Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Реферат

на тему

**язык программирования python для задач машинного обучения**

|  |  |  |
| --- | --- | --- |
| Студент |  | С. А. Жишкевич |
| Преподаватель |  | М. В. Стержанов |

Минск 2019

СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ** 2](#_Toc24706233)

[**1. NumPy** 8](#_Toc24706234)

[**2. Matplotlib** 12](#_Toc24706235)

[**3. SciPy** 15](#_Toc24706236)

[**4. Pandas** 17](#_Toc24706237)

[**5. Scikit-learn** 21](#_Toc24706238)

[**ЗАКЛЮЧЕНИЕ** 25](#_Toc24706239)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 26](#_Toc24706240)

# **ВВЕДЕНИЕ**

На сегодняшний день язык программирования Python занимает лидирующие позиции в мировых рейтингах. Этому послужили низкий порог вхождения и относительная простота написания программ на данном языке. Он довольно прост для работы и чтения, а его использования снижает стоимость разработки и обслуживания ПО.

Но почему python имеет такой успех у machine-learning-специалистов?

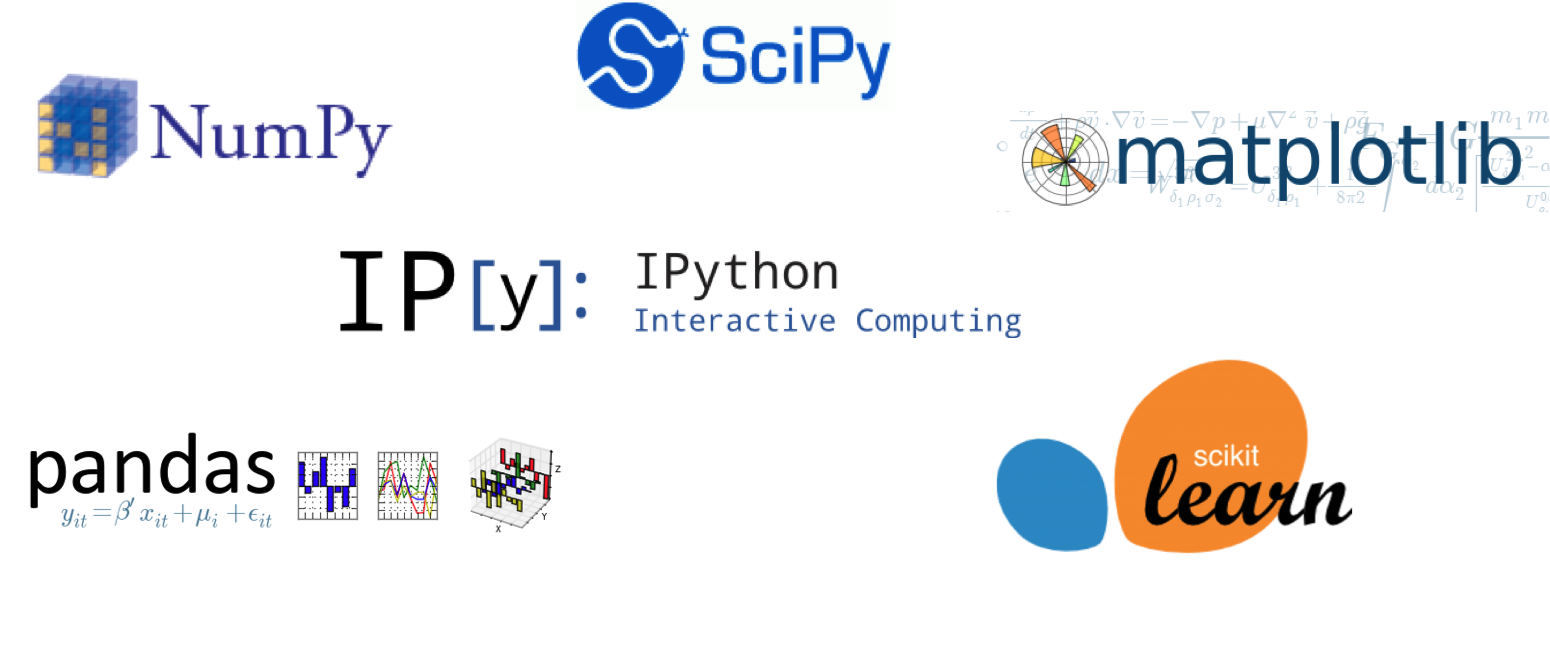


Рис.1 – python-библиотеки, используемые в machine learning

Ответов здесь может быть несколько: язык имеет простой интерфейс взаимодействия, что позволяет сконцентрироваться на непосредственной задаче. Благодаря простоте написания кода, программист тратить своё время на непосредственное решение поставленной задачи, избавляя себя от необходимости настройки окружения и так далее.

Возможность выбора нескольких парадигм программирования. Если необходимо быстро разработать прототип, то python как нельзя лучше подходит для этой задачи, потому как каждый может выбрать что-то удобное для себя: ООП, процедурное, функциональная парадигмы.

Динамическая типизация позволяет без лишних действий получать и преобразовывать данные.

Также язык обладает отличной производительностью при обработке данных.

При работе с языком разработчику не нужно уделять много внимания непосредственно написанию кода: все внимание он может сосредоточить на решении более сложных задач, связанных с машинным обучением.

Язык обладает большой поддержкой в виде библиотек и фреймворков, которые активным образом используются в машинном обучении. В научных расчетах используется Numpy, в продвинутых вычислениях — SciPy, в извлечении и анализе данных — SciKit-Learn. Эти библиотеки работают в таких фреймворках, как TensorFlow, CNTK и Apache Spark. Существует фреймворк для Python, разработанный специально для машинного обучения — это PyTorch.

