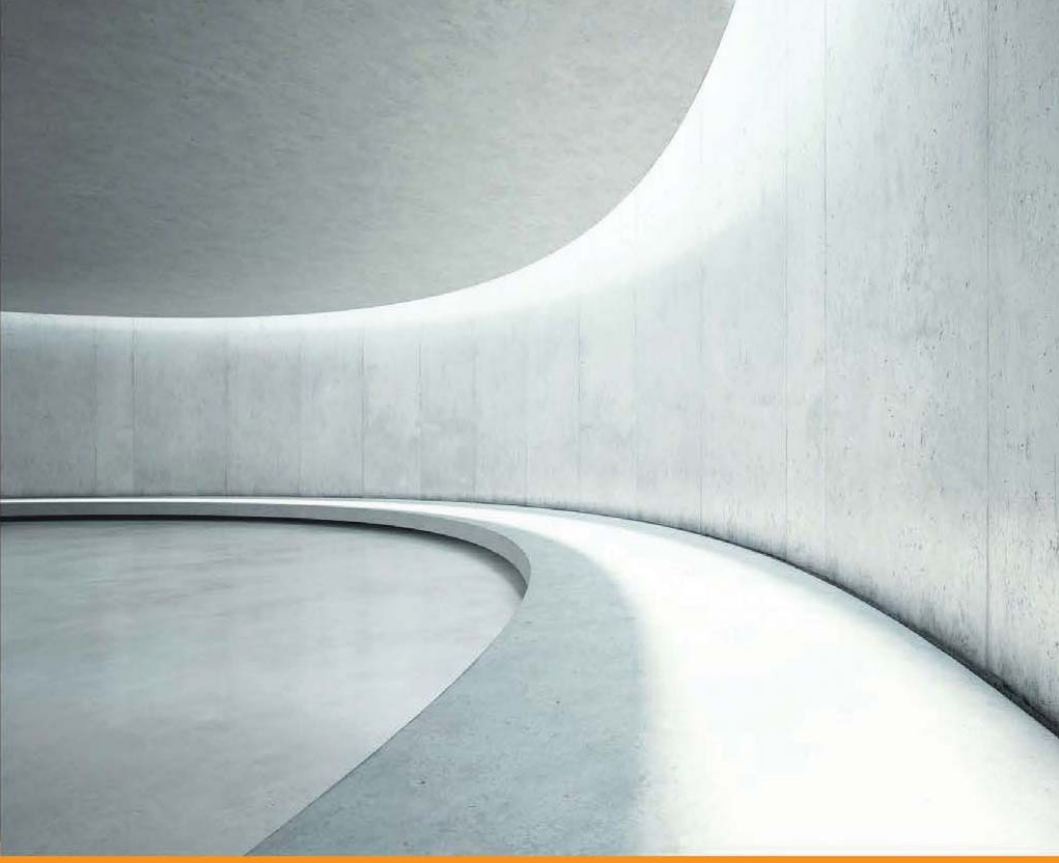


Построение систем машинного обучения на языке Python

Луис Педро Коэльо Вилли Ричарт

ДМК издательство

УДК 004.438Python:004.6
ББК 32.973.22
К76

К76 Луис Педро Коэльо, Вилли Ричарт

Построение систем машинного обучения на языке Python. 2-е издание / пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2016. – 302 с.: ил.

ISBN 978-5-97060-330-7

Применение машинного обучения для лучшего понимания природы данных – умение, необходимое любому современному разработчику программ или аналитику. Python – замечательный язык для создания приложений машинного обучения. Благодаря своей динамичности он позволяет быстро производить разведочный анализ данных и экспериментировать с ними. Обладая первоклассным набором библиотек машинного обучения с открытым исходным кодом, Python дает возможность сосредоточиться на решаемой задаче и в то же время опробовать различные идеи.

Книга начинается с краткого введения в предмет машинного обучения и знакомства с библиотеками NumPy, SciPy, scikit-learn. Но довольно быстро авторы переходят к более серьезным проектам с реальными наборами данных, в частности, тематическому моделированию, анализу корзины покупок, облачным вычислениям и др.

Издание рассчитано на программистов, пишущих на Python и желающих узнать о построении систем машинного обучения и научиться извлекать из данных ценную информацию, необходимую для решения различных задач.

ISBN 978-1-78439-277-2 (англ.)
ISBN 978-5-97060-330-7 (рус.)

