Machine Learning with Python Cookbook

Practical Solutions from Preprocessing to Deep Learning

Chris Albon

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Крис Элбон

Машинное обучение с использованием Python. Сборник рецептов

Санкт-Петербург
«БХВ-Петербург»
2019

УДК 004.8+004.438Python
ББК 32.973.26-018.1
Э45

Элбон Крис

Э45 Машинное обучение с использованием Python. Сборник рецептов: Пер. с англ. — СПб.: БХВ-Петербург, 2019. — 384 с.: ил.

ISBN 978-5-9775-4056-8

Книга содержит около 200 рецептов решения практических задач машинного обучения, таких как загрузка и обработка текстовых или числовых данных, отбор модели, уменьшение размерности и многие другие. Рассмотрена работа с языком Python и его библиотеками, в том числе pandas и scikit-learn. Решения всех задач сопровождаются подробными объяснениями. Каждый рецепт содержит работающий программный код, который можно вставлять, объединять и адаптировать, создавая собственное приложение.

Приведены рецепты решений с использованием: векторов, матриц и массивов; обработки данных, текста, изображений, дат и времени; уменьшения размерности и методов выделения или отбора признаков; оценивания и отбора моделей; линейной и логистической регрессии, деревьев, лесов и *k* ближайших соседей; опорно-векторных машин (SVM), наивных байесовых классификаторов, кластеризации и нейронных сетей; сохранения и загрузки натренированных моделей.

Для разработчиков систем машинного обучения

УДК 004.8+004.438Python
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Сависте*
Перевод с английского	*Андрея Логунова*
Компьютерная верстка	*Ольги Сергиенко*
Оформление обложки	*Карины Соловьевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-1-491-98938-8 (англ.)
ISBN 978-5-9775-4056-8 (рус.)

Оглавление

Об авторе

Крис Альбон (Chris Albon) — аналитик данных и политолог с десятилетним опытом применения статистического обучения, искусственного интеллекта и разработки программного обеспечения для политических, социальных и гуманитарных проектов — от мониторинга выборов до оказания помощи в случае стихийных бедствий. В настоящее время Крис является ведущим аналитиком данных в BRCK — кенийском стартапе, создающем прочную сеть для пользователей Интернета на формирующемся рынке.

Предисловие

За последние несколько лет машинное (само)обучение стало частью широкого спектра повседневных, некоммерческих и правительственных операций. По мере роста популярности машинного обучения развивалась мелкосерийная индустрия высококачественной литературы, которая преподносила прикладное машинное обучение практикующим специалистам. Эта литература была очень успешной в подготовке целого поколения аналитиков данных и инженеров машинного обучения. Кроме того, в этой литературе рассматривалась тема машинного обучения с точки зрения предоставления учебного ресурса, демонстрировавшего специалисту, что такое машинное обучение и как оно работает. Вместе с тем, хотя этот подход и был плодотворным, он упустил из виду другую точку зрения на эту тему: как на материальную часть, гайки и болты, повседневного машинного обучения. В этом и состоит мотивация данной книги — предоставить читателям не фолиант по машинному обучению, а гаечный ключ для профессионала, чтобы книга лежала с зачитанными до дыр страницами на рабочих столах, помогла решать оперативные повседневные задачи практикующего специалиста по машинному обучению.

Если говорить точнее, то в книге используется задачно-ориентированный подход к машинному обучению, с почти 200 самодостаточными решениями (программный код можно скопировать и вставить, и он будет работать) наиболее распространенных задач, с которыми столкнется аналитик данных или инженер по машинному обучению, занимающийся созданием моделей.

Конечная цель книги — быть справочником для специалистов, строящих реальные машинно-обучающиеся системы. Например, представьте, что читатель имеет файл JSON, содержащий 1000 категориальных и числовых признаков с пропущенными данными и векторами категориальных целей с несбалансированными классами, и хочет получить интерпретируемую модель. Мотивация для этой книги состоит в предоставлении рецептов, чтобы помочь читателю освоить такие процессы, как:

- загрузка файла JSON (см. рецепт 2.5);
- стандартизация признаков (см. рецепт 4.2);
- кодирование словарей признаков (см. рецепт 5.3);
- импутация пропущенных значений классов (см. рецепт 5.4);
- уменьшение количества признаков с помощью главных компонент (см. рецепт 9.1);
- отбор наилучшей модели с помощью рандомизированного поиска (см. рецепт 12.2);

- тренировка классификатора на основе случайного леса (см. рецепт 14.4);
- отбор случайных признаков в случайных лесах (см. рецепт 14.7).

