sharing ... 90
Набор данных Covertype ... 91
Машины опорных векторов ... 91
Кусочно-линейная функция потерь и ее варианты ... 97
Объяснение реализации алгоритма SVM в Scikit-learn ... 98
Поиск нелинейных SVM с привлечением подвыборки ... 101
Реализация SVM в крупном масштабе на основе SGD ... 104
Отбор признаков посредством регуляризации ... 112
Добавление нелинейности в алгоритм SGD ... 114
Испытание явных высокоразмерных отображений ... 115
Доводка гиперпараметров ... 117
Другие альтернативы быстро обучающихся реализаций SVM ... 121
Резюме ... 133

Глава 4. Искусственные нейронные сети и глубокое обучение ... 134
Архитектура нейронной сети ... 135
Чему и как нейронные сети обучаются ... 144
Выбор правильной архитектуры ... 148
Нейронные сети в действии ... 149
Параллелизация для библиотеки sknn ... 150
Нейронные сети и регуляризация ... 151
Нейронные сети и гиперпараметрическая оптимизация ... 153
Нейронные сети и границы решения ... 154
Глубокое обучение в крупном масштабе с H2O ... 157
Крупномасштабное глубокое обучение с H2O ... 158
Сеточный поиск в H2O ... 161
Глубокое обучение и предтренировка без учителя ... 162
Глубокое обучение с theanets ... 162
Автокодировщики и обучение без учителя ... 164
Автокодировщик ... 164
Резюме ... 168

Глава 5. Глубокое обучение с библиотекой TensorFlow ... 170
Инсталляция TensorFlow ... 172
Операции TensorFlow ... 172
Машинное обучение в TensorFlow посредством SkFlow ... 177
Глубокое обучение с большими файлами – инкрементное обучение ... 183
Инсталляция библиотеки Keras и платформа TensorFlow ... 186

Сверточные нейронные сети в TensorFlow посредством Keras ... 190
Сверточный слой ... 192
Объединяющий слой ... 193
Полносвязный слой ... 194
CNN-сети с подходом на основе инкрементной тренировки ... 195
Вычисления на GPU ... 196
Резюме ... 199

Глава 6. Классификационные и регрессионные деревья в крупном масштабе ... 200
Агрегация бутстрапированных выборок ... 203
Случайный лес и экстремально рандомизированный лес ... 204
Быстрая параметрическая оптимизация посредством рандомизированного поиска ... 208
Экстремально рандомизированные деревья и большие наборы данных ... 210
Алгоритм CART и бустинг ... 214
Машины градиентного бустинга ... 214
Алгоритм XGBoost ... 221
Регрессия на основе XGBoost ... 224
Потоковая передача больших наборов данных посредством XGBoost ... 227
Персистентность модели XGBoost ... 228
Внеядерный алгоритм CART в среде H2O ... 229
Случайный лес и сеточный поиск в H2O ... 229
Стохастический градиентный бустинг и сеточный поиск в H2O ... 231
Резюме ... 233

Глава 7. Обучение без учителя в крупном масштабе ... 235
Методы машинного обучения без учителя ... 235
Разложение признаков – PCA ... 236
Алгоритм PCA в среде H2O ... 246
Кластеризация – алгоритм K-средних ... 247
Методы инициализации ... 250
Допущения алгоритма K-средних ... 251
Подбор оптимальной величины K ... 253
Масштабирование алгоритма K-средних – мини-пакет ... 257
Алгоритм K-средних в среде H2O ... 261
Алгоритм LDA ... 263
Масштабирование алгоритма LDA – оперативная память, CPU и машины ... 271
Резюме ... 272

Глава 8. Распределенные среды – Hadoop и Spark ... 273
От автономной машины к набору узлов ... 273
Зачем нужна распределенная платформа? ... 275
Настройка виртуальной машины ... 276
Виртуализатор VirtualBox ... 277
Конфигуратор Vagrant ... 279

Использование виртуальной машины ... 279
Экосистема Hadoop ... 281
Архитектура ... 281
Распределенная файловая система HDFS ... 282
Вычислительная парадигма MapReduce ... 289
Менеджер ресурсов YARN ... 298
Платформа Spark ... 299
Библиотека pySpark ... 299
Резюме ... 309

Глава 9. Практическое машинное обучение в среде Spark ... 310
Настройка виртуальной машины для данной главы ... 310
Распространение переменных по всем узлам кластера ... 311
Широковещательные переменные только для чтения ... 311
Аккумуляторные переменные только для записи ... 313
Широковещательные и аккумуляторные переменные – пример ... 314
Предобработка данных в среде Spark ... 316
Файлы JSON и объекты DataFrame платформы Spark ... 317
Работа с пропущенными данными ... 319
Группирование и создание таблиц в оперативной памяти ... 320
Запись предобработанного объекта DataFrame или RDD-набора на диск ... 322
Работа с объектами DataFrame ... 323
Машинное обучение с платформой Spark ... 326
Платформа Spark на наборе данных KDD99 ... 326
Чтение набора данных ... 327
Конструирование признаков ... 329
Тренировка ученика ... 334
Оценка результативности ученика ... 335
Возможности конвейера машинного обучения ... 336
Ручная доводка ... 338
Перекрестная проверка ... 340
Заключительная очистка ... 342
Резюме ... 342

Приложение. Введение в графические процессоры и платформа Theano ... 344
Вычисления на GPU ... 344
Платформа Theano – параллельные вычисления на GPU ... 346
Установка платформы Theano ... 347

Предметный указатель ... 350

Предисловие

> «Самое приятное в мозге – то, что можно узнать, что невежество вытесняется знанием и что крохотные частички знания постепенно образуют солидные пирамиды».
>
> *– Дуглас Хофштадтер*

Машинное обучение часто называют *подобластью искусственного интеллекта, которая реально работает*. Его цель состоит в том, чтобы, основываясь на существующем подмножестве данных (тренировочном наборе), с максимально возможной точностью найти функцию для предсказания исходов подмножества ранее не наблюдавшихся данных (тестового набора). Это происходит либо в форме меток и классов (задачи классификации), либо в форме непрерывного значения (задачи регрессии). Материальные примеры машинного обучения в реальных приложениях простираются от предсказания будущих курсов акций до идентификации половой принадлежности автора, исходя из серии документов. В этой книге читатель получит объяснение самых важных принципов работы машинного обучения, а также методов, подходящих для обработки крупных наборов данных, благодаря практическим примерам на языке Python. Мы рассмотрим обучение с учителем (классификация и регрессия) и обучение без учителя (в т. ч. анализ главных компонент, кластеризацию данных и тематическое моделирование), которые оказались применимыми к большим наборам данных.

Работающие в области ИТ крупные корпорации, такие как Google, Facebook и Uber, подняли нешуточный ажиотаж, утверждая, что они успешно применили подобного рода методы машинного обучения в крупном масштабе. С появлением и распространением больших данных спрос на масштабируемые решения в области машинного обучения рос экспоненциально, и множество других компаний и отдельных специалистов устремилось на поиски зрелых плодов скрытых корреляций в больших наборах данных. К сожалению, большинство обучающихся алгоритмов не очень хорошо масштабируется, перегружая центральные процессоры и память настольного компьютера или большого вычислительного кластера. В нынешние времена, несмотря на то что *большие данные* преодолели пик ажиотажа, масштабируемых решений для машинного обучения до сих пор не так уж и много.

Откровенно говоря, нам по-прежнему приходится обходить немало узких мест даже с наборами данных, которые едва ли можно классифицировать как *большие данные* (речь идет о наборах данных объемом до 2 Гб или меньше). Главная задача настоящей книги состоит в том, чтобы предоставить способы (иногда нестандартные) для применения самых мощных методов машинного обучения с открытым исходным кодом в более крупном масштабе без привлечения дорогостоящих корпоративных решений или больших вычислительных кластеров. На протяжении всей книги мы будем использовать Python и некоторые другие легкодоступные решения, которые хорошо интегрируются в масштабируемых конвейерах машинного обучения. Чтение книги станет путешествием, которое переопределит все то,

что вы знали о машинном обучении, позволив вам сделать значительный рывок вперед в анализе по-настоящему больших данных.

О чем эта книга рассказывает

Глава 1 «*Первые шаги на пути к масштабируемости*» ставит задачу масштабируемого машинного обучения с правильной перспективой и знакомит с инструментами, которые мы будем использовать в этой книге.

Глава 2 «*Масштабируемое обучение в Scikit-learn*» обсуждает тему стратегии стохастического градиентного спуска с ограничением потребления оперативной памяти; этот прием основывается на теме *внеядерного* обучения. Мы также охватим технические приемы подготовки данных, включая хэширование признаков, способные справляться с самыми разнообразными данными.