Таким образом, можно сказать, что язык программирования Python идеально подходить для машинного обучения.

**1. Jupyter Notebook**

Машинное обучение – это область, где постоянно происходят изменения и новые разработки. Было бы очень удобно, чтобы можно было поделиться своими наработками и, более того, коллега мог бы сразу запустить и посмотреть на результат нашей деятельности.

**Jupyter Notebook**– невероятно мощный инструмент для интерактивной разработки и представления проектов в области наук о данных.

В Jupyter notebook вы можете разрабатывать, документировать и выполнять приложения на языке Python, он состоит из двух компонентов: веб-приложение, запускаемое в браузере, и ноутбуки – файлы, в которых можно работать с исходным кодом программы, запускать его, вводить и выводить данные и т.п.

Веб приложение позволяет:

* редактировать Python код в браузере, с подсветкой синтаксиса, автоотступами и автодополнением;
* запускать код в браузере;
* отображать результаты вычислений с медиа представлением (схемы, графики);
* работать с языком разметки Markdown и LaTeX.

Начнем с определения: что такое «**notebook**» (блокнот)? Блокнот объединяет код и его вывод в единый документ, который объединяет визуализацию, повествовательный текст, математические уравнения и другие мультимедиа. Этот интуитивно понятный рабочий процесс способствует итеративной и быстрой разработке, что делает ноутбуки все более популярным выбором для представления в данных и их анализа. Каждый файл **.ipynb** представляет собой текстовый файл, который описывает содержимое вашей записной книжки в формате [JSON](https://en.wikipedia.org/wiki/JSON). Каждая ячейка и ее содержимое, включая вложения изображений, которые были преобразованы в строки текста, перечислены в нем вместе с некоторыми [метаданными](https://ipython.org/ipython-doc/3/notebook/nbformat.html#metadata).

Ключевыми элементами любого ноутбука являются:

*kernel* (Ядро) – это «вычислительный движок», который выполняет код, содержащийся в документе ноутбука.

*cell* (Ячейка) – это контейнер для текста, который будет отображаться в записной книжке, или код, который будет выполняться ядром записной книжки.