Читатель имеет возможность:

1. Копировать/вставлять программный код с полной уверенностью, что он действительно работает с включенным игрушечным набором данных.
2. Прочитать обсуждение, чтобы получить представление о теории, лежащей в основе метода, который этот программный код исполняет, и узнать, какие параметры важно учитывать.
3. Вставлять/комбинировать/адаптировать программный код из рецептов для конструирования фактического приложения.

Для кого предназначена книга

Данная книга не является введением в машинное (само)обучение. Если вы не чувствуете себя уверенно в области основных понятий машинного обучения либо никогда не проводили время за изучением машинного обучения, то не покупайте эту книгу. Она предназначена для практикующих специалистов машинного обучения, которые, чувствуя себя комфортно с теорией и понятиями машинного обучения, извлекут пользу из краткого справочника, содержащего программный код для решения задач, с которыми они сталкиваются, работая ежедневно с машинным обучением.

Предполагается также, что читатель уверен в своих знаниях языка программирования Python и управления его пакетами.

Для кого не предназначена книга

Как заявлено ранее, эта книга не является введением в машинное (само)обучение. Данная книга не должна быть вашим первым изданием по этой теме. Если вы не знакомы с такими понятиями, как перекрестная проверка, случайный лес и градиентный спуск, то вы, вероятно, не извлечете из этой книги такой же пользы, которую можно получить от одного из многих высококачественных текстов, специально предназначенных для ознакомления с этой темой. Я рекомендую прочитать одну из таких книг, а затем вернуться к этой книге, чтобы узнать рабочие, практические решения для задач машинного обучения.

Терминология, используемая в книге

Машинное (само)обучение опирается на методы из широкого спектра областей, включая информатику, статистику и математику. По этой причине при обсуждении машинного обучения существуют значительные расхождения в используемой терминологии.

Наблюдение — единое целое в нашем уровне исследования, например человек, сделка купли-продажи или запись.

Обучающийся алгоритм — алгоритм, используемый для того, чтобы обучиться наилучшим параметрам модели, например линейной регрессии, наивного байесовского классификатора или деревьев решений.

Модели — результат тренировки обучающегося алгоритма. Обучающиеся алгоритмы заучивают модели, которые мы затем используем для предсказания.

Параметры — веса или коэффициенты модели, заученные в ходе тренировки.

Гиперпараметры — настройки обучающегося алгоритма, которые необходимо отрегулировать перед тренировкой.

Результативность — метрический показатель, используемый для оценивания качества модели.

Потеря — метрический показатель, который максимизируется или минимизируется посредством тренировки.

Тренировка — применение обучающегося алгоритма к данным с использованием численных подходов, таких как градиентный спуск.

Подгонка — применение обучающегося алгоритма к данным с использованием аналитических подходов.

Данные — коллекция наблюдений.

Признательности

Эта книга была бы невозможна без любезной помощи множества друзей и незнакомых мне людей. Перечислить всех, кто протянул руку помощи в реализации этого проекта, будет невозможно, но я хотел бы хотя бы упомянуть Анжелу Басса, Терезу Борсух, Джастина Бозонье, Андре де Бруина, Нума Дхамани, Дэна Фридмана, Джоэла Груса, Сару Гвидо, Билла Камбороглу, Мэта Келси, Лиззи Кумар, Хилари Паркер, Нити Подьял, Себастьяна Рашку и Шрея Шанкар.

Я должен им всем по кружке пива или дать пять.

Комментарии переводчика

В центре внимания машинного обучения и его подобласти, глубокого обучения, находится обучающаяся система, то есть система, способная с течением времени приобретать новые знания и улучшать свою работу, используя поступающую информацию[1]. В зарубежной специализированной литературе для *передачи* знаний и *приобретения* знаний существуют отдельные термины — train (*натренировать*, обучить) и learn (*выучить*, обучиться).