Глава 3 «*Быстрообучающиеся реализации машин SVM*» посвящена потоковым алгоритмам, которые способны обнаруживать нелинейности в форме машин опорных векторов. Мы представим альтернативы программной библиотеке Scikit-learn, в частности программы LIBLINEAR и Vowpal Wabbit, которые, несмотря на то что выполняются как команды внешней оболочки, с легкостью обертываться в сценарии Python и ими управляются.

Глава 4 «*Нейронные сети и глубокое обучение*» охватывает полезную методику применения глубоких нейронных сетей в платформе Theano и крупномасштабные приложения на основе платформы H2O. Хотя эта тема сегодня является актуальной, их успешное применение может представлять серьезную трудность, уже не говоря о предложении масштабируемых решений. Мы также обратимся к предварительной тренировке без учителя с автокодировщиками на основе библиотеки theanets.

Глава 5 «*Глубокое обучение с библиотекой TensorFlow*» охватывает интересную методику глубокого обучения нейронных сетей и онлайновые методы. Хотя платформа TensorFlow находится в стадии становления, она предоставляет изящные решения для машинного обучения. Мы также воспользуемся возможностями библиотеки Keras по работе со сверточными нейронными сетями в среде TensorFlow.

Глава 6 «*Классификационные и регрессионные деревья в крупном масштабе*» посвящена объяснению масштабируемых решений с использованием алгоритмов случайного леса, градиентного бустинга и экстремального градиентного бустинга XGboost. Алгоритм машинного обучения CART, аббревиатура для классификационных и регрессионных деревьев, обычно применяется в рамках ансамблевых методов. Мы также предоставим примеры крупномасштабного приложения в среде H2O.

Глава 7 «*Обучение без учителя в крупном масштабе*» вплотную посвящена обучению без учителя, а именно будут рассмотрены алгоритмы PCA, кластерного анализа и тематического моделирования с использованием соответствующего подхода, позволяющего масштабировать их вертикально.

Глава 8 «*Распределенные среды – Hadoop и Spark*» научит настраивать систему Spark в среде виртуальной машины с переходом от одиночной машины к парадигме вычислительной сети. Поскольку Python способен с легкостью связывать индивидуальные наработки и подключать их к кластеру машин, привлечение возможностей кластера Hadoop становится простым делом.

Глава 9 «*Практическое машинное обучение на платформе Spark*» посвящена непосредственной работе с платформой Spark и обучает необходимым основам, для того чтобы начать напрямую управлять данными и создавать прогнозные модели на больших наборах данных.

Приложение «Введение в графические процессоры и библиотеку Theano» коснется основ работы в среде Theano и вычислений на GPU. Это пособие поможет установить и подготовить свою собственную среду для использования Theano на GPU, если, конечно, располагаемая система это позволяет сделать.

Что требуется для этой книги

Выполнение примеров кода, прилагаемых к данной книге, требует установки Python 2.7 или старших версий в Mac OS, Linux или Microsoft Windows.

В примерах по всей книге будут часто использоваться важные программные библиотеки Python для научных и статистических вычислений, в частности SciPy, NumPy, Scikit-learn, StatsModels и реже matplotlib и pandas. Мы также задействуем приложение для внеядерных облачных вычислений под названием H2O.

Эта книга в значительной мере зависит от интерактивной среды программирования Jupyter и ее так называемых блокнотов, или записных книжек, приводимых в действие ядром Python. Для этой книги мы воспользуемся ее последней версией 4.1 (на момент публикации перевода – 5.0).

Первая глава предоставит все пошаговые инструкции и некоторые полезные советы по настройке среды Python, упомянутых выше рабочих программных библиотек и всего необходимого инструментария.

Для кого эта книга

Эта книга предназначена для начинающих и действующих практиков в области науки о данных, разработчиков и всех тех, кто намеревается работать с большими и сложными наборами данных. Мы постарались сделать эту книгу максимально доступной для самой широкой аудитории. И все же, учитывая, что темы в этой книге достаточно продвинутые, читателям рекомендуется знать основные понятия из области машинного обучения, в частности классификацию и регрессию, функции минимизации ошибки и перекрестную проверку, однако это условие не является обязательным.

Мы также предполагаем, что читатель обладает некоторым опытом программирования на Python, работы с блокнотами Jupyter и выполнением команд из командной строки, а также познаниями в математике на разумном уровне, чтобы усвоить концепции, лежащие в основе разных больших решений, которые мы здесь предлагаем. Текст написан в стиле, который поймут программисты на других языках (R, Java и MATLAB). В идеальном случае он очень подходит для исследователя-аналитика (но не ограничен лишь им), знакомого с машинным обучением и заинтересованного в мобилизации возможностей языка Python, в отличие от других языков, таких как R или MATLAB, из-за его широких возможностей выполнять вычислительные задачи, работать с оперативной памятью и выполнять ввод-вывод данных.

Условные обозначения

В этой книге вы найдете несколько текстовых стилей, которые выделяют различные виды информации. Вот некоторые примеры этих стилей и объяснение их значения.

Ключевые слова в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, пути файловой системы, фиктивные URL, ввод данных пользователем и дескрипторы Twitter показаны следующим образом: «Во время обследования линейной модели сначала проверьте атрибут `coef_`».

Фрагмент исходного кода оформляется следующим образом:

```
from sklearn import datasets
iris = datasets.load_iris()
```

Поскольку в большинстве примеров мы будем использовать блокноты Jupyter, фрагменты исходного кода будут оформляться в соответствии с тем, как он выглядит в ячейках блокнотов: входные данные будут всегда помечены как `In:`, а выходные данные – часто как `Out:`. На компьютере требуется только ввести исходный код после `In:` и проверить, соответствуют ли результаты содержимому `Out:`:

```
In:
clf.fit(X, y)

Out:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
degree=3, gamma=0.0, kernel='rbf', max_iter=-1, probability=False,
random_state=None, shrinking=True, tol=0.001, verbose=False)
```

Когда команда предназначена для выполнения в командной строке терминала, такая команда будет предваряться префиксом `$>`, в противном случае, если она предназначена для интепретатора Python REPL, ей будет предшествовать приглашение `>>>`:

```
$ python
>>> import sys
>>> print sys.version_info
```

Новые термины и **важные слова** показаны полужирным шрифтом. Слова, которые выводятся на экран, например в меню или диалоговых окнах, выглядят в тексте следующим образом: «Как правило, необходимо просто ввести в ячейки после **In:** исходный код и его выполнить».


Предупреждения или важные примечания появляются в этом поле.


Подсказки и приемы появляются тут.


Дополнения к тексту оригинала книги.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или вебсайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Комментарий переводчика

Весь материал книги приведен в соответствие с последними действующими версиями программных библиотек (время перевода книги – май-июнь 2017 г.) и протестирован в среде Windows 10. При тестировании исходного кода за основу взят Python версии 2.7.13.

Большинство содержащихся в книге технических терминов и аббревиатур для удобства кратко определено в сносках, а для некоторых терминов в силу отсутствия единой терминологии приведены соответствующие варианты наименований или пояснения.

Прилагаемый к книге адаптированный и скорректированный исходный код примеров лучше всего разместить в подпапке домашней папки пользователя (`/home/Ваши_проекты_Python` или `C:\Users\[ИМЯ_ПОЛЬЗОВАТЕЛЯ]\Ваши_проекты_Python`). Ниже приведена структура папки с прилагаемыми примерами:

`Chapter 01-09`	Исходный код примеров в виде записных книжек Jupyter
`py_scripts`	Исходный код примеров в виде сценариев Python
`vowpal_wabbit_for_windows`	Исполнимые файлы программного продукта Vowpal Wabbit для 32- и 64-разрядных ОС Windows

Для просмотра исходного кода примеров лучше всего пользоваться блокнотами Jupyter. Они более читабельны, содержат графики, цветные рисунки и расширенные пояснения.

Далее приведены особенности инсталляции некоторых используемых программных библиотек Python.

Особенности программного обеспечения

В обычных условиях библиотеки Python можно скачать из каталога библиотек Python `PyPi` (https://pypi.python.org/). Однако следует учесть, что для работы библиотек SciPy и Scikit-learn в Windows требуется, чтобы в системе была установлена библиотека Numpy+MKL. Библиотека **Numpy+MKL** привязана к библиотеке Intel® Math Kernel Library и включает в свой состав необходимые динамические библиотеки (DLL) в каталоге numpy.core. Ее следует скачать с репозитория whl-файлов (http://www.lfd.uci.edu/~gohlke/pythonlibs/) и установить (например, `pip install numpy-1.13.0+mkl-cp27-cp27m-win_amd64.whl` для 64-разрядной операционной системы Windows и среды Python 2.7) как whl (соответствующая процедура установки описана ниже).