ОГЛАВЛЕНИЕ

ГЛАВА 1. Введение в машинное обучение на языке Python

Машинное обучение – это наука о том, как научить машину самостоятельно решать задачи. Вот так все просто. Но дьявол кроется в деталях, и именно поэтому вы читаете эту книгу.

Быть может, данных слишком много, а априорных знаний о них слишком мало. И вы надеетесь, что алгоритмы машинного обучения помогут справиться с этой проблемой, а потому начинаете в них разбираться. Но через некоторое время впадаете в ступор: какой же из мириада алгоритмов выбрать?

Или, быть может, вас заинтересовали общие вопросы машинного обучения, и вы начали читать блоги и статьи на эту тему. Все это показалось вам таким крутым и волшебным, что вы приступили к собственным исследованиям и загрузили простенькие данные в решающее дерево или в машину опорных векторов. Но вот эксперимент завершился, и возникает вопрос: а правильно ли он был поставлен? Насколько оптимальны полученные результаты? И как узнать, не существует ли алгоритма получше? Да и вообще – правильно ли были подобраны данные?

Вы не одиноки! Мы оба (авторы) проходили через это – искали сведения о том, как на практике выглядит то, о чем пишут в учебниках по машинному обучению. Оказалось, что очень многое – «черная магия», о которой авторы стандартных учебников забывают упомянуть. Так что в некотором роде эта книга написана нами для нас же, только на несколько лет моложе. Здесь вы найдете не только краткое введение в методы машинного обучения, но и уроки, которые мы извлекли, идя по этому пути. Мы надеемся, что благодаря этой книге дорога, ведущая в самые захватывающие области Информатики, станет чуть более гладкой.

Машинное обучение и Python – команда мечты

Цель машинного обучения – научить машину (точнее, программу) решать задачу, предъявив ей несколько примеров (с правильными и неправильными решениями). Предположим, что каждое утро, включив компьютер, вы делаете одно и то же: сортируете пришедшую почту, раскладывая письма в папки по темам. Через какое-то время эта работа вам надоест, и вы захотите ее автоматизировать. Один из возможных подходов – проанализировать собственное поведение и выписать правила, которыми вы руководствуетесь при сортировке писем. Однако это громоздкое и отнюдь не совершенное решение. Некоторые правила вы упустите из виду, другие сформулируете излишне детально. Гораздо лучше выбрать какой-то набор метаданных о письмах, задать пары (тело письма, имя папки) и поручить алгоритму вывести наилучший набор правил. Такие пары называются обучающими данными, а получившийся набор правил можно будет применить к будущим письмам, которых алгоритм еще не видел. Это и есть машинное обучение в простейшем виде.

Разумеется, машинное обучение (его еще часто называют добычей данных или прогностическим анализом) – не новая дисциплина. Наоборот, своими успехами в последние годы она обязана практическому применению проверенных временем методов и идей из других областей знания, в частности математической статистики. Цель человека – извлечь полезную информацию из данных, например, выявить скрытые закономерности и взаимосвязи. Читая об успешных применениях машинного обучения (Вы ведь уже открыли для себя сайт `www.kaggle.com`, правда?), вы убедитесь, что прикладная статистика вовсю используется специалистами.

Ниже вы увидите, что процесс поиска подходящего подхода к машинному обучению вовсе не линеен. Напротив, приходится многократно возвращаться назад, пробуя другие сочетания исходных данных и алгоритмов. Именно изыскательская природа этого процесса делает применение Python чрезвычайно уместным. Ведь Python, будучи интерпретируемым высокоуровневым языком программирования, как будто специально придуман для опробования разных вариантов. К тому же, он работает быстро. Конечно, он медленнее C и подобных ему статически типизированных языков. Но при наличии огромного числа простых в использовании библиотек, зачастую написанных на C, вам не придется жертвовать скоростью ради гибкости.

Что вы узнаете (и чего не узнаете) из этой книги

В этой книге вы найдете общий обзор обучающих алгоритмов, которые чаще всего применяются в различных отраслях машинного обучения, и узнаете, на что обращать внимание при их применении. Но по своему опыту мы знаем, что такие «интересные» вещи, как использование и настройка отдельных алгоритмов, например метода опорных векторов, классификации по ближайшим соседям, или их ансамблей, – занимают лишь малую часть рабочего времени специалиста по машинному обучению. А основное время тратится на довольно скучную работу:

- чтение и очистка данных;
- изучение исходных данные и попытки понять их;
- размышления о том, как лучше подать данные на вход алгоритма обучения;
- выбор подходящей модели и алгоритма;
- правильное измерение качества работы алгоритма.