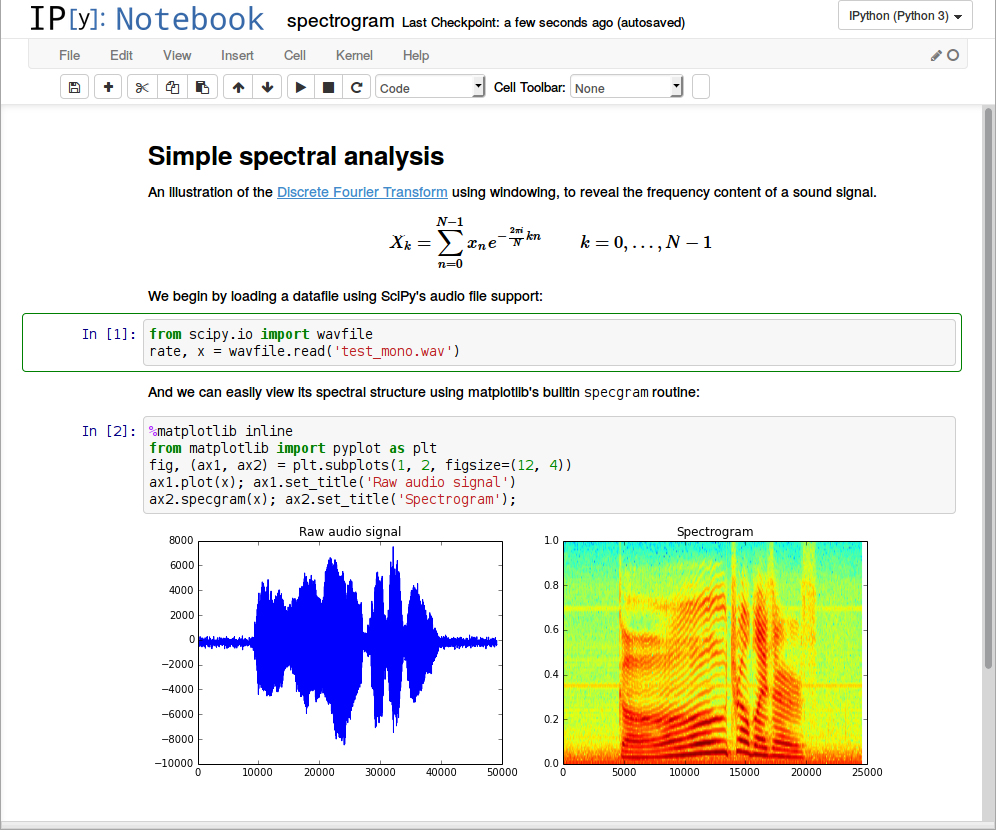


Рис. 2 – пример ноутбука

Есть два основных типа ячеек:

Ячейка кода содержит код, который должен быть выполнен в ядре, и отображает его вывод ниже.

Ячейка Markdown содержит текст, отформатированный с использованием Markdown, и отображает его вывод на месте при запуске.

Первая ячейка в новой записной книжке всегда является ячейкой **кода**.

Как правило, выходные данные ячейки поступают из любых текстовых данных, специально напечатанных во время выполнения ячеек, а также из значения последней строки в ячейке, будь то переменная-одиночка, вызов функции или что-то еще. Например:

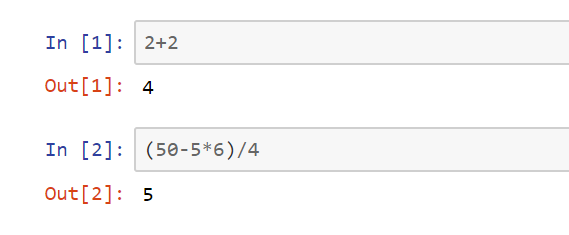


Рис.3 – пример ввода\вывода

За каждым ноутбуком работает ядро. Когда вы запускаете ячейку кода, этот код выполняется в ядре, и любой вывод возвращается обратно в ячейку для отображения. Состояние ядра сохраняется во времени и между ячейками – оно относится к документу в целом, а не к отдельным ячейкам. Например, если вы импортируете библиотеки или объявляете переменные в одной ячейке, они будут доступны в другой. Таким образом, вы можете думать о документе блокнота как о чем-то сравнимом с файлом сценария, за исключением того, что он является мультимедийным.

Существуют не только ядра для разных версий Python, но и [более 100 языков](https://github.com/jupyter/jupyter/wiki/Jupyter-kernels), включая Java, C и даже Fortran. Исследователи данных могут быть особенно заинтересованы в ядрах для [R](https://irkernel.github.io/) и [Julia](https://github.com/JuliaLang/IJulia.jl), а также в [imatlab](https://github.com/imatlab/imatlab" \t "_blank) и ядре [Calysto MATLAB Kernel](https://github.com/calysto/matlab_kernel" \t "_blank) для Matlab. Ядро [SoS](https://github.com/vatlab/SOS" \t "_blank) обеспечивает многоязычную поддержку в пределах одного ноутбука. Каждое ядро имеет свои собственные инструкции по установке, но, вероятно, потребует от вас выполнения некоторых команд на вашем компьютере.

В заключение можно сказать, что Juputyer Notebook – это мощный инструмент, который позволяет выстроить процесс разработки максимально гибко. Возможность с легкостью делиться своими наработками действительно очень удобна.

# **2. NumPy**

NumPy это open-source модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты. Они обеспечивают функционал, который можно сравнить с функционалом MatLab. NumPy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. SciPy (Scientific Python) расширяет функционал numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия, и другие прикладные математические техники.

Главной особенностью numpy является объект array. Массивы схожи со списками в python, исключая тот факт, что элементы массива должны иметь одинаковый тип данных, как float и int. С массивами можно проводить числовые операции с большим объемом информации в разы быстрее и, главное, намного эффективнее чем со списками.

Создание массива из списка:

a = np.array([1, 4, 5, 8], float)

>>> a

array([ 1., 4., 5., 8.])

>>> type(a)

<class 'numpy.ndarray'>

Здесь функция array принимает два аргумента: список для конвертации в массив и тип для каждого элемента. Ко всем элементам можно получить доступ и манипулировать ими также, как вы бы это делали с обычными списками:

>>> a[:2]

array([ 1., 4.])

>>> a[3]

8.0

>>> a[0] = 5.

>>> a

array([ 5., 4., 5., 8.])

Массивы могут быть и многомерными. В отличии от списков можно задавать команды в скобках. Вот пример двумерного массива (матрица):

>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)

>>> a

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

>>> a[0,0]

1.0

>>> a[0,1]

2.0

Транспонирование массивов также возможно, при этом создается новый массив:

>>> a = np.array(range(6), float).reshape((2, 3))

>>> a

array([[ 0., 1., 2.], [ 3., 4., 5.]])

>>> a.transpose()

array([[ 0., 3.], [ 1., 4.], [ 2., 5.]])

Два или больше массивов можно сконкатенировать при помощи метода concatenate:

>>> a = np.array([1,2], float)

>>> b = np.array([3,4,5,6], float)

>>> c = np.array([7,8,9], float)

>>> np.concatenate((a, b, c))

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

Также есть некоторое количество функций для создания специальных матриц. Для создания квадратной матрицы с главной диагональю, которая заполненная единицами, воспользуемся методом identity:

>>> np.identity(4, dtype=float)

array([[ 1., 0., 0., 0.],

[ 0., 1., 0., 0.],

[ 0., 0., 1., 0.],

[ 0., 0., 0., 1.]])

Когда для массивов мы используем стандартные математические операции, должен соблюдаться принцип: элемент--элемент. Это означает, что массивы должны быть одинакового размера во время сложения, вычитания и тому подобных операций:

>>> a = np.array([1,2,3], float)

>>> b = np.array([5,2,6], float)

>>> a + b

array([6., 4., 9.])

>>> a – b

array([-4., 0., -3.])

>>> a \* b

array([5., 4., 18.])

>>> b / a

array([5., 1., 2.])

>>> a % b

array([1., 0., 3.])

>>> b\*\*a

array([5., 4., 216.])

Для двухмерных массивов, умножение остается поэлементным и не соответствует умножению матриц:

>>> a = np.array([[1,2], [3,4]], float)

>>> b = np.array([[2,0], [1,3]], float)

>>> a \* b

array([[2., 0.], [3., 12.]])

Вдобавок к стандартным операторам, в numpy включена библиотека стандартных математических функций, которые могут быть применены поэлементно к массивам. Собственно функции: abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, и arctanh.

NumPy обеспечивает много функций для работы с векторами и матрицами. Функция dot возвращает скалярное произведение векторов:

>>> a = np.array([1, 2, 3], float)

>>> b = np.array([0, 1, 1], float)

>>> np.dot(a, b)

5.0

Функция dot также может умножать матрицы:

>>> a = np.array([[0, 1], [2, 3]], float)

>>> b = np.array([2, 3], float)

>>> c = np.array([[1, 1], [4, 0]], float)

>>> a

array([[ 0., 1.], [ 2., 3.]])

>>> np.dot(b, a)

array([ 6., 11.])

>>> np.dot(a, b)

array([ 3., 13.])

>>> np.dot(a, c)

array([[ 4., 0.], [ 14., 2.]])

>>> np.dot(c, a)

array([[ 2., 4.], [ 0., 4.]])

Также можно получить скалярное, тензорное и внешнее произведение матриц и векторов. Заметим, что для векторов внутреннее и скалярное произведение совпадает.

>>> a = np.array([1, 4, 0], float)

>>> b = np.array([2, 2, 1], float)

>>> np.outer(a, b)

array([[ 2., 2., 1.], [ 8., 8., 4.], [ 0., 0., 0.]])

>>> np.inner(a, b)

10.0

>>> np.cross(a, b)

array([ 4., -1., -6.])

NumPy также предоставляет набор встроенных функций и методов для работы с линейной алгеброй. Это всё можно найти в под-модуле linalg. Этими модулями также можно оперировать с вырожденными и невырожденными матрицами. Определитель матрицы ищется таким образом:

>>> a = np.array([[4, 2, 0], [9, 3, 7], [1, 2, 1]], float)

>>> a

array([[ 4., 2., 0.], [ 9., 3., 7.], [ 1., 2., 1.]])

>>> np.linalg.det(a)

-53.999999999999993

Также можно найти собственный вектор и собственное значение матрицы:

>>> vals, vecs = np.linalg.eig(a)

>>> vals

array([ 9. , 2.44948974, -2.44948974])

>>> vecs

array([[-0.3538921 , -0.56786837, 0.27843404], [-0.88473024, 0.44024287, -0.89787873], [-0.30333608, 0.69549388, 0.34101066]])

# **3. Matplotlib**

**Библиотека matplotlib** - это бибилиотека двумерной графики для языка программирования python с помощью которой можно создавать высококачественные рисунки различных форматов. Matplotlib представляет собой модуль-пакет для python.

Matplotlib cостоит из множества модулей. Модули наполнены различными классами и функциями, которые иерархически связаны между собой.

Создание рисунка в matplotlib схоже с рисованием в реальной жизни. Так художнику нужно взять основу (холст или бумагу), инструменты (кисти или карандаши), иметь представление о будущем рисунке (что именно он будет рисовать) и, наконец, выполнить всё это и нарисовать рисунок деталь за деталью.

В matplotlib все эти этапы также существуют, и в качестве художника-исполнителя здесь выступает сама библиотека. От пользователя требуется управлять действиями художника-matplotlib, определяя что именно он должен нарисовать и какими инструментами. Обычно создание основы и процесс непросредственно отображения рисунка отдаёт полностью на откуп matplotlib. Таким образом, пользователь библиотеки matplotlib выступает в роли управленца. И чем проще ему управлять конечным результатом работы matplotlib, тем лучше.

Так как matplotlib организована иерархически, а наиболее простыми для понимания человеком являются самые высокоуровневые функции, то знакомство с matplotlib начинают с самого высокоуровневого интерфейса **matplotlib.pyplot**. Так, чтобы нарисовать гистограмму с помощью этого модуля, нужно вызывать всего одну команду: matplotlib.pyplot.hist(arr).

Пользователю не нужно думать как именно библиотека нарисовала эту диаграмму. Если бы мы рисовали гистрограмму самостоятельно , то заметили бы, что она состоит из повоторяющихся по форме фигур - прямоугольников. А чтобы нарисовать прямоугольник, нужно знать хотя бы координату одного угла и ширину/длину. Рисовали же бы мы прямоугольник линиями, соединяя угловые точки прямоугольника.

Этот пример отображает иерархичность рисунков, когда итоговая диаграмма (высокий уровень) состоит из простых геометрических фигур (более низкий, средний уровень), созданных несколькими универсальными методами рисования (низкий уровень). Если бы каждый рисунок нужно было бы создавать вот так, с нуля, это было бы очень долго и утомительно.

Интерфейс matplotlib.pyplot является набором команд и функций, которые делают синтаксис графических matploltib команд похожим на команды, используемые в среде MATLAB(с). Изначально matplotlib планировался как свободная альтернатива MATLAB(с), где в одной среде имелись бы средства как для рисования, так и для численного анализа. Именно так в Matplotlib появился pylab, который объединяет модули pyplot и numpy в одно пространство имён.

Интерфейс pyplot позволяет пользователю сосредоточиться на выборе готовых решений и настройке базовых параметров рисунка. Это его главное достоинство, поэтому изучение matplotlib лучше всего начинать именно с интерфейса pyplot.

Рисунки в matplotlib создаются путём последовательного вызова команд: либо в интерактивном режиме (в консоли), либо в скрипте (текстовый файл с python-кодом). Графические элементы (точки, линии, фигуры и т.д.) наслаиваются одна на другую последовательно. При этом последующие перекрывают предыдущие, если они занимают общее участки на рисунке (регулируется параметром **zorder**).

В matplotlib работает правило "текущей области" ("current axes"), которое означает, что все графические элементы наносятся на текущую область рисования. Несмотря на то, что областей рисования может быть несколько, однf из них всегда является текущей.

Как было сказано выше самым главным объектом в matplotlib является рисунок Figure. Поэтому создание научной графики нужно начинать именно с создания рисунка. Создать рисунок в matplotlib означает задать форму, размеры и свойства основы-холста (canvas), на котором будет создаваться будущий график.

Создать рисунок figure позволяет метод plt.figure(). После вызова любой графической команды, то есть функции, которая создаёт какой-либо графический объект, например, plt.scatter() или plt.plot(), всегда существует хотя бы одна область для рисования (по умолчанию прямоугольной формы).

Чтобы результат рисования, то есть текущее состояние рисунка, отразилось на экране, можно воспользоваться командой plt.show(). Будут показаны все рисунки (figures), которые были созданы.

**import** **matplotlib.pyplot** **as** **plt**

fig = plt.figure() *# Создание объекта Figure*

**print** (fig.axes) *# Список текущих областей рисования пуст*

**print** (type(fig)) *# тип объекта Figure*

plt.scatter(1.0, 1.0) *# scatter - метод для нанесения маркера в точке (1.0, 1.0)*

**print** (fig.axes)

plt.show()

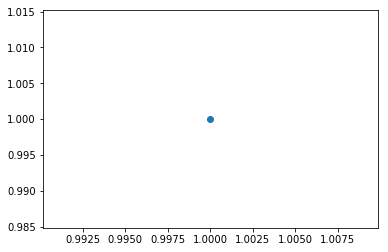


Рис.4 – пример «точка», используя matplotlib

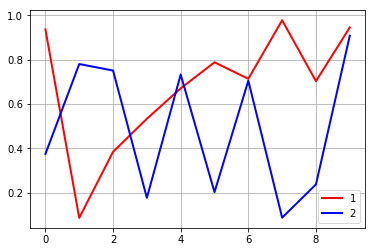


Рис.5 – пример графика

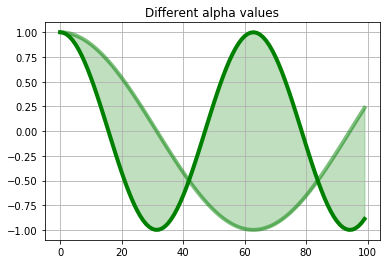


Рис.6 – пример графика

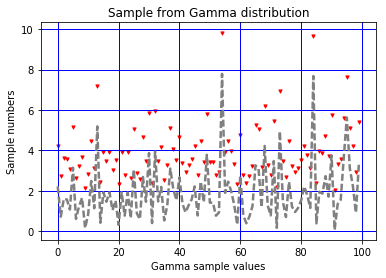


Рис.7 – пример графика

**4. SciPy**

*SciPy* ещё одна библиотека для языка Python предназначенная для выполнения научных и инженерных расчётов.

Библиотека позволяет выполнять следующее:

- поиск минимумов и максимумов функций

- вычисление интегралов функций

- обработка сигналов

- обработка изображений

- работа с генетическими алгоритмами

- решение обычных дифференциальных уравнений

- статистика

Использование SciPy, по сути, позволяет работать в среде аналогичной Matlab и др.

Библиотека содержит несколько основных модулей:  
 -scipy.constants – физические и математические константы

-scipy.linalg – задачи линейной алгебры

-scipy.optimize – поиск корней уравнений, оптимизация

-scipy.integrate – численное интегрирование и решение обычных дифференциальных уравнений

-scipy.interpolate – интерполяция и сглаживание

# **5. Pandas**

 Данный пакет делает Python мощным инструментом для анализа данных. Пакет дает возможность строить сводные таблицы, выполнять группировки, предоставляет удобный доступ к табличным данным, а при наличии пакета matplotlib дает возможность рисовать графики на полученных наборах данных. Далее будут показаны основы работы с пакетом, такие как загрузка данных, обращение к полям, фильтрация и построение сводных.

Для начала, скажем, пару слов о структурах хранения данных в Pandas. Основными являются Series и DataFrame.  
Series – это проиндексированный одномерный массив значений. Он похож на простой словарь типа dict, где имя элемента будет соответствовать индексу, а значение – значению записи.  
DataFrame — это проиндексированный многомерный массив значений, соответственно каждый столбец DataFrame, является структурой Series.  
Итак, со структурами чуток разобрались. Перейдем непосредственно к работе с пакетом. Для начала анализа каких-либо данных их надо загрузить. Pandas предоставляет широкий выбор источников данных, например:

* SQL
* Текстовые файлы
* Excel файлы
* HTML

Для примера загрузим 2 текстовых файла. Это можно сделать функцией read\_csv():

from pandas import read\_csv

df1 = read\_csv("df1.txt")

df2 = read\_csv("df2.txt",";") *#второй аргумент задает разделитель*

Вот так просто можно загрузить данные из файла и получить их табличное представление.

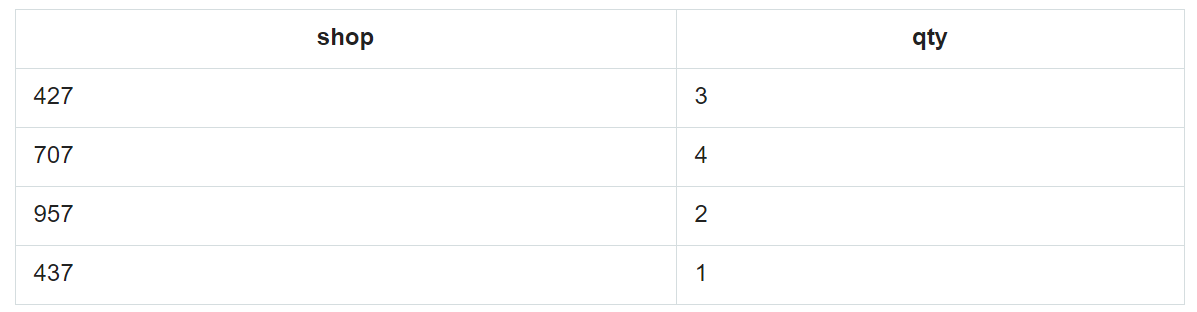


Рис. 8 – пример dataframe df1

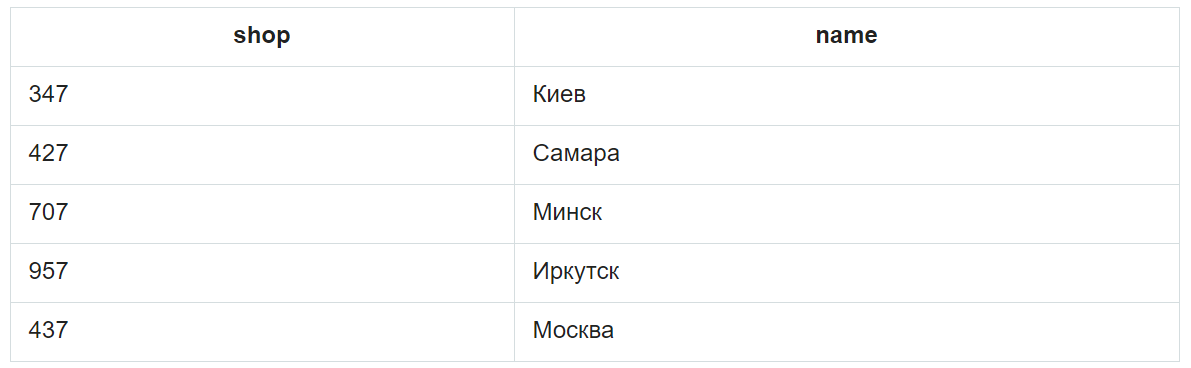


Рис. 9 – пример dataframe df2

Над наборами данных можно выполнять различные действия, например объединение, добавление столбцов, добавление записей, фильтрация, построение сводных и другие. Давайте теперь, чтобы продемонстрировать все описанные выше возможности, следующие задачи:

1. в набор с городами магазинов добавим поле `country` и заполним соответствующими странами
2. выберем украинский магазин и поменяем его номер
3. добавим магазин, полученный на предыдущем шаге, к общему списку
4. добавим количество из df1 к набору df2
5. построим сводную таблицу по странам и количеству отгрузок

Итак, для добавления нового столбца в набор данных существует команда [insert()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.insert.html?highlight=insert#pandas.DataFrame.insert):

country = [u'Украина',u'РФ',u'Беларусь',u'РФ',u'РФ'] df2.insert(1,'country',country)

В нашем случае функции передается 3 аргумент:

1. номер позиции, куда будет вставлен новый столбец
2. имя нового столбца
3. массив значений столбца (в нашем случае, это обычный список list)

Вид df2 после выполнения выше описанных операций:



Рис. 10 – пример dataframe df2

Теперь к нашему основному списку магазинов df2, можно подтянуть количество из набора данных df1. Сделать это можно с помощью функции [merge()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html?highlight=merge#pandas.DataFrame