Далее приведены сведения о других основных библиотеках:

- **NumPy** – основополагающая библиотека, необходимая для научных вычислений на Python;
- **Matplotlib** – библиотека для работы с двумерными графиками. Требует наличия numpy и некоторых других;
- **Pandas** – инструмент для анализа структурных данных и временных рядов. Требует наличия numpy и некоторых других;
- **Scikit-learn** – интегратор классических алгоритмов машинного обучения. Требует наличия numpy+mkl;
- **SciPy** – библиотека, используемая в математике, естественных науках и инженерном деле. Требует наличия numpy+mkl;
- **Jupyter** – интерактивная вычислительная среда.

Факультативно:

- **PyQt5** – библиотека инструментов для программирования визуального интерфейса, требуется для работы инструментальной среды программирования Spyder;
- **Spyder** – инструментальная среда программирования на Python.

Протокол установки программного обеспечения

```
python -m pip install --upgrade pip
pip install numpy
```

либо как whl:

```
pip install numpy-1.13.0+mkl-cp27-cp27m-win_amd64.whl
pip install scipy
```

либо как whl:

```
pip install scipy-0.19.0-cp27-cp27m-win_amd64.whl
pip install Scikit-learn
```

либо как whl:

```
pip3 install scikit_learn-0.18.1-cp27-cp27m-win_amd64.whl
pip install matplotlib
pip install pandas
pip install gensim
pip install jupyter
pip install neurolab
pip install theano
pip3 install tensorflow
```

Следует отметить, что библиотека TensorFlow НЕ работает с Python 2. Для работы с библиотекой TensorFlow необходимо инсталлировать Python 3. Блокнот Jupyter (гл. 5) с примерами применения библиотеки TensorFlow использует ядро Python 3.

```
pip install scikit-neuralnetwork
pip install h2o
pip install keras
```

Факультативно:

```
pip install pyqt5
pip install spyder
```

Примечание: в зависимости от базовой ОС, версий языка Python и версий программных библиотек устанавливаемые вами версии whl-файлов могут отличаться от приведенных выше, где показаны последние на июнь 2017 г. версии для 64-разрядной ОС Windows и Python версии 2.7.

Работа со связкой Hadoop/Vagrant/Jupyter

Начало работы:

```
vagrant up (в папке с файлом Vagrantfile)
set PATH=%PATH%;C:\Program Files\Git\usr\bin (в командной строке cmd)
vagrant ssh
vagrant@sparkbox:~$ ./start_hadoop.sh (в консоли vagrant)
vagrant@sparkbox:~$ ./start_jupyter_yarn.sh
http://localhost:8888 (в браузере)
```

Запуск автономного режима без поддержки менеджера YARN:

```
vagrant up (в папке с файлом Vagrantfile)
set PATH=%PATH%;C:\Program Files\Git\usr\bin (в командной строке cmd)
vagrant ssh
vagrant@sparkbox:~$ ./start_jupyter_yarn.sh
http://localhost:8888 (в браузере)
```

Конец работы с Hadoop/Vagrant/Jupyter:

```
Ctrl + C (в командной строке блокнота)
vagrant@sparkbox:~$ ./stop_hadoop.sh (в консоли vagrant)
vagrant@sparkbox:~$ exit
vagrant halt (в командной строке)
```

Добавления в системную переменную Path в ОС Windows

После установки соответствующего программного обеспечения следует проверить наличие следующих значений в системной переменной Path:

```
%HADOOP_HOME%\bin
C:\HashiCorp\Vagrant\bin
C:\Program Files\Git\cmd
C:\Program Files\Git\usr\bin
C:\Program Files\mingw-w64\x86_64-7.1.0-win32-seh-rt_v5-rev0\mingw64\bin
```

Установка библиотек Python из whl-файла

Библиотеки для Python можно разрабатывать не только на чистом Python. Довольно часто библиотеки пишутся на C (динамические библиотеки), и для них пишется обертка Python, или же библиотека пишется на Python, но для оптимизации узких мест часть кода пишется на C. Такие библиотеки получаются очень быстрыми, однако библиотеки с вкраплениями кода на C программисту на Python тяжелее установить ввиду банального отсутствия соответствующих знаний либо необходимых компонентов и настроек в рабочей среде (в особенности в Windows). Для решения описанных проблем разработан специальный формат (файлы с расширением .whl) для распространения библиотек, который содержит заранее скомпилированную версию библиотеки со всеми ее зависимостями. Формат whl поддерживается всеми основными платформами (Mac OS X, Linux, Windows).

Установка производится с помощью менеджера библиотек pip. В отличие от обычной установки командой `pip install <имя_библиотеки>`, вместо имени библиотеки указывается путь к whl-файлу `pip install <путь/к/whl_файлу>`. Например:

```
pip install C:\temp\scipy-0.19.0-cp27-cp27m-win_amd64.whl
```

Откройте окно командой строки и при помощи команды `cd` перейдите в каталог, где размещен ваш whl-файл. Просто скопируйте туда имя вашего whl-файла. В этом случае полный путь указывать не понадобится. Например:

```
pip install scipy-0.19.0-cp27-cp27m-win_amd64.whl
```

При выборе библиотеки важно, чтобы разрядность устанавливаемой библиотеки и разрядность интерпретатора совпадали. Пользователи Windows могут брать whl-файлы на веб-странице http://www.lfd.uci.edu/~gohlke/pythonlibs/ Кристофа Голька из Лаборатории динамики флуоресценции Калифорнийского университета в г. Ирвайн. Библиотеки там постоянно обновляются, и в архиве содержатся все, какие только могут понадобиться.

Установка и настройка инструментальной среды Spyder

Spyder – это инструментальная среда для научных вычислений для языка Python (Scientific PYthon Development EnviRonment) для Windows, Mac OS X и Linux. Это

простая, легковесная и бесплатная интерактивная среда разработки на Python, которая предлагает функционал, аналогичный среде разработки на MATLAB, включая готовые к использованию виджеты `PyQt5` и `PySide`: редактор исходного кода, редактор массивов данных `NumPy`, редактор словарей, консоли Python и IPython и многое другое.

Чтобы установить среду Spyder в Ubuntu Linux, используя официальный менеджер библиотек, нужна всего одна команда:

```
sudo apt-get install spyder
```

Чтобы установить с использованием менеджера библиотек pip:

```
sudo apt-get install python-qt5 python-sphinx
sudo pip install spyder
```

И чтобы обновить:

```
sudo pip install -U spyder
```

Установка среды Spyder в Fedora 25:

```
dnf install python-spyder
```

Установка среды Spyder в Windows:

```
pip install spyder
```

Примечание: среда Spyder требует обязательной установки библиотеки PyQt5.

Глава 1

Первые шаги к масштабируемости

Добро пожаловать в книгу по масштабируемому машинному обучению на Python.

В этой главе мы обсудим способы эффективного обучения на больших данных в среде Python и как это можно осуществить, используя всего одну машину или кластер из других машин, который, к примеру, можно получить в веб-службах облачных вычислений **Amazon Web Services** (AWS) или в веб-службах платформы Google-облако.

В настоящей книге мы будем использовать реализацию масштабируемых алгоритмов машинного обучения на языке Python. Иными словами, они смогут работать с большим объемом данных и не дадут сбоя из-за нехватки оперативной памяти. Кроме того, работа таких алгоритмов будет занимать разумное количество времени, достаточно приемлемое для прототипа в области науки о данных и для развертывания проекта в эксплуатационной среде. Главы книги организованы вокруг решений (таких как потоковая передача данных), алгоритмов (таких как нейронные сети или ансамбль деревьев) и платформ (Hadoop или Spark). Мы также предложим небольшой справочник по алгоритмам машинного обучения и объясним, как сделать их масштабируемыми и пригодными для решения задач с крупными наборами данных.

С учетом таких стартовых предпосылок вам потребуется изучить основы (чтобы уяснить перспективу, с которой эта книга была написана), а также установить и настроить все основные инструменты, которые позволят незамедлительно приступить к чтению глав.

В этой главе мы представим следующие темы:

- что в действительности означает термин «масштабируемость»;
- на какие узкие места необходимо обратить внимание во время работы с данными;
- какие задачи эта книга поможет решать;
- как использовать Python для эффективного анализа наборов данных в крупном масштабе;
- каким образом быстро настроить свою машину для выполнения представленных в этой книге примеров.

Давайте же начнем наше совместное путешествие по масштабируемым решениям в среде Python!