В процессе изучения и осмысления исходных данных нам понадобится статистика и не очень сложная математика. И, как вы убедитесь, методы, казавшиеся такими скучными на занятиях по математике, могут стать по-настоящему увлекательными, когда применяются для анализа интересных данных.

Наше путешествие начинается с чтения данных. Вы поймете, что поиск ответа на вопрос, как быть с некорректными или отсутствующими данными, – скорее искусство, чем точная наука. Правильное решение воздастся сторицей, потому что позволит применить к данным больше обучающих алгоритмов, а, значит, повысит шансы на успех.

Когда данные будут представлены в виде структур данных в программе, вы захотите понять, с чем же все-таки работаете. Достаточно ли данных для ответа на интересующие вас вопросы? Если нет, то как добыть дополнительные данные? А, быть может, данных слишком много? Тогда нужно подумать, как лучше всего произвести выборку из них.

Часто бывает, что данные не стоит подавать сразу на вход алгоритма машинного обучения, а надо предварительно улучшить их. Алгоритм с благодарностью ответит на это более качественными результатами. Иногда даже оказывается, что простой алгоритм на предварительно обработанных данных работает лучше, чем очень изощренный алгоритм на данных в первозданном виде. Эта часть

работы, называемая **подготовкой признаков** (feature engineering), чаще всего оказывается безумно увлекательной задачей, приносящей немедленные плоды. Вы сразу же видите результаты творческого нестандартного подхода.

Таким образом, чтобы выбрать подходящий обучающий алгоритм, недостаточно просто ткнуть наугад в один из трех-четырех имеющихся в вашем арсенале (как вы скоро увидите, их вообще-то больше). Это вдумчивый процесс взвешивания различных критериев – от качества работы до функциональных требований. Вы хотите получить результат быстро, даже в ущерб качеству? Или предпочитаете потратить больше времени, но получить наилучший возможный результат? У вас есть отчетливое представление о будущих данных или лучше бы не делать слишком ограничительных предположений на этот счет?

Наконец, измерение качества работы – то место, где начинающий изучать машинное обучение может наделать больше всего ошибок. Есть ошибки простые, например, контроль результатов на тех же данных, на которых производилось обучение. А есть и посложнее, например, подача на вход несбалансированного обучающего набора. Как и раньше, именно данные определяют успех или неудачу всего предприятия.

Как мы видим, лишь четвертый пункт в нашем списке относится собственно к возне с алгоритмами. И тем не менее, мы надеемся убедить вас в том, что и остальные четыре задачи могут быть не менее увлекательными. Нам хотелось бы, чтобы, прочитав книгу до конца, вы влюбились не столько в алгоритмы обучения, сколько в сами данные.

Поэтому мы не станем нагружать вас теоретическими вопросами разнообразных алгоритмов машинного обучения, поскольку на эту тему существует немало отличных книг (их перечень вы найдете в приложении), а вместо этого попытаемся развить интуицию настолько, чтобы вы поняли идею и смогли сделать первые шаги. Так что эту книгу ни в коем случае нельзя считать *авторитетным руководством* по машинному обучению. Это скорее учебник для начинающих. Мы надеемся возбудить в вас любопытство настолько, чтобы вам захотелось продолжить изучение этой интереснейшей области знаний.

Далее в этой главе мы познакомимся с основами библиотек NumPy и SciPy для Python и обучим наш первый алгоритм с помощью библиотеки scikit-learn. По ходу дела мы введем основные понятия машинного обучения, которые будут использоваться на протяжении всей книги. В последующих главах мы подробно рассмотрим все

пять вышеперечисленных шагов, демонстрируя различные аспекты машинного обучения на примерах из разных областей.

Что делать, если вы застряли

Мы старались как можно понятнее изложить все идеи, необходимые, чтобы воспроизвести описанные в книге примеры. Тем не менее, не исключено, что вы зайдете в тупик. Причины могут быть различными: опечатки, непредвиденные комбинации версий пакетов или недопонимание.

В таком случае обратиться за помощью можно в разные места. Скорее всего, кто-то уже сталкивался с вашей проблемой, и решение имеется на одном из следующих сайтов вопросов и ответов.