[1] См. **https://ru.wikipedia.org/wiki/Обучающаяся_система**, а также **https://bigenc.ru/mathematics/text/1810335**. — *Прим. перев.*

Training (тренировка) — это работа, которую выполняет исследователь-проектировщик для получения обучившейся модели, в основе которой лежит обучающийся алгоритм, по сути искатель минимумов (или максимумов) для надлежащим образом сформулированных функций, а learning (самообучение, заучивание) — это работа, которую выполняет алгоритм-ученик по приобретению новых знаний или изменению и закреплению существующих знаний и поведения[2]. Когда же в русской специальной литературе используется термин "обучение", то он несет в себе двусмысленность, потому что под ним может подразумеваться и передача знаний, и получение знаний одновременно, как, например, в случае с термином "машинное обучение", который может означать и тренировку алгоритмических машин, и способность таких машин обучаться, что нередко вносит путаницу и терминологический разброд в переводной литературе при решении дилеммы "training-learning", в то время как появление в зарубежной технической литературе термина learning в любом виде подразумевает исключительно второе — *самообучение*, *заучивание* алгоритмом весов и других параметров. Отсюда вытекает одно важное следствие: английский термин machine learning обозначает *приобретение знаний алгоритмической машиной*, а следовательно, более соответствовать оригиналу будет термин *"машинное самообучение"* или *"автоматическое обучение"*. Весомым аргументом в пользу этого термина является и то, что с начала 60-х и до середины 80-х годов XX столетия у нас в ходу был похожий термин — "обучающиеся машины". Проблематика обучающихся и самопроизводящихся машин изучалась в работах А. Тьюринга "Может ли машина мыслить?" (1960, обучающиеся машины), К. Шеннона "Работы по теории информации и кибернетике" (самовоспроизводящиеся машины), Н. Винера "Кибернетика, или управление и связь в животном и машине" (1961), Н. Нильсона "Обучающиеся машины" (1974) и Я. З. Цыпкина "Основы теории обучающихся систем" (1970).

В настоящем переводе далее за основу принят зарубежный подход, который неизбежно привел к некоторой корректировке терминологии. Соответствующие области исследования переведены как *машинное самообучение* и *глубокое самообучение*. Применяемые в машинном обучении и глубоком обучении алгоритмы, модели и системы переведены как *обучающиеся*, *машинно-обучающиеся* и *глубоко обучающиеся*. То есть, акцент делается не на классификации алгоритма в соответствующей иерархии, а на его характерном свойстве. Далее методы, которые реализуются в обучающихся алгоритмах, переведены как *методы самообучения* (*ср.* методы обучения). Как известно, эти методы делятся на три широкие категории. Следуя принципу бритвы Оккама, они переведены как методы контролируемого самообучения (*ср.* обучение с учителем), методы неконтролируемого самообучения (*ср.* обучение без учителя) и методы самообучения с максимизацией вознаграждения, или подкрепления (*ср.* обучение с подкреплением). Последний термин обусловлен тем, что в его основе лежит алгоритм, который "учится максимизировать некое понятие вознаграждения", получаемого за правильно выполненное действие[3].

---

[2] См. **http://www.basicknowledge101.com/subjects/learningstyles.html#diy**. — *Прим. перев.*

[3] См. **https://en.wikipedia.org/wiki/Reinforcement_learning**. — *Прим. перев.*

Среди многих гиперпараметров, которые позволяют настроить работу обучающегося алгоритма, имеется rate of learing, который переведен как *скорость заучивания* (*ср.* темп обучения).

Исходный код

Перевод книги снабжен пояснениями и ссылками, размещенными в сносках. Вся кодовая база книги протестирована в среде Windows 10. При тестировании исходного кода за основу взят Python версии 3.6.4 (время перевода — май 2018 г.).

В книге используется ряд специализированных библиотек. В обычных условиях библиотеки Python можно скачать и установить из каталога библиотек Python `PyPi` (**https://pypi.python.org/**) при помощи менеджера пакетов `pip`. Однако следует учесть, что в ОС Windows для работы некоторых библиотек, в частности `scipy`, `scikit-learn`, 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. Библиотеку Numpy+MKL следует скачать с хранилища whl-файлов на веб-странице Кристофа Голька из Лаборатории динамики флуоресценции Калифорнийского университета в г. Ирвайн (**http://www.lfd.uci.edu/~gohlke/pythonlibs/**) и установить при помощи менеджера пакетов `pip` как whl (соответствующая процедура установки пакетов в формате WHL описана ниже). Например, для 64-разрядной операционной системы Windows и среды Python 3.6 команда будет такой:

```
pip install numpy-1.14.2+mkl-cp36-cp36m-win_amd64.whl
```

Стоит также отметить, что эти особенности установки не относятся к ОС Linux и Mac OS X.

Протокол установки библиотек

Далее предлагается список команд локальной установки библиотек, скачанных с хранилища whl-файлов.

```
python -m pip install --upgrade pip
pip install numpy-1.14.2+mkl-cp36-cp36m-win_amd64.whl
pip install scipy-1.1.0-cp36-cp36m-win_amd64.whl
pip install scikit_learn-0.19.1-cp36-cp36m-win_amd64.whl
pip install beautifulsoup4-4.6.0-py3-none-any.whl
pip install opencv_python-3.4.1-cp36-cp36m-win_amd64.whl
```

Следующие ниже библиотеки устанавливаются стандартным образом:

```
pip install matplotlib
pip install pandas
pip install sqlalchemy
pip install nltk
```

```
pip install fancyimpute
pip install seaborn
pip install pydotplus
pip install graphviz
pip install keras
pip install pydot
pip install joblib
```

> **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 май 2018 г. версии для 64-разрядной ОС Windows и Python 3.6.4.

Ниже перечислены адреса библиотек, которые следует скачать из хранилища и установить локально:

- numpy (**https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy**);
- scipy (**https://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy**);
- scikit-learn (**https://www.lfd.uci.edu/~gohlke/pythonlibs/#scikit-learn**);
- BeautifulSoup (**https://www.lfd.uci.edu/~gohlke/pythonlibs**);
- OpenCV (**https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv**).

Установка библиотек Python из whl-файлов

Библиотеки для Python можно разрабатывать не только на чистом Python. Довольно часто библиотеки программируются на C (динамические библиотеки), и для них пишется обертка Python. Либо 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-1.1.0-cp36-cp36m-win_amd64.whl
```

Откройте окно командной 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-1.1.0-cp36-cp36m-win_amd64.whl
```

При выборе библиотеки важно, чтобы разрядность устанавливаемой библиотеки и разрядность интерпретатора совпадали. Пользователи Windows могут брать whl-файлы с веб-сайта Кристофа Голька. Библиотеки там постоянно обновляются, и в архиве содержатся все, какие только могут понадобиться.

Блокноты Jupyter

В корневой папке размещены файлы с расширением ipynb. Это файлы блокнотов интерактивной среды программирования Jupyter (**http://jupyter.org/**). Блокноты Jupyter позволяют иметь в одном месте исходный код, результаты выполнения исходного кода, графики данных и документацию, которая поддерживает синтаксис упрощенной разметки Markdown и мощный синтаксис LaTeX.

Интерактивная среда программирования Jupyter — это зонтичный проект, который наряду с Python предназначен для выполнения в обычном веб-браузере небольших программ и фрагментов программного кода на других языках программирования, в том числе Julia, R и многих других (уже более 40 языков).

Интерактивная среда программирования Jupyter устанавливается стандартным образом при помощи менеджера пакетов `pip`.

```
pip install jupyter
```

Для того чтобы запустить интерактивную среду Jupyter, нужно в командной оболочке или окне терминала набрать и исполнить приведенную ниже команду:

```
jupyter notebook
```

Локальный сервер интерактивной среды Jupyter запустится в браузере, заданном по умолчанию (как правило, по адресу **http://localhost:8888/**).

Векторы, матрицы, массивы

Введение

Библиотека NumPy лежит в основе стека машинного самообучения на Python и позволяет эффективно работать со структурами данных, часто используемыми в машинном самообучении: векторами, матрицами и тензорами. Хотя NumPy не находится в центре внимания книги, эта библиотека будет часто появляться в последующих главах. В данной главе рассматриваются наиболее распространенные операции NumPy, с которыми мы, скорее всего, столкнемся во время работы с потоками операций машинного самообучения.

1.1. Создание вектора

Задача

Требуется создать вектор.

Решение

Использовать библиотеку NumPy для создания одномерного массива:

```
# Загрузить библиотеку
import numpy as np

# Создать вектор как строку
vector_row = np.array([1, 2, 3])

# Создать вектор как столбец
vector_column = np.array([[1],
                          [2],
                          [3]])
```

Обсуждение

Основной структурой данных NumPy является многомерный массив. Для того чтобы создать вектор, мы просто создаем одномерный массив. Как и векторы, эти массивы могут быть представлены горизонтально (т. е. как строки) или вертикально (т. е. как столбцы).

Дополнительные материалы для чтения

- "Векторы", математический ресурс Math's Fun ("Забавная математика") (**http://bit.ly/2FB5q1v**).
- "Евклидов вектор", Википедия (**http://bit.ly/2FtnRoL**).

1.2. Создание матрицы

Задача

Требуется создать матрицу.

Решение

Для создания двумерного массива использовать библиотеку NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2],
                   [1, 2],
                   [1, 2]])
```

Обсуждение

Для создания матрицы можно использовать двумерный массив NumPy. В нашем решении матрица содержит три строки и два столбца (столбец единиц и столбец двоек).

На самом деле NumPy имеет специальную матричную структуру данных:

```
matrix_object = np.mat([[1, 2],
                        [1, 2],
                        [1, 2]])

matrix([[1, 2],
        [1, 2],
        [1, 2]])
```

Однако матричная структура данных не рекомендуется по двум причинам. Во-первых, массивы являются де-факто стандартной структурой данных NumPy. Во-вторых, подавляющее большинство операций NumPy возвращают не матричные объекты, а массивы.

Дополнительные материалы для чтения

- "Матрица", Википедия (**http://bit.ly/2Ftnevp**).
- "Матрица", математический ресурс Wolfram MathWorld (**http://bit.ly/2Fut7IJ**).

1.3. Создание разреженной матрицы

Задача

Имеются данные с очень малым количеством ненулевых значений, которые требуется эффективно представить.

Решение

Создать разреженную матрицу:

```
# Загрузить библиотеки
import numpy as np
from scipy import sparse

# Создать матрицу
matrix = np.array([[0, 0],
                   [0, 1],
                   [3, 0]])

# Создать сжатую разреженную матрицу-строку (CSR-матрицу)
matrix_sparse = sparse.csr_matrix(matrix)
```

Обсуждение

В машинном самообучении часто возникает ситуация, когда имеется огромное количество данных; однако большинство элементов в данных являются нулями. Например, представьте матрицу, в которой столбцы — все фильмы в Netflix, строки — каждый пользователь Netflix, а значения — сколько раз пользователь смотрел конкретный фильм. Эта матрица будет иметь десятки тысяч столбцов и миллионы строк! Однако, поскольку большинство пользователей не смотрят почти все фильмы, подавляющая часть элементов матрицы будет равняться нулю.

Разреженные матрицы хранят только ненулевые элементы и исходят из того, что все другие значения будут равняться нулю, что приводит к значительной вычислительной экономии. В нашем решении мы создали массив NumPy с двумя ненулевыми значениями, а затем преобразовали его в разреженную матрицу. Если мы посмотрим на разреженную матрицу, то увидим, что в ней хранятся только ненулевые значения:

```
# Взглянуть на разреженную матрицу
print(matrix_sparse)
```

```
  (1, 1) 1
  (2, 0) 3
```

Существует несколько типов разреженных матриц. В *сжатых разреженных матрицах-строках* (compressed sparse row, CSR) элементы (1, 1) и (2, 0) представляют индексы ненулевых значений (с отсчетом от нуля), соответственно 1 и 3. Например,

элемент 1 находится во второй строке и втором столбце. Мы можем увидеть преимущество разреженных матриц, если создадим гораздо более крупную матрицу с еще большим количеством нулевых элементов, а затем сравним эту крупную матрицу с нашей исходной разреженной матрицей:

```
# Создать более крупную матрицу
matrix_large = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 1, 0, 0, 0, 0, 0, 0, 0],
                         [3, 0, 0, 0, 0, 0, 0, 0, 0]])

# Создать сжатую разреженную матрицу-строку (CSR-матрицу)
matrix_large_sparse = sparse.csr_matrix(matrix_large)

# Взглянуть на исходную разреженную матрицу
print(matrix_sparse)

  (1, 1) 1
  (2, 0) 3

# Взглянуть на более крупную разреженную матрицу
print(matrix_large_sparse)

  (1, 1) 1
  (2, 0) 3
```

Как мы видим, несмотря на то, что мы добавили в более крупную матрицу еще больше нулевых элементов, ее разреженное представление точно такое же, как и наша исходная разреженная матрица. То есть добавление нулевых элементов не изменило размер разреженной матрицы.

Как уже отмечалось, существует множество различных типов разреженных матриц, таких как сжатая разреженная матрица-столбец, список списков и словарь ключей. Хотя объяснение различных типов и их последствий выходит за рамки этой книги, стоит отметить, что "лучшего" типа разреженной матрицы не существует, однако между ними есть содержательные различия, и мы должны понимать, почему мы выбираем один тип и не выбираем другой.

Дополнительные материалы для чтения

- "Разреженные матрицы", документация SciPy (**http://bit.ly/2HReBZR**).
- "101 способ хранения разреженной матрицы", блог-пост (**http://bit.ly/2HS43cI**).

1.4. Выбор элементов

Задача

Требуется выбрать один или несколько элементов в векторе или матрице.

Решение

Массивы NumPy позволяют это легко сделать:

```
# Загрузить библиотеку
import numpy as np

# Создать вектор-строку
vector = np.array([1, 2, 3, 4, 5, 6])

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Выбрать третий элемент вектора
vector[2]
```

```
3
```

```
# Выбрать вторую строку, второй столбец
matrix[1,1]
```

```
5
```

Обсуждение

Как и большинство вещей в Python, массивы NumPy имеют нулевую индексацию, т. е. индекс первого элемента равен 0, а не 1. С учетом этого NumPy предлагает широкий спектр методов для выбора (т. е. индексирования и нарезки) элементов или групп элементов в массивах:

```
# Выбрать все элементы вектора
vector[:]
```

```
array([1, 2, 3, 4, 5, 6])
```

```
# Выбрать все вплоть до третьего элемента включительно
vector[:3]
```

```
array([1, 2, 3])
```

```
# Выбрать все после третьего элемента
vector[3:]
```

```
array([4, 5, 6])
```

```
# Выбрать последний элемент
vector[-1]
```

```
6
```

```
# Выбрать первые две строки и все столбцы матрицы
matrix[:2,:]

array([[1, 2, 3],
       [4, 5, 6]])

# Выбрать все строки и второй столбец
matrix[:,1:2]

array([[2],
       [5],
       [8]])
```

1.5. Описание матрицы

Задача

Требуется описать форму, размер и размерность матрицы.

Решение

Использовать атрибуты `shape`, `size` и `ndim`:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]])

# Взглянуть на количество строк и столбцов
matrix.shape

(3, 4)

# Взглянуть на количество элементов (строки * столбцы)
matrix.size

12

# Взглянуть на количество размерностей
matrix.ndim

2
```

Обсуждение

Эти операции могут показаться тривиальными (и это действительно так). Однако время от времени будет полезно проверить форму и размер массива для дальнейших вычислений и просто в качестве проверки состояния дел после некоторой операции.

1.6. Применение операций к элементам

Задача

Требуется применить некоторую функцию к нескольким элементам массива.

Решение

Использовать класс `vectorize` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Создать функцию, которая добавляет к чему-то 100
add_100 = lambda i: i + 100

# Создать векторизованную функцию
vectorized_add_100 = np.vectorize(add_100)

# Применить функцию ко всем элементам в матрице
vectorized_add_100(matrix)

array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

Обсуждение

Класс NumPy `vectorize` конвертирует обычную функцию в функцию, которая может применяться ко всем элементам массива или части массива. Стоит отметить, что `vectorize` по существу представляет собой цикл `for` над элементами и не увеличивает производительность. Кроме того, массивы NumPy позволяют выполнять операции между массивами, даже если их размерности не совпадают (этот процесс называется *трансляцией*). Например, мы можем создать гораздо более простую версию нашего решения, используя трансляцию:

```
# Добавить 100 ко всем элементам
matrix + 100
```

```
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

1.7. Нахождение максимального и минимального значений

Задача

Требуется найти максимальное или минимальное значение в массиве.

Решение

Использовать функции `max` и `min` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Вернуть максимальный элемент
np.max(matrix)
```

```
9
```

```
# Вернуть минимальный элемент
np.min(matrix)
```

```
1
```

Обсуждение

Часто требуется узнать максимальное и минимальное значения в массиве или подмножестве массива. Это может быть достигнуто с помощью методов `max` и `min`. Используя параметр `axis`, можно также применить операцию вдоль определенного направления:

```
# Найти максимальный элемент в каждом столбце
np.max(matrix, axis=0)
```

```
array([7, 8, 9])
```

```
# Найти максимальный элемент в каждой строке
np.max(matrix, axis=1)
```

```
array([3, 6, 9])
```

1.8. Вычисление среднего значения, дисперсии и стандартного отклонения

Задача

Требуется вычислить некоторые описательные статистические показатели о массиве.

Решение

Использовать функции `mean`, `var` и `std` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Вернуть среднее значение
np.mean(matrix)
```

```
5.0
```

```
# Вернуть дисперсию
np.var(matrix)
```

```
6.666666666666667
```

```
# Вернуть стандартное отклонение
np.std(matrix)
```

```
2.5819888974716112
```

Обсуждение

Так же как с функциями `max` и `min`, мы можем легко получать описательные статистические показатели о всей матрице или делать расчеты вдоль одной оси:

```
# Найти среднее значение в каждом столбце
np.mean(matrix, axis=0)
```

```
array([ 4., 5., 6.])
```

1.9. Реформирование массивов

Задача

Требуется изменить форму (количество строк и столбцов) массива без изменения значений элементов.

Решение

Использовать метод `reshape` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу 4x3
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9],
                   [10, 11, 12]])

# Реформировать матрицу в матрицу 2x6
matrix.reshape(2, 6)
```

```
array([[ 1, 2, 3,  4,  5,  6],
       [ 7, 8, 9, 10, 11, 12]])
```

Обсуждение

Метод `reshape` позволяет реструктурировать массив так, что мы сохраняем те же самые данные и при этом организуем их как другое количество строк и столбцов. Единственное требование состоит в том, чтобы формы исходной и новой матриц содержали одинаковое количество элементов (т. е. матрицы имели одинаковый размер). Размер матрицы можно увидеть с помощью атрибута `size`:

```
matrix.size
```

```
12
```

Одним из полезных аргументов в методе `reshape` является `-1`, который фактически означает "столько, сколько необходимо", поэтому `reshape(-1, 1)` означает одну строку и столько столбцов, сколько необходимо:

```
matrix.reshape(1, -1)
```

```
array([[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]])
```

Наконец, если мы предоставим одно целое число, то метод `reshape` вернет одномерный массив этой длины:

```
matrix.reshape(12)
```

```
array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

1.10. Транспонирование вектора в матрицу

Задача

Требуется транспонировать вектор в матрицу.

Решение

Использовать метод `T`:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Транспонировать матрицу
matrix.T
```

```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

Обсуждение

Транспонирование — это универсальная линейно-алгебраическая операция, в которой индексы столбцов и строк каждого элемента меняются местами. Вне предмета линейной алгебры, как правило, игнорируется один нюанс, который заключается в том, что технически вектор не может быть транспонирован, потому что он является лишь коллекцией значений:

```
# Транспонировать вектор
np.array([1, 2, 3, 4, 5, 6]).T
```

```
array([1, 2, 3, 4, 5, 6])
```

Вместе с тем общепринято называть транспонирование вектора преобразованием вектора-строки в вектор-столбец (обратите внимание на вторую пару скобок) или наоборот:

```
# Транспонировать вектор-строку
np.array([[1, 2, 3, 4, 5, 6]]).T
```

```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

1.11. Сглаживание матрицы

Задача

Требуется преобразовать матрицу в одномерный массив.

Решение

Использовать метод `flatten`:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Сгладить матрицу
matrix.flatten()
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Обсуждение

Метод `flatten` представляет собой простой метод преобразования матрицы в одномерный массив. В качестве альтернативы, чтобы создать вектор-строку, мы можем применить метод `reshape`:

```
matrix.reshape(1, -1)
```

```
array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

1.12. Нахождение ранга матрицы

Задача

Требуется узнать ранг матрицы.

Решение

Использовать линейно-алгебраический метод `matrix_rank` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 1, 1],
                   [1, 1, 10],
                   [1, 1, 15]])
```

```
# Вернуть ранг матрицы
np.linalg.matrix_rank(matrix)
2
```

Обсуждение

Ранг матрицы — это размерности векторного пространства, которые покрываются ее столбцами или строками. В библиотеке NumPy найти ранг матрицы легко благодаря методу `matrix_rank`.

Дополнительные материалы для чтения

- "Ранг матрицы", учебный ресурс CliffsNotes (**http://bit.ly/2HUzkMs**).

1.13. Вычисление определителя матрицы

Задача

Требуется узнать определитель матрицы.

Решение

Использовать линейно-алгебраический метод `det` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [2, 4, 6],
                   [3, 8, 9]])

# Вернуть определитель матрицы
np.linalg.det(matrix)

0.0
```

Обсуждение

Иногда может быть полезно вычислить определитель матрицы. Бибилиотека NumPy делает это легко с помощью метода `det`.

Дополнительные материалы для чтения

- "Определитель" | "Сущность линейной алгебры", глава 5, Youtube-канал 3Blue1Brown (**http://bit.ly/2FA6ToM**).
- "Определитель", математический ресурс Wolfram MathWorld (**http://bit.ly/2FxSUzC**).

1.14. Получение диагонали матрицы

Задача

Требуется получить элементы главной диагонали матрицы.

Решение

Использовать метод `diagonal`:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [2, 4, 6],
                   [3, 8, 9]])

# Вернуть диагональные элементы
matrix.diagonal()
```

Обсуждение

Библиотека NumPy позволяет легко получать элементы главной диагонали матрицы с помощью метода `diagonal`. Кроме того, с помощью параметра `offset` можно получить диагональ в стороне от главной диагонали:

```
# Вернуть диагональ на одну выше главной диагонали
matrix.diagonal(offset=1)

array([2, 6])

# Вернуть диагональ на одну ниже главной диагонали
matrix.diagonal(offset=-1)

array([2, 8])
```

1.15. Вычисление следа матрицы

Задача

Требуется вычислить след матрицы.

Решение

Использовать метод `trace`:

```
# Загрузить библиотеку
import numpy as np
```

```
# Создать матрицу
matrix = np.array([[1, 2, 3],
                   [2, 4, 6],
                   [3, 8, 9]])

# Вырнуть след
matrix.trace()
```

```
14
```

Обсуждение

След матрицы является суммой элементов главной диагонали и часто используется за кадром в методах машинного самообучения. Имея многомерный массив NumPy, мы можем вычислить след с помощью метода `trace`. В качестве альтернативы мы также можем вернуть диагональ матрицы и вычислить сумму ее элементов:

```
# Вернуть диагональ и сумму ее элементов
sum(matrix.diagonal())
```

```
14
```

Дополнительные материалы для чтения

- "След квадратной матрицы", математический ресурс MathOnline (**http://bit.ly/2FunM45**).

1.16. Нахождение собственных значений и собственных векторов

Задача

Требуется найти собственные значения и собственные векторы квадратной матрицы.

Решение

Использовать метод `linalg.eig` библиотеки NumPy:

```
# Загрузить библиотеку
import numpy as np

# Создать матрицу
matrix = np.array([[1, -1, 3],
                   [1,  1, 6],
                   [3,  8, 9]])
```