Подробное объяснение термина масштабируемости

Несмотря на весь нынешний ажиотаж вокруг больших данных, большие наборы данных существовали задолго до того, как был введен сам термин. Большое количество текстовых данных, последовательностей ДНК и огромное количество данных с радиотелескопов всегда представляли проблему для ученых и исследователей-аналитиков. Поскольку большинство алгоритмов машинного обучения имеет вычислительную сложность $O(n^2)$ или даже $O(n^3)$, где n – это число тренировочных прецедентов, исследователи и аналитики прежде решали поставленные крупными наборами данных сложные задачи, привлекая более эффективные алгоритмы обработки данных. Алгоритм машинного обучения считается масштабируемым, когда после соответствующей настройки он может работать в условиях больших наборов данных. Набор данных может быть большим в силу большого количества прецедентов либо переменных, либо в силу обоих причин, а масштабируемый алгоритм может справляться с ними эффективно, поскольку его время выполнения увеличивается почти линейно в соответствии с размером задачи. Следовательно, это просто вопрос обмена в соотношении 1:1 большего количества времени (или большей вычислительной мощи) на большее количество данных. Между тем обычный алгоритм машинного обучения, когда сталкивается с большими объемами данных, не масштабируется; он попросту прекращает работать либо работает со временем выполнения, которое увеличивается нелинейно, например экспоненциально, тем самым делая обучение неосуществимым.

Внедрение дешевых систем хранения данных, больших RAM и многоядерных CPU кардинально все изменило, увеличив возможности одиночных ноутбуков по анализу больших объемов данных. Появление в недавнем прошлом еще одного игрока стало переломным моментом, переориентировав внимание с одиночных мощных машин на кластеры серийных компьютеров (более дешевых и легкодоступных). Эта серьезная перемена обусловила внедрение вычислительной сетевой парадигмы **MapReduce** и платформы с открытым исходным кодом Apache Hadoop с ее **распределенной файловой системой Hadoop** HDFS (Hadoop distributed file system) и в целом параллельных вычислений в компьютерных сетях.

Чтобы выяснить, каким образом обе эти перемены глубоко и положительно повлияли на возможности решения крупномасштабных задач, прежде всего следует выяснить, что на самом деле мешало (и по-прежнему мешает, в зависимости от того, насколько крупной является решаемая задача) выполнять анализ крупных наборов данных.

Независимо от того, какую задачу вы решаете, в конечном счете вы обнаружите, что не можете выполнить анализа своих данных в силу следующих ниже ограничений:

- вычислительная емкость оказывает влияние на время, затрачиваемое на выполнение анализа;
- емкость каналов ввода-вывода данных влияет на то, сколько данных может быть передано за единицу времени из хранилища данных в оперативную память машины;
- емкость оперативной памяти влияет на то, насколько большими будут данные, которые можно обработать за один раз.

Ваш компьютер имеет ограничения, которые будут определять, сможете ли вы обучиться на данных и сколько потребуется времени, прежде чем вы упретесь в стену. Вычислительные ограничения бывают во многих вычислительно емких расчетах, проблемы, связанные с вводом-выводом, образуют узкое место для быстрого доступа к данным, и, наконец, ограничения по памяти могут вынудить принимать лишь часть данных, тем самым ограничивая возможности матричных вычислений, к которым можно было бы обратиться, либо прецизионность или даже строгость получаемых оценок.

Каждое из этих аппаратных ограничений будет также оказывать разное влияние по степени серьезности относительно анализируемых данных:

- высокие данные, отличительная особенность которых состоит в том, что они имеют большое количество прецедентов;
- широкие данные, которые характерны тем, что имеют большое количество признаков;
- высокие и широкие данные, которые имеют одновременно большое количество прецедентов и признаков;
- разреженные данные, которые отличаются тем, что имеют большое количество нулевых записей или записей, которые можно преобразовать в нули (т. е. матрица данных может быть высокой и/или широкой, но информативной, при этом не все записи в матрице имеют информационное наполнение).

И в конце дело сводится к алгоритму, который вы собираетесь использовать, чтобы обучиться на данных. Каждый алгоритм имеет свои собственные свойства и способен преобразовывать данные, используя решение, на которое по-разному воздействует смещение или дисперсия. Следовательно, относительно задачи, которую вы до сих пор решали при помощи машинного обучения, вы рассчитывали, что определенные алгоритмы могут работать лучше других, основываясь при этом на своем опыте или эмпирической проверке. В случае с крупномасштабными задачами при выборе алгоритма необходимо добавить еще несколько совсем других соображений:

- какова вычислительная сложность алгоритма, т. е. влияет ли число строк и столбцов в данных на число вычислений линейным или нелинейным образом. Большинство решений в области машинного обучения основывается на алгоритмах квадратичной или кубической сложности, тем самым строго ограничивая их применимость к большим данным;
- сколько в модели параметров; здесь дело не просто в проблеме дисперсии оценок (переподгонке), а во времени, которое может потребоваться на их вычисление;
- можно ли параллелизировать процессы оптимизации, т. е. можно ли легко разделить вычисления между многочисленными узлами или ядрами CPU, или же приходится опираться на одиночный последовательный процесс оптимизации;
- должен ли алгоритм обучаться сразу на всех данных, или же вместо этого можно использовать одиночные примеры либо небольшие пакеты данных.

Если перекрестно оценить аппаратные ограничения и свойства данных, с одной стороны, и подобного рода алгоритмы – с другой, то можно получить множество возможных проблемных сочетаний, которые могут стать препятствием для полу-

чения результатов от проведения крупномасштабного анализа. С практической точки зрения все проблемные сочетания можно решить на основе трех подходов:

- вертикального масштабирования, т. е. улучшения производительности одиночной машины путем модификации программного обеспечения и/или оборудования (больше оперативной памяти, более быстрые CPU и дисковая память, а также использование модулей GPU);
- горизонтального масштабирования, т. е. распределения вычислений (и производительности) по многочисленным машинам с привлечением внешних ресурсов, а именно другой дисковой памяти и других модулей CPU (либо GPU);
- вертикального и горизонтального масштабирования, т. е. взятия лучшего решения из вертикальных и горизонтальных решений вместе взятых.

Приведение крупномасштабных примеров

Несколько мотивирующих примеров прояснит ситуацию и сделает ее запоминающейся.

Возьмем два простых примера:

- способность предсказывать **кликабельность** (click-through-rate, CTR), т. е. отношение числа щелчков к числу показов. В наши дни, когда интернет-реклама настолько распространилась вширь и вглубь, что отъедает значительные куски у традиционных СМИ, она помогает довольно много зарабатывать;
- способность предложить правильную информацию своим клиентам, когда они ищут предлагаемые вашим сайтом продукты и услуги, может по-настоящему улучшить ваши возможности их продавать, в случае если вы сможете угадывать, что помещать во главу результатов их поискового запроса.

В обоих случаях мы располагаем довольно большими наборами данных, которые продуцируются пользователями в результате их взаимодействия в Интернете.

В зависимости от бизнеса, который мы имеем в виду (тут можно вообразить некоторых крупных игроков), в обоих указанных случаях мы, очевидно, говорим о миллионах точек данных в день. В случае рекламы данные, разумеется, являются высокими, потому что они представляют собой непрерывный поток информации с заменой старых данных на новые, в большей мере отражающих рынки и потребителей. В случае поисковой системы данные являются широкими, дополняясь компонентом, предоставляемым результатами, которые вы предложили своим клиентам: например, если вы занимаетесь туристическим бизнесом, то у вас будет довольно много признаков об отелях, местах посещений и предлагаемых услугах.

Безусловно, масштабируемость создает трудности в обеих этих задачах:

- необходимо обучаться на данных, которые растут каждый день, и причем обучаться быстро, потому что, пока вы обучаетесь, продолжают поступать новые данные. При этом вам приходится иметь дело с данными, которые, очевидно, не смогут уместиться в оперативной памяти, потому что матрица слишком высокая или слишком большая;
- необходимо часто выполнять обновления модели машинного обучения с целью размещения новых данных. Для этого потребуется алгоритм, который может обрабатывать информацию в нужные сроки. Вычислительную сложность $O(n^2)$ или $O(n^3)$ практически невозможно обработать ввиду ко-

личества данных; потребуется такой алгоритм, который сможет работать с пониженной сложностью (такой как $O(n)$) или путем разделения данных, в результате которого n будет гораздо меньше;
- необходимо уметь предсказывать быстро, потому что предсказания должны предоставляться только новым клиентам. И опять-таки, здесь важную роль играет вычислительная сложность используемого алгоритма.

Проблема масштабируемости может быть решена одним или несколькими способами:
- вертикальное масштабирование путем снижения размерности задачи; например, в случае поисковой системы, путем эффективного отбора релевантных признаков для использования;
- вертикальное масштабирование с использованием приемлемого алгоритма; например, в случае рекламных данных, существуют соответствующие алгоритмы для эффективного обучения на потоках данных;
- горизонтальное масштабирование процесса обучения путем привлечения многочисленных машин;
- вертикальное масштабирование процесса внедрения с эффективным использованием многопроцессорной обработки и векторизации на одиночном сервере.

В настоящей книге мы покажем, какие практические задачи могут решаться каждым из этих предложенных решений или алгоритмов. В результате вы научитесь автоматически связывать отдельное ограничение по времени и исполнению (CPU, оперативная память или операции ввода-вывода) с самым подходящим решением среди тех, которые мы предлагаем.

Введение в язык Python

Поскольку наше исследование будет зависеть от языка Python – общедоступного языка программирования, выбранного в качестве базового для настоящей книги, – мы должны сделать короткую остановку, чтобы познакомиться с языком, прежде чем приступим к выяснению того, каким образом Python способен помочь легко масштабировать решение задачи с массивными данными вертикально и горизонтально.

Созданный в 1991 г. как общецелевой, интерпретируемый, объектно-ориентированный язык программирования, Python медленно, но верно завоевывал научное сообщество и в конечном итоге превратился в зрелую экосистему, состоящую из специализированных библиотек для обработки и анализа данных. Он позволяет проводить бесчисленные и быстрые эксперименты, легко разрабатывать программы, воплощающие теоретические предпосылки, и быстро развертывать научные приложения.

Как практик машинного обучения вы найдете использование Python интересным по нескольким причинам:
- он предлагает большую, зрелую систему библиотек для анализа данных и машинного обучения. Это гарантирует, что вы получите все, в чем вы, возможно, нуждаетесь в ходе анализа данных, и иногда даже больше;
- он очень универсален. Независимо от опыта или стиля программирования (объектно-ориентированный, функциональный или процедурный), программирование на Python доставит вам удовольствие;

- если вы еще с ним незнакомы, но хорошо знаете другие языки, такие как C/C++ или Java, то он очень прост в освоении и использовании. После того как вы уясните основы, лучший способ узнать больше – сразу же приступить к программированию;
- он является кросс-платформенным; ваши решения будут работать отлично и гладко в операционных системах Windows, Linux и Mac OS. Вам не придется беспокоиться о переносимости исходного кода;
- хотя он является интерпретируемым, его быстродействие, несомненно, выше, по сравнению с другими основными языками для анализа данных, такими как R и MATLAB (хотя он не сопоставим с C, Java и недавно появившимся языком Julia);
- он может работать с большими данными в оперативной памяти в силу минимального объема потребляемой оперативной памяти и превосходного управления ею. Сборщик «мусора» в оперативной памяти будет часто экономить уйму времени, когда вы будете загружать, преобразовывать, нарезать, группировать, сохранять или отбрасывать данные, используя различные циклы и повторные операции по подготовке данных к их анализу.

Если вы еще не являетесь экспертом в этом языке (на самом деле, чтобы иметь возможность использовать эту книгу максимальным образом, мы требуем лишь элементарных знаний языка), то всю информацию о языке можно почерпнуть (а также найти основные установочные файлы) непосредственно на главном сайте языка Python по адресу https://www.python.org/.

Вертикальное масштабирование средствами Python

Python – это интерпретируемый язык; он выполняет чтение вашего сценария из оперативной памяти и выполняет его динамически, тем самым получая доступ к необходимым ресурсам (файлам, объектам в памяти и т. п.). Помимо того что он – интерпретируемый, следует учитывать еще один важный аспект при использовании Python для анализа данных и машинного обучения – Python является однопоточным языком. Однопоточность означает, что любая программа на Python выполняется последовательно от начала до конца сценария и что Python не может использовать в своих интересах дополнительных вычислительных возможностей, предлагаемых многочисленными процессами и процессорами, которые могут иметься у компьютера (большинство компьютеров в наше время многоядерные).

Учитывая такую ситуацию, вертикальное масштабирование на основе Python может достигаться на основе разных стратегий:

- компиляция сценариев Python для достижения большей скорости исполнения. Несмотря на то что она легко осуществима, например при помощи **PyPy** – **динамического** (JIT) компилятора, который можно найти на http://pypy.org/, мы в нашей книге на самом деле к такому решению не обращались, потому что оно требует написания алгоритмов на Python с нуля;
- использование Python в качестве оберточного языка, тем самым связывая выполняемые на Python операции с выполнением внешних библиотек и программ, некоторые из которых приспособлены к многоядерной обработке. В нашей книге вы найдете много примеров, где вызываются специализированные библиотеки, в частности **библиотека для машин опорных векторов** LIBSVM (Library for Support Vector Machines), или такие програм-

мы, как **Vowpal Wabbit** (VW), **XGBoost** или **H2O**, для выполнения операций по машинному обучению;

- эффективное использование методов векторизации, т. е. специальных библиотек для выполнения матричных вычислений. Это может быть достигнуто при помощи библиотек **NumPy** или **pandas**, в которых задействуются вычисления с использованием модулей GPU. Модули GPU точно так же, как и многоядерные CPU, имеют свою собственную оперативную память и способны выполнять вычисления в параллельном режиме (как вы можете догадаться, они имеют многочисленные крошечные ядра). Методы векторизации на основе модулей GPU могут невероятно ускорить вычисления, в особенности во время работы с нейронными сетями. Однако GPU имеют свои собственные ограничения. Прежде всего располагаемая память имеет определенный канал ввода-вывода для передачи данных в память GPU и возврата результатов назад в CPU, и они требуют параллельного программирования посредством специального программного интерфейса (API), такого как **CUDA** для модулей GPU производства NVIDIA (отсюда следует, что требуется установка надлежащих драйверов и программ);
- сведение большой задачи к порциям и решение каждой порции поочередно в оперативной памяти (алгоритмы парадигмы «разделяй и властвуй»). Это влечет за собой разбиение данных из оперативной памяти или диска на части либо извлечение из них подвыборок и управление приближенными решениями задачи машинного обучения, что позволяет добиваться довольно эффективных результатов. Важно отметить, что разбиение на части и извлечение подвыборок может действовать как для прецедентов, так и для признаков (либо для обоих одновременно). Если исходные данные хранятся в дисковой памяти, то для итоговой производительности определяющими, безусловно, станут ограничения канала ввода-вывода;
- эффективное привлечение как многопроцессорной, так и многопоточной обработки в зависимости от используемого обучающегося алгоритма. Некоторые алгоритмы естественным образом могут разбивать свои операции на параллельные. В таких случаях единственным ограничением будут CPU и оперативная память (так как данные должны быть реплицированы для каждого параллельного рабочего процесса, который будет использоваться). Некоторые другие алгоритмы вместо этого используют преимущества многопоточной обработки, тем самым одновременно управляя бóльшим числом операций на тех же блоках памяти.

Горизонтальное масштабирование средствами Python

Горизонтальное масштабирование решений состоит в объединении многочисленных машин в кластер. При подключении машин (т. е. масштабируя вширь) можно также масштабировать каждую из них вертикально (т. е. вглубь), используя более мощные конфигурации (тем самым усиливая CPU, оперативную память и каналы ввода-вывода), применяя методы, упомянутые нами в предыдущем абзаце, и улучшая их производительность.

Подключая многочисленные машины, можно мобилизовать их вычислительную мощь в параллельном виде. Ваши данные будут распределены по многочисленным дисковым хранилищам/устройствам памяти, тем самым ограничивая

число операций передачи данных по каналу ввода-вывода и заставляя каждую машину обрабатывать только имеющиеся у нее данные (т. е. свое собственное дисковое хранилище или оперативную память).

В нашей книге это транслируется в эффективное использование внешних ресурсов посредством следующих платформ:

- платформы H2O;
- платформы Hadoop и ее компонентов, таких как распределенная файловая система HDFS, вычислительная парадигма MapReduce и менеджер ресурсов **YARN**;
- платформы Spark поверх Hadoop.

Python будет управлять всеми этими платформами (например, платформа Spark управляется ее Python'овским интерфейсом под названием pySpark).

Python для крупномасштабного машинного обучения

С учетом наличия многих удобных библиотек для машинного обучения и того факта, что этот язык программирования довольно популярен среди исследователей-аналитиков, Python был выбран в качестве базового языка для написания всего представленного в настоящей книге исходного кода.

В этой книге, по мере необходимости, мы будем предоставлять дальнейшие инструкции по установке других необходимых библиотек или инструментов. Здесь же вместо этого мы приступим к установке самых главных компонентов, т. е. языка Python и наиболее часто используемых библиотек, применяемых для вычислений и машинного обучения.

Выбор между Python 2 и Python 3

Перед тем как начинать, важно уяснить, что существуют две основные ветви Python: версии 2 и 3. Поскольку в обеих версиях много базовой функциональности изменилось, сценарии, созданные для одной версии, иногда несовместимы с другой (они будут работать, вызывая сообщения об ошибках и предупреждения). Несмотря на то что третья версия является новейшей, более старая по-прежнему широко используется в научной среде и является версией по умолчанию для многих действующих систем (в основном для совместимости при обновлениях). Когда версия 3 была выпущена (в 2008 г.), большинство научных библиотек еще не было готово, и поэтому научное сообщество продолжило использовать предыдущую версию. К счастью, с тех пор почти все библиотеки были обновлены, за исключением лишь некоторых (см. http://py3readiness.org по поводу обзора совместимости), оставшихся несовместимыми с Python 3.

Несмотря на недавний рост популярности Python 3 (который продолжит свое развитие в будущем, и мы не должны это забывать), Python 2 все еще широко используется среди исследователей и аналитиков. Кроме того, в течение длительного времени Python 2 устанавливался по умолчанию (например, в Ubuntu), таким образом, эта версия будет наиболее вероятной, которую большинство читателей будет иметь под рукой. Исходя из всех этих причин, для этой книги мы примем за основу Python 2. Это решение вызвано не какой-то *любовью к старым технологиям*, а попросту является практичным выбором в пользу того, чтобы

сделать книгу «*Крупномасштабное машинное обучение на Python*» доступной для большей аудитории:

- исходный код на Python 2 сразу охватит всю существующую аудиторию экспертов в области данных;
- пользователи Python 3 очень легко смогут преобразовать наши сценарии для работы в версии Python, которую они предпочитают, потому что написанный нами исходный код легко конвертируем.

В случае если есть необходимость подробно разобраться в различиях между Python 2 и Python 3, мы предлагаем прочитать эту веб-страницу о написании совместимого между Python 2–3 исходного кода: http://python-future.org/compatible_idioms.html. Кроме того, на веб-сайте **Python-Future** можно найти полезные сведения о том, как преобразовать исходный код Python 2 в Python 3: http://python-future.org/automatic_conversion.html.

Инсталляция среды Python

В качестве первого шага мы создадим рабочую среду для науки о данных, которую можно использовать для репликации и тестирования примеров из книги и создания прототипов собственных больших решений.

Независимо от того, на каком языке вы собираетесь разрабатывать свое приложение, с Python вы не будете испытывать трудностей при получении своих данных, создании из них модели и извлечении правильных параметров, которые необходимы для выполнения прогнозов в эксплуатационной среде.

Python – это общедоступный объектно-ориентированный кросс-платформенный язык программирования, который, по сравнению с его прямыми конкурентами (например, C/C++ и Java), производит очень сжатый и читаемый код. Это позволяет создавать действующий прототип программного обеспечения в очень короткие сроки и тестировать, поддерживать и масштабировать его в будущем. Он завоевал статус наиболее используемого языка среди программного инструментария исследователя-аналитика, потому что, будучи языком общего назначения, он в конечном итоге приобрел гибкость благодаря большому разнообразию имеющихся библиотек, которые способны легко и быстро помочь в решении самого широкого спектра как распространенных, так и нишевых задач.

Пошаговая установка

Если вы никогда не использовали Python (однако это не исключает того, что он на вашей машине может быть уже установлен), то сначала необходимо скачать установщик с основного веб-сайта проекта https://www.python.org/downloads/ (напомним, что мы используем версию 2) и затем установить его на локальной машине.

Этот раздел предоставляет вам полный контроль над тем, что может быть установлено на вашей машине. Это очень удобно, если вы собираетесь использовать Python одновременно как язык для прототипирования и программирования промышленного кода. Кроме того, этот раздел поможет отслеживать используемые версии библиотек. Так или иначе, предупреждаем, что пошаговая установка в действительности потребует некоторого времени и усилий. С другой стороны, установка готового научного дистрибутива облегчит бремя инсталляционной процедуры и хорошо подойдет для первого раза и обучения, потому что экономит довольно много времени, хотя и установит на ваш компьютер сразу большое ко-

личество библиотек (которые вы по большей части, возможно, никогда не будете использовать). Поэтому если вы хотите начать немедленно и не желаете слишком уж беспокоиться по поводу контроля над инсталляцией, то просто пропустите эту часть и перейдите к следующему разделу «*Научные дистрибутивы*».

Поскольку Python является мультиплатформенным языком программирования, вы найдете установщики для компьютеров, которые работают под управлением операционной системы Windows либо Linux-/Unix-подобных ОС. Напомним, что в некоторых дистрибутивах Linux (таких как Ubuntu) язык Python 2 уже включен в репозиторий по умолчанию, что делает процесс установки еще проще.

1. Откройте оболочку Python, введите python в терминале или щелкните по значку Python.
2. Затем для тестирования результата инсталляции выполните следующие команды в интерактивной оболочке Python, или ее стандартной интерактивной среде программирования **REPL** (от англ. Read-Eval-Print Loop, работающей в цикле чтения-вычисления-печати результата), или других решениях, таких как Spyder или PyCharm:

```
>>> import sys
>>> print(sys.version)
```

Если была поднята синтаксическая ошибка, то, значит, вы выполняете Python 3 вместо Python 2. Если же ошибка отсутствует и вы можете прочитать, что используемой версией является Python 2.7.x (во время написания книги последней была версия 2.7.12), то поздравляем с установкой версии Python, которую мы выбрали для этой книги. Для тех же читателей, кто работает с Python 3, подойдут все версии начиная с Python 3.4 (во время написания последней была версия 3.5.2).

Напоминаем, что, когда команда выдается в командой строке терминала, она предваряется префиксом `$`. В противном случае, если речь о стандартной среде Python REPL, ей предшествует подсказка `>>>`.

Установка библиотек

В зависимости от вашей системы и предыдущих инсталляций среда Python может не оказаться укомплектованной всем тем, в чем вы нуждаетесь, если не установлен дистрибутив (который, с другой стороны, обычно укомплектован гораздо шире, чем вам, возможно, понадобится).

Чтобы установить любую нужную библиотеку, можно применить команду `pip` или `easy_install`; однако в будущем поддержка команды `easy_install` будет прекращена, и **pip** имеет перед ней важные преимущества.

Инструмент установки библиотек Python **pip** непосредственно получает доступ к Интернету и выбирает их из каталога библиотек Python **PyPI** (https://pypi.python.org/pypi). PyPI представляет собой репозиторий, содержащий сторонние библиотеки с открытым исходным кодом, которые постоянно поддерживаются в работоспособном состоянии и сохраняются в репозитории их авторами.

Устанавливать библиотеки лучше всего при помощи `pip` по следующим причинам:

- он является предпочтительным диспетчером библиотек Python и начиная с Python 2.7.9 и Python 3.4 по умолчанию включен в двоичные установщики Python;

- он обеспечивает функциональность по деинсталляции библиотек;
- он возвращает вашу систему в исходное состояние и оставляет ее чистой, если по какой-либо причине установленная библиотека перестала работать.

Команда `pip` работает в командной строке и ускоряет весь процесс установки, обновления и удаления библиотек Python.

Как уже упоминалось, если вы работаете как минимум с Python 2.7.9 или Python 3.4, то команда `pip` должна уже иметься в наличии. Чтобы удостовериться в том, какие инструменты были установлены на локальной машине, выполните прямую проверку на возможные ошибки при помощи следующей команды:

```
$ pip -V
```

В некоторых инсталляциях в Linux и Mac OS устанавливается Python 3, а не Python 2, в результате чего эта команда может присутствовать как `pip3`, поэтому при получении ошибки при поиске `pip` попробуйте выполнить следующую команду:

```
$ pip3 -V
```

Если это так, то напомним, что `pip3` подходит только для установки библиотек в Python 3. Поскольку в книге мы работаем с Python 2 (если, разумеется, вы не решили использовать новейшую версию Python 3), то вашим основным установщиком библиотек всегда будет `pip`.

Как вариант можно также проверить доступность старой команды `easy_install`:

```
$ easy_install --version
```

Использовать `easy_install` вместо `pip` целесообразно, если вы работаете под Windows, потому что `pip` не устанавливает двоичных библиотек; поэтому если при установке библиотеки вы испытываете неожиданные трудности, то `easy_install` может сэкономить вам уйму времени.

Если проверка закончилась ошибкой, то вам действительно необходимо установить `pip` с нуля (и при этом также `easy_install`).

Для установки `pip` просто следуйте инструкциям на https://pip.pypa.io/en/stable/installing/. Самый безопасный путь состоит в том, чтобы скачать сценарий `get-pip.py` по прямой ссылке с https://bootstrap.pypa.io/get-pip.py и затем выполнить его при помощи следующей команды:

```
$ python get-pip.py
```

Между прочим, этот сценарий также установит настроечный инструмент `setuptools` с https://pypi.python.org/pypi/setuptools, который содержит `easy_install`.

В качестве альтернативы, если вы работаете под управлением Unix-подобной операционной системы Debian/Ubuntu, укороченный способ установки всего, что нужно, будет состоять в команде `apt-get`:

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

После проверки этого основного требования теперь все готово к установке всех библиотек, которые потребуются для выполнения примеров, прилагаемых к настоящей книге. Чтобы установить типовую библиотеку `<lib>`, нужно просто выполнить следующую команду:

```
$ pip install <lib>
```

Как вариант, если вы предпочитаете использовать команду easy_install, можно также выполнить следующую команду:

```
$ easy_install <lib>
```

После этого библиотека <lib> и все библиотеки, от которых она зависит, будут скачаны и установлены.

Если вы не уверены в том, была библиотека установлена или нет, просто попробуйте импортировать из нее модуль. Если интерпретатор Python поднимает ошибку импорта **ImportError**, то можно сделать вывод, что библиотека не была установлена.

Посмотрим на примере. Вот что происходит, когда библиотека NumPy была установлена:

```
>>> import numpy
>>>
```

А вот что – если она не установлена:

```
>>> import numpy
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named numpy
```

В последнем случае, прежде чем ее импортировать, надо установить библиотеку посредством pip или easy_install.

Позаботьтесь о том, чтобы не перепутать библиотеки с модулями. При помощи pip вы устанавливаете библиотеку, а в среду Python вы импортируете модуль. Иногда библиотека и модуль имеют одинаковое название, но во многих случаях они не совпадают. Например, модуль **sklearn** включен в библиотеку **Scikit-learn**.

Способы обновления библиотек

Как правило, вы окажетесь в ситуации, когда необходимо обновить библиотеку, потому что некая связанная с ней другая библиотека, т. н. зависимость, требует наличия новой версии, либо имеется дополнительный функционал, который требуется задействовать. Для этого сначала нужно проверить версию установленной библиотеки, обратившись к атрибуту __version__, как показано на примере с библиотекой NumPy ниже:

```
>>> import numpy
>>> numpy.__version__ # 2 символа подчеркивания перед ним и после него
'1.9.0'
```

Далее, если нужно обновить ее до более нового выпуска, скажем, в точности до версии 1.9.2, то из командной строки можно выполнить следующую ниже команду:

```
$ pip install -U numpy==1.9.2
```

Как вариант (но мы его не рекомендуем, только в случае, если это оказывается необходимым), можно также использовать следующую команду:

```
$ easy_install --upgrade numpy==1.9.2
```

Наконец, если вы попросту заинтересованы в ее обновлении до последней доступной версии, то просто выполните следующую команду:

```
$ pip install -U numpy
```

Можно также выполнить альтернативную команду `easy_install`:

```
$ easy_install --upgrade numpy
```

Научные дистрибутивы

Из того, что вы прочли до сих пор, следует, что работа по созданию рабочей среды занимает у исследователя-аналитика много времени. Сначала нужно установить Python и затем, одну за другой, можно установить все библиотеки, в которых вы будете нуждаться. (Иногда инсталляционная процедура может пойти не так гладко, как вы надеялись.)

Если вы хотите сэкономить время и усилия и гарантированно получить полностью рабочую, готовую к использованию среду Python, то можно просто скачать, установить и использовать научный дистрибутив Python. Помимо языка Python, такие дистрибутивы также содержат множество предварительно установленных библиотек, и иногда в них даже имеются для использования дополнительные инструменты и интегрированные среды разработки (IDE). Несколько из них хорошо известно среди исследователей-аналитиков, и в последующих разделах вы познакомитесь с некоторыми главными особенности двух таких комплектов программного обеспечения, которые мы сочли самыми полезными и практичными.

Чтобы немедленно сосредоточиться на содержимом настоящей книги, мы предлагаем сначала быстро скачать и установить научный дистрибутив, в частности **Anaconda** (который, по нашему мнению, является самым полным из всех), и затем после выполнения примеров из этой книги полностью его деинсталлировать, чтобы установить только один Python, дополнив его лишь теми библиотеками, в которых вы нуждаетесь для разработки своих проектов.

И опять-таки, если это возможно, скачайте и установите версию дистрибутива, содержащую Python 2.

В качестве первого комплекта программного обеспечения мы рекомендуем попробовать дистрибутив Python Anaconda (https://www.continuum.io/downloads), предлагаемый компанией Continuum Analytics, который включает почти 200 библиотек, в т. ч. NumPy, SciPy, pandas, IPython, matplotlib, Scikit-learn и StatsModels. Это кросс-платформенный дистрибутив, который можно установить на машины с другими существующими дистрибутивами и версиями Python, а его базовая версия бесплатна. Дополнительные надстройки, которые содержат расширенные возможности, оплачиваются отдельно. Anaconda представляет двоичный диспетчер библиотек **conda** как инструмент командной строки для управления установкой библиотек. Как утверждается на веб-сайте дистрибутива, задача Anaconda состоит в том, чтобы обеспечить дистрибутив Python, готовый к работе на уровне предприятия, для крупномасштабной обработки данных, прогнозной аналитики и научных вычислений. Что касается версии 2.7 Python, то мы рекомендуем как минимум дистрибутив Anaconda 4.0.0. (Чтобы взглянуть на устанавливаемые вместе с Anaconda библиотеки, следует обратиться к списку на https://docs.continuum.io/anaconda/pkg-docs.)

Второе предложение – дистрибутив **WinPython** (http://winpython.sourceforge.net/). Этот дистрибутив может оказаться довольно интересной альтернативой, если вы работаете под Windows и желаете, чтобы ваш дистрибутив был переносимым (извините, но версии под Linux и Mac OS отсутствуют). WinPython – это тоже бесплатный дистрибутив Python с открытым исходным кодом, который поддерживается сообществом. И он тоже разработан, имея в виду исследователей-аналитиков, и включает в себя комплект основных библиотек, таких как NumPy, SciPy, matplotlib и IPython (в основном те же, что и в Anaconda). Он так же, как и Anaconda, содержит инструментальную среду разработки **Spyder**, которая может быть полезной, если у вас есть опыт работы в интерфейсе языка MATLAB. Решающее преимущество данного дистрибутива состоит в том, что он переносим (его можно поместить в любом каталоге или даже на флеш-накопитель USB), тем самым на компьютере могут существовать различные версии, их можно перемещать с одного компьютера Windows на другой, и более старая версия дистрибутива легко заменяется на более новую всего лишь путем смены каталога. При выполнении WinPython или его оболочки он автоматически установит все необходимые для выполнения Python переменные окружения так, как будто он был инсталлирован в регулярном режиме и зарегистрирован в системе.

Во время написания настоящей книги последним по времени дистрибутивом WinPython для версии Python 2.7 был созданный в октябре 2015 г. релиз 2.7.10; с тех пор публиковались обновления дистрибутива WinPython только для версии Python 3. После установки дистрибутива в операционной системе, возможно, потребуется обновить некоторые ключевые библиотеки, необходимые для выполнения примеров из этой книги.

Введение в Jupyter

Бесплатный проект IPython был запущен Фернандо Пересом в 2001 г. с целью решения проблемы отсутствия в Python стека научных исследований с использованием пользовательского программного интерфейса, который мог бы включать в процесс разработки программного обеспечения научный подход (главным образом экспериментирование и интерактивный поиск).

Научный подход подразумевает проведение быстрых экспериментов в отношении разных гипотез в воспроизводимой форме (подобно задаче разведочного анализа в науке о данных). Используя IPython во время написания своего исходного кода, вы сможете реализовывать разведочные, итеративные и эмпирические (путем проб и ошибок) исследования более естественным образом.

В конце 2015 г. значительная часть проекта IPython переместилась в новый проект под названием **Jupyter** (http://jupyter.org/). Этот новый проект расширяет потенциальное удобство использования исходного интерфейса IPython до широкого спектра языков программирования[1]. (Для получения полного списка языков программирования посетите https://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages).

Благодаря мощной идее ядер языков программирования специальный внешний интерфейс позволяет передавать в них программы, которые выполняют

[1] Далее в книге все упоминания интерактивной среды IPython заменены на Jupyter, являющуюся более общей и более современной интерактивной средой программирования, куда IPython входит в качестве главной составной части (как нулевое ядро). – *Прим. перев.*

пользовательский программный код, и в качестве отклика получать результат исполненного исходного кода; вы можете пользоваться единым интерфейсом и интерактивным стилем программирования независимо от того, на каком языке пишется программа.

Jupyter (IPython является нулевым ядром, запускаемым с самого начала) можно описать как инструмент для выполнения интерактивных задач, пригодных для работы в консоли или веб-ориентированном блокноте, который предлагает специальные команды, помогающие разработчикам лучше понимать и создавать исходный код, пишущийся в настоящий момент.

В отличие от интегрированной среды разработки (IDE), которая строится вокруг идеи написания сценария, его последующего выполнения и оценки его результатов, среда программирования Jupyter позволяет писать свой исходный код порциями, именуемыми **ячейками**, последовательно выполнять каждую из них и оценивать результаты каждой ячейки отдельно, исследуя одновременно текстовые и графические результаты. Помимо графической интеграции, эта среда предоставляет дополнительную помощь благодаря настраиваемым командам, богатой истории (в формате JSON) и вычислительному параллелизму для повышения производительности во время работы с тяжелыми численными вычислениями.

Такой подход также в особенности продуктивен для заданий, связанных с разработкой исходного кода на основе данных, поскольку он автоматически реализует часто пренебрегаемую обязанность по документированию и иллюстрированию того, каким образом анализ данных был проведен, его предпосылок и допущений и его промежуточных и конечных результатов. Если часть вашего задания, кроме того, состоит в презентации работы с целью убедить в проекте внутренние или внешние заинтересованные стороны, то Jupyter может действительно совершать волшебство, выполняя презентацию за вас, при этом требуя лишь небольших дополнительных усилий. На https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks имеется множество примеров использования блокнотов, некоторые из которых могут вдохновить вас в вашей работе, как это вышло у нас.

На самом деле мы должны признаться, что поддержание чистого, актуального блокнота Jupyter сэкономило нам неисчислимое количество времени, когда неожиданно всплыли незапланированные встречи с менеджерами/заинтересованными сторонами, которые потребовали от нас торопливо представить состояние нашей работы.

Одним словом, Jupyter предлагает вам следующие возможности:

- видеть промежуточные результаты (и их отладку) для каждого шага анализа;
- выполнять только некоторые разделы (или ячейки) с исходным кодом;
- хранить промежуточные результаты в формате JSON и иметь возможность выполнять на них версионный контроль;
- выполнять презентацию работы (в виде комбинации текста, исходного кода и изображений), делясь ею при помощи средства просмотра блокнотов Jupyter Notebook Viewer (http://nbviewer.jupyter.org/) и легко экспортируя его в PY, HTML, PDF или даже в слайдовые презентации.

Блокноты Jupyter являются нашим предпочтительным выбором на протяжении всей книги. Они используются для ясной и эффективной иллюстрации операций, связанных с изложением материала, на основе сценариев и данных, и последующих результатов.

Хотя мы рекомендуем использовать исключительно Jupyter, если вы используете среду интерпретатора REPL или IDE, то вы можете использовать те же самые инструкции и ожидать идентичных результатов (за исключением форматов и расширений оператора печати для возвращаемых результатов).

Если в вашей операционной системе Jupyter не установлен, то его можно быстро установить, воспользовавшись следующей командой:

```
$ pip install jupyter
```

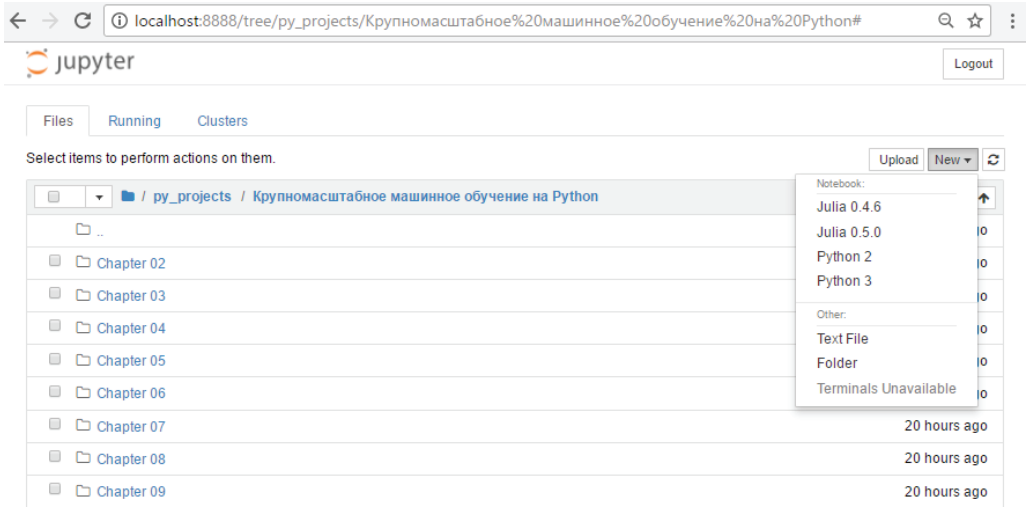
Полные инструкции по установке Jupyter (с охватом разных операционных систем) можно найти на http://jupyter.readthedocs.io/en/latest/install.html.

Если Jupyter уже установлен, то он должен быть обновлен как минимум до версии 4.1.

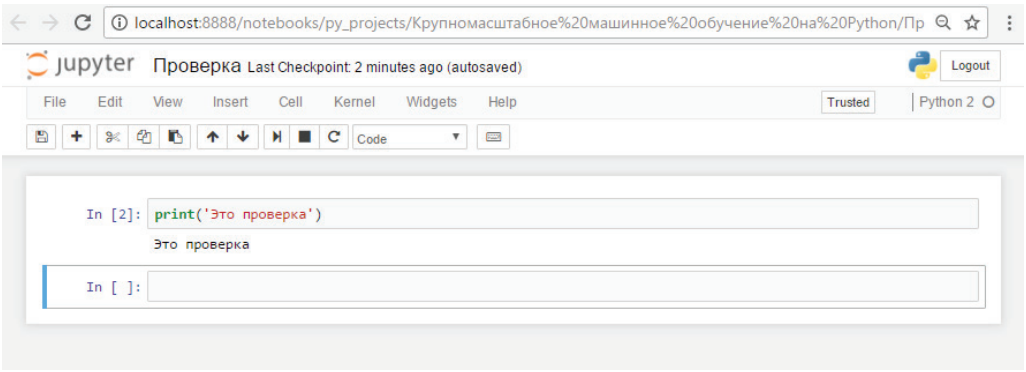
После установки можно сразу начать использовать Jupyter, вызвав его из командной строки:

```
$ jupyter notebook
```

Открыв экземпляр Jupyter в браузере, щелкните по кнопке **New** и в разделе блокнотов Notebooks выберите **Python 2** (в разделе могут присутствовать другие ядра, в зависимости от того, что установлено):



В этом месте новый пустой блокнот будет похож на следующий ниже снимок экрана, и можно приступать к вводу команд в ячейки:



Например, можно набрать в ячейку следующее:

```
In:
print («Это проверка»)
```

Набрав в ячейку исходный код, просто нажмите кнопку воспроизведения (под вкладкой **Cell**), чтобы его выполнить и получить результат. Ниже под ячейкой с введенным кодом появится еще одна ячейка. Если при заполнении ячеек нажать кнопку с плюсом в верхней строке меню, то вы получите новую ячейку, при этом перемещаться от ячейки к ячейке можно при помощи стрелок в меню.

Большинство других функций вполне интуитивно понятно, и мы приглашаем вас их попробовать. Чтобы лучше познакомиться с тем, как работает Jupyter, можно воспользоваться кратким практическим руководством, в частности http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/, либо получить специализированную книгу с описанием функционала Jupyter.

Для полного изложения всего спектра функциональных возможностей Jupyter при выполнении ядра IPython обратитесь к следующим двум книгам издательства Packt Publishing:

- *IPython Interactive Computing and Visualization Cookbook* («Книга рецептов по интерактивным вычислениям и визуализации в IPython»), автор Сирилл Россан (Cyrille Rossant), сентябрь 2014;
- *Learning IPython for Interactive Computing and Data Visualization* («Изучение IPython для интерактивных вычислений и визуализации данных»), автор Сирилл Россан, апрель 2013.

Следует учитывать, что для большей наглядности каждый блок инструкций Jupyter имеет пронумерованный оператор ввода и оператор вывода, поэтому в этой книге вы увидите, что исходный код структурирован в двух блоках, за исключением случаев, когда результат на выходе тривиален, – в этом случае ожидайте увидеть всего один входной оператор:

```
In: <вводимый исходный код>
Out: <полученный итоговый результат>
```

Как правило, вам придется лишь набирать исходный код в ячейки после оператора **In:** и выполнять его. Затем вы будете сравнивать полученные вами выходные данные с результатом в книге после оператора **Out:**, который мы фактически получили на наших компьютерах, когда тестировали приводимый исходный код.