- `http://metaoptimize.com/qa`: этот сайт посвящен исключительно машинному обучению. Почти на каждый вопрос имеются квалифицированные ответы от специалистов. Даже если у вас нет никаких вопросов, полезно взять за правило время от времени заходить сюда и читать ответы.
- `http://stats.stackexchange.com`: этот сайт носит название Cross Validated, он похож на MetaOptimize, но посвящен больше вопросам из области статистики.
- `http://stackoverflow.com`: этот сайт во многом напоминает предыдущий, но диапазон рассматриваемых вопросов программирования гораздо шире. Так, на нем задается больше вопросов о некоторых используемых в этой книге пакетах, в частности, SciPy и matplotlib.
- Канал `#machinelearning` на сайте `https://freenode.net/`: это IRC-канал, посвященный проблемам машинного обучения. Здесь собирается небольшое, но очень активное и всегда готовое оказать помощь сообщество специалистов по машинному обучению.
- `http://www.TwoToReal.com`: это сайт мгновенных вопросов и ответов, созданный авторами книги для оказания помощи по вопросам, не попадающим ни в одну из описанных выше категорий. Если вы зададите здесь вопрос, то один из авторов немедленно получит сообщение, если находится в сети, и будет готов вступить с вами в беседу.

Как уже было сказано в начале книги, наша цель – помочь вам быстро освоить азы машинного обучения и дальше двигаться уже самостоятельно. Поэтому мы горячо рекомендуем составить свой список

блогов, относящихся к машинному обучению, и регулярно заглядывать в них. Это лучший способ узнать, что работает, а что – нет.

Единственный блог, о котором мы хотели бы упомянуть прямо сейчас (и подробнее рассказать в приложении) – `http://blog.kaggle.com`, блог компании Kaggle, которая проводит конкурсы по машинному обучению. Обычно победителям предлагается рассказать о своих подходах к конкурсным задачам, о том, какие стратегии не сработали, и как они пришли к победной стратегии. Этот блог обязателен к прочтению, пусть даже он окажется единственным.

Приступая к работе

В предположении, что Python уже установлен (годится любая версия, начиная с 2.7), нам нужно еще установить пакеты NumPy и SciPy для численных расчетов и matplotlib для визуализации.

Введение в NumPy, SciPy и matplotlib

Прежде чем говорить о конкретных алгоритмах машинного обучения, следует определиться с тем, как лучше хранить данные, которые мы будем обрабатывать. Это важно, потому что даже самый изощренный обучающий алгоритм ничем не поможет, если он не завершается. А так может случиться просто потому, что доступ к данным слишком медленный. Или потому, что выбранное представление данных заставляет операционную систему постоянно пробуксовывать. Добавьте сюда тот факт, что Python все-таки интерпретируемый язык (хотя и хорошо оптимизированный), слишком медленный для численных расчетов по сравнению с C или FORTRAN. Но тогда возникает вопрос – почему же так много ученых и компаний ставят свое благополучие на Python, даже в задачах, требующих очень интенсивных вычислений?

Дело в том, что, работая на Python, очень просто перепоручить численные расчеты низкоуровневым расширениям, реализованным на C или FORTRAN. Именно так устроены библиотеки NumPy и SciPy (`http://scipy.org/Download`). В этом тандеме NumPy отвечает за высоко оптимизированные многомерные массивы – основную структуру данных для большинства современных алгоритмов. А SciPy на базе этих массивов реализует быстрые численные алгоритмы. Наконец, matplotlib (`http://matplotlib.org/`) – пожалуй, самая удобная и функционально развитая библиотека для построения высококачественных графиков на Python.

Установка Python

По счастью, для всех популярных операционных систем, то есть Windows, Mac и Linux, существуют готовые инсталляторы NumPy, SciPy и matplotlib. Если вы не уверены, сможете ли справиться с их установкой, то установите дистрибутив Anaconda Python (его можно скачать с сайта `https://store.continuum.io/cshop/anaconda/`), созданный Трэвисом Олифантом (Travis Oliphant), основателем и активным автором проекта SciPy. От других дистрибутивов, например Enthought Canopy (`https://www.enthought.com/downloads/`) или Python(x,y) (`http://code.google.com/p/pythonxy/wiki/Downloads`), Anaconda отличается полной совместимостью с Python 3 – версией Python, которой мы будем пользоваться в этой книге.

NumPy как средство эффективной и SciPy как средство интеллектуальной обработки данных

Давайте разберем несколько простых примеров применения NumPy, а затем посмотрим, что SciPy дает сверх того. Попутно мы сделаем первую попытку построить графики с помощью чудесного пакета Matplotlib.

Желающие углубиться в тему могут ознакомиться с интересными примерами применения NumPy, приведенными в пособии по адресу `http://www.scipy.org/Tentative_NumPy_Tutorial`.

Весьма полезна также книга Ivan Idris «NumPy Beginner's Guide» (второе издание), вышедшая в издательстве Packt Publishing. Дополнительные учебные материалы можно найти на сайте `http://scipy-lectures.github.com`, а также в официальном руководстве по SciPy по адресу `http://docs.scipy.org/doc/scipy/reference/tutorial`.

В этой книге мы пользуемся версиями NumPy 1.8.1 и SciPy 0.14.0.

Изучаем NumPy

Итак, импортируем NumPy и немного поэкспериментируем. Для этого нужно запустить интерактивную оболочку Python:

```
>>> import numpy
>>> numpy.version.full_version
1.8.1
```

Поскольку мы не хотим загрязнять пространство имен, то, конечно же, не станем писать такой код:

```
>>> from numpy import *
```

Так как, к примеру, пакет `numpy.array` мог бы замаскировать пакет `array` из стандартной библиотеки Python. Вместо этого мы будем пользоваться следующим удобным сокращением:

```
>>> import numpy as np
>>> a = np.array([0,1,2,3,4,5])
>>> a
array([0, 1, 2, 3, 4, 5])
>>> a.ndim
1
>>> a.shape
(6,)
```

Только что мы создали массив – точно так же, как список в Python. Однако в массивах NumPy хранится дополнительная информация о форме. В данном случае мы имеем одномерный массив с шестью элементами. Пока никаких сюрпризов.

Теперь можно преобразовать этот массив в двумерную матрицу:

```
>>> b = a.reshape((3,2))
>>> b
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> b.ndim
2
>>> b.shape
(3, 2)
```

Интересные вещи начинаются с осознанием того, как сильно оптимизирован пакет NumPy. Например, в следующих операциях копирование по возможности не производится:

```
>>> b[1][0] = 77
>>> b
array([[ 0, 1],
       [77, 3],
       [ 4, 5]])
>>> a
array([ 0, 1, 77, 3, 4, 5])
```

В данном случае мы изменили один элемент `b` с `2` на `77` и, как видим, это изменение отразилось и на массиве `a`. Но если копирование таки необходимо, то это можно организовать:

```
>>> c = a.reshape((3,2)).copy()
>>> c
array([[ 0, 1],
       [77, 3],
       [ 4, 5]])
>>> c[0][0] = -99
>>> a
array([ 0, 1, 77, 3, 4, 5])
>>> c
array([[-99, 1],
       [ 77, 3],
       [ 4, 5]])
```

Теперь `c` и `a` – совершенно независимые копии.

Еще одно преимущество массивов NumPy – распространение операций на отдельные элементы. Например, умножение массива NumPy на число порождает массив такого же размера, в котором все элементы умножены на это число:

```
>>> d = np.array([1,2,3,4,5])
>>> d*2
array([ 2, 4, 6, 8, 10])
```

То же справедливо и для других операций:

```
>>> d**2
array([ 1, 4, 9, 16, 25])
```

Сравните с обычным списком Python:

```
>>> [1,2,3,4,5]*2
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
>>> [1,2,3,4,5]**2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Разумеется, прибегая к массивам NumPy, мы приносим в жертву гибкость списков Python. Такие простые операции, как добавление или удаление элементов, для массивов NumPy реализуются сложнее. Но, к счастью, в нашем распоряжении есть и то, и другое, так что можно выбирать подходящий инструмент по ситуации.

Индексирование

Своей эффективностью библиотека NumPy отчасти обязана разнообразием способов доступа к массивам. Помимо знакомого по спискам индексирования, можно использовать в качестве индексов сами массивы:

```
>>> a[np.array([2,3,4])]
array([77, 3, 4])
```

Учитывая, что логические условия также распространяются на отдельные элементы, мы получаем очень удобный способ доступа к данным:

```
>>> a>4
array([False, False, True, False, False, True], dtype=bool)
>>> a[a>4]
array([77, 5])
```

Следующая команда позволяет ограничить выбросы:

```
>>> a[a>4] = 4
>>> a
array([0, 1, 4, 3, 4, 4])
```

Поскольку эта бывает необходимо очень часто, существует специальная функция `clip`, которая позволяет задать обе границы с помощью одного вызова:

```
>>> a.clip(0,4)
array([0, 1, 4, 3, 4, 4])
```

Обработка отсутствующих значений

Средства индексирования NumPy оказываются особенно полезны, когда нужно произвести предварительную обработку данных, прочитанных из текстового файла. Чаще всего данные содержат некорректные значения, которые мы можем пометить как не-числа с помощью константы `numpy.NAN`:

```
>>> c = np.array([1, 2, np.NAN, 3, 4]) # допустим, что мы прочли
                                       # это из текстового файла
>>> c
array([ 1., 2., nan, 3., 4.])
>>> np.isnan(c)
array([False, False, True, False, False], dtype=bool)
>>> c[~np.isnan(c)]
array([ 1., 2., 3., 4.])
>>> np.mean(c[~np.isnan(c)])
2.5
```

Сравнение времени работы

Давайте сравним производительность NumPy и обычных списков Python. Следующая программа вычисляет сумму квадратов чисел от 1 до 1000 и замеряет время работы. Для большей точности мы прогоним ее 10 000 раз и напечатаем общее время.

```
import timeit
normal_py_sec = timeit.timeit('sum(x*x for x in range(1000))',
                              number=10000)
naive_np_sec = timeit.timeit(
                'sum(na*na)',
                setup="import numpy as np; na=np.arange(1000)",
                number=10000)
good_np_sec = timeit.timeit(
                'na.dot(na)',
                setup="import numpy as np; na=np.arange(1000)",
                number=10000)

print("Normal Python: %f sec" % normal_py_sec)
print("Naive NumPy: %f sec" % naive_np_sec)
print("Good NumPy: %f sec" % good_np_sec)

Normal Python: 1.050749 sec
Naive NumPy: 3.962259 sec
Good NumPy: 0.040481 sec
```

Сделаем два любопытных наблюдения. Во-первых, если использовать NumPy только для хранения данных (Naive NumPy), то программа работает в 3,5 раза дольше. Это странно, потому что мы ожидали, что написанное на C расширение будет работать гораздо быстрее. Одна из причин заключается в том, что доступ к отдельным элементам массива из Python обходится довольно дорого. Ускорение мы получим лишь в том случае, если сумеем применить алгоритм, не выходя за пределы оптимизированного кода расширения. Другое наблюдение поражает: использование входящей в NumPy функции `dot()`, которая делает в точности то же самое, приводит к 25-кратному ускорению. Вывод – реализуя любой алгоритм, нужно стремиться к тому, чтобы вместо цикла по отдельным элементам массива на Python воспользоваться какой-нибудь оптимизированной функцией, входящей в состав NumPy или SciPy.

Однако за быстродействие приходиться расплачиваться. Работая с массивами NumPy, мы утрачиваем невероятную гибкость списков Python, в которых можно хранить практически всё. Все элементы массива NumPy должны иметь одинаковый тип.

```
>>> a = np.array([1,2,3])
>>> a.dtype
dtype('int64')
```

Если мы попытаемся использовать элементы разных типов, как в следующем примере, то NumPy постарается привести их к одному и тому же, наиболее разумному в данных обстоятельствах, типу:

```
>>> np.array([1, "stringy"])
array(['1', 'stringy'], dtype='<U7')
>>> np.array([1, "stringy", set([1,2,3])])
array([1, stringy, {1, 2, 3}], dtype=object)
```

Изучаем SciPy

Поверх эффективных структур данных NumPy библиотека SciPy надстраивает многочисленные алгоритмы, работающие с массивами. Какой бы численный алгоритм из описываемых в современных учебниках ни взять, с большой вероятностью он будет так или иначе поддержан в SciPy. Это относится к операциям над матрицами, линейной алгебре, оптимизации, кластеризации, пространственным операциям и даже быстрому преобразованию Фурье. Поэтому прежде чем приступать к самостоятельной реализации численного алгоритма, обязательно посмотрите, нет ли его в модуле `scipy`.

Для удобства полное пространство имен NumPy доступно также через SciPy. Поэтому, начиная с этого места, мы будем обращаться ко всем средствам NumPy с помощью пространства имен SciPy. В корректности этого подхода легко убедиться, сравнив ссылки на любую функцию:

```
>>> import scipy, numpy
>>> scipy.version.full_version
0.14.0
>>> scipy.dot is numpy.dot
True
```

Алгоритмы разбиты на следующие группы.

Пакеты SciPy	Функциональность
`cluster`	• Иерархическая кластеризация (`cluster.hierarchy`) • Векторное квантование / метод *K* средних (`cluster.vq`)
`constants`	• Физические и математические константы • Методы преобразования
`fftpack`	Алгоритм дискретного преобразования Фурье
`integrate`	Процедуры интегрирования
`interpolate`	Интерполяция (линейная, кубическая и т. п.)
`io`	Ввод-вывод данных
`linalg`	Процедуры линейной алгебры на основе оптимизированных библиотек BLAS и LAPACK
`ndimage`	Обработка многомерных изображений

Пакеты SciPy	Функциональность
odr	Ортогональная регрессия
optimize	Оптимизация (нахождение минимумов и корней)
signal	Обработка сигналов
sparse	Разреженные матрицы
spatial	Пространственные структуры данных и алгоритмы
special	Специальные математические функции, например, функция Бесселя и якобиан
stats	Математическая статистика

Для нас наибольший интерес представляют пакеты `scipy.stats`, `scipy.interpolate`, `scipy.cluster` и `scipy.signal`. Сейчас мы вкратце рассмотрим возможности пакета stats, а изучение остальных отложим до тех глав, где они понадобятся.

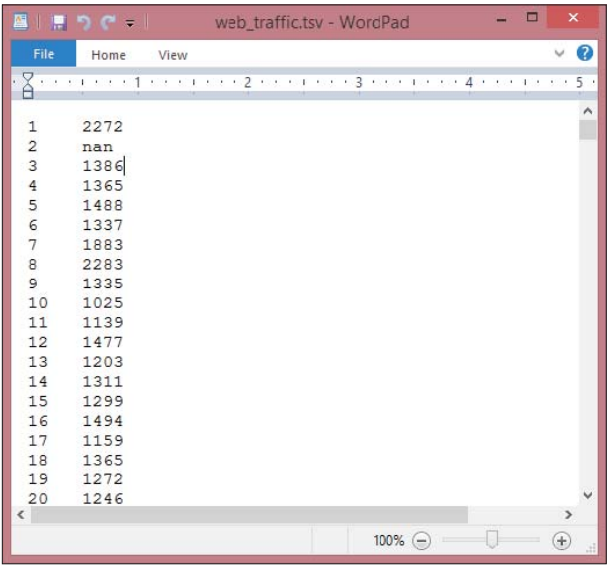
Наше первое (простенькое) приложение машинного обучения

Хватит слов, возьмем для примера гипотетическую недавно образованную компанию MLaaS, которая предоставляет платные услуги машинного обучения через Интернет. Наша компания растет, и ей понадобилось улучшить инфраструктуру для обслуживания поступающих запросов. Мы не хотим выделять слишком много ресурсов, потому что это будет дорого стоить. С другой стороны, если не зарезервировать достаточно ресурсов, то мы потеряем деньги из-за невозможности обслужить все запросы. Вопрос заключается в том, когда мы достигнем предельной пропускной способности инфраструктуры, которую оценили в 100 000 запросов в час. Хотелось бы заранее знать, когда запросить дополнительные серверы в облаке, чтобы обслужить все входящие запросы, но не платить за неиспользованные ресурсы.

Чтение данных

Мы собрали и агрегировали статистику веб за последний месяц, эти данные находятся в файле `ch01/data/web_traffic.tsv` (расширение `tsv` означает, что значения разделены знаками табуляции). Данные представляют собой количество запросов в час. В каждой строке указан час (по порядку) и количество запросов за этот час.

Ниже показаны первые несколько строк:



Метод `genfromtxt()` из библиотеки SciPy позволяет легко прочитать эти данные:

```
>>> import scipy as sp
>>> data = sp.genfromtxt("web_traffic.tsv", delimiter="\t")
```

Чтобы столбцы правильно распознавались, необходимо указать, что разделителем служит знак табуляции.

Простая проверка показывает, что данные прочитаны верно:

```
>>> print(data[:10])
[[ 1.00000000e+00 2.27200000e+03]
[ 2.00000000e+00 nan]
[ 3.00000000e+00 1.38600000e+03]
[ 4.00000000e+00 1.36500000e+03]
[ 5.00000000e+00 1.48800000e+03]
[ 6.00000000e+00 1.33700000e+03]
[ 7.00000000e+00 1.88300000e+03]
[ 8.00000000e+00 2.28300000e+03]
[ 9.00000000e+00 1.33500000e+03]
[ 1.00000000e+01 1.02500000e+03]]
>>> print(data.shape)
(743, 2)
```

Как видим, создан двумерный массив, содержащий 743 результата измерений.

Предварительная обработка и очистка данных

Для SciPy удобнее представить данные в виде двух векторов длиной 743 каждый. Первый вектор, `x`, будет содержать часы, второй, `y`, – количество запросов в течение соответствующего часа. Для такого разделения массива в SciPy предусмотрена специальная форма индексов, позволяющая выбирать отдельные столбцы:

```
x = data[:,0]
y = data[:,1]
```

В SciPy есть и много других способов выбрать данные из массива. Дополнительные сведения об индексировании, вырезании и итерировании см. в руководстве по адресу `http://www.scipy.org/Tentative_NumPy_Tutorial`.

Проблема в том, что в векторе `y` встречаются недопустимые значения – `nan`. И что с ними делать? Посмотрим, сколько раз встречается такое значение:

```
>>> sp.sum(sp.isnan(y))
8
```

Как видим, отсутствуют всего 8 из 743 значений, так что можно спокойно удалить их. Напомним, что массив SciPy можно индексировать другим массивом. Метод `sp.isnan(y)` возвращает массив булевых величин, показывающих, является элемент числом или нет. Оператор `~` вычисляет логическое отрицание этого массива, то есть позволяет выбрать из векторов `x` и `y` только элементы, содержащие число:

```
>>> x = x[~sp.isnan(y)]
>>> y = y[~sp.isnan(y)]
```

Чтобы составить первое представление о данных, нанесем на график диаграмму рассеяния, воспользовавшись библиотекой matplotlib. В этой библиотеке есть пакет pyplot, имитирующий интерфейс MATLAB – очень удобный и простой в использовании:

```
>>> import matplotlib.pyplot as plt
>>> # представляем точки (x,y) кружочками диаметра 10
>>> plt.scatter(x, y, s=10)
>>> plt.title("Web traffic over the last month")
>>> plt.xlabel("Time")
>>> plt.ylabel("Hits/hour")
>>> plt.xticks([w*7*24 for w in range(10)],
               ['week %i' % w for w in range(10)])
>>> plt.autoscale(tight=True)
```

```
>>> # рисуем полупрозрачную сетку пунктирными линиями
>>> plt.grid(True, linestyle='-', color='0.75')
>>> plt.show()
```

Дополнительные сведения о построении графиков можно найти на странице `http://matplotlib.org/users/pyplot_tutorial.html`.

Из графика видно, что в первые несколько недель трафик был более-менее стабильным, но в последнюю неделю наблюдался резкий рост.

Выбор подходящей модели и обучающего алгоритма

Составив представление о данных, вернемся к исходному вопросу: как долго наш сервер сможет обслуживать входящий трафик? Для ответа на него нужно сделать следующее:

1. Построить модель, отражающую зашумленные данные.
2. С помощью этой модели экстраполировать данные на будущее и определить, в какой момент следует расширить инфраструктуру.

Прежде чем строить модель...

Можно считать, что модель – это упрощенное теоретическое приближение к сложной реальности. Поэтому в модели всегда существует неточность, которую называют погрешностью аппроксимации. Именно величина погрешности и позволяет выбрать подходящую модель из множества возможных. Погрешность вычисляется как квадрат расстояния между реальными и предсказанными моделью данными; если `f` – обученная модельная функция, то погрешность равна:

```
def error(f, x, y):
  return sp.sum((f(x)-y)**2)
```

Векторы `x` и `y` содержат подготовленные ранее статистические данных о запросах. Тут мы наглядно видим удобство векторных функций в SciPy. Предполагается, что обученная модель принимает вектор и возвращает результаты в виде вектора того же размера, так что разность между результатом и вектором `y` корректно определена.

Начнем с прямой линии

Предположим ненадолго, что данные можно смоделировать прямой линией. Тогда наша задача – найти такую прямую, для которой погрешность аппроксимации минимальна. Именно это и делает функция `polyfit()` из SciPy. Получив на входе векторы `x`, `y` и требуемую степень полинома (для прямой линии степень равна 1), она находит модельную функцию, которая минимизирует определенную выше функцию погрешности:

```
fp1, residuals, rank, sv, rcond = sp.polyfit(x, y, 1, full=True)
```

Функция `polyfit()` возвращает параметры подобранной модельной функции `fp1`. Если задать `full=True`, то мы получим дополнительную информацию. Нас из нее интересуют только невязки, которые и описывают погрешность аппроксимации:

```
>>> print("Параметры модели: %s" % fp1)
Параметры модели: [ 2.59619213 989.02487106]
>>> print(residuals)
[  3.17389767e+08]
```

Это означает, что наилучшую линейную аппроксимацию дает следующая функция:

```
f(x) = 2.59619213 * x + 989.02487106.
```

Затем мы воспользуемся функцией `poly1d()` для построения модельной функции по параметрам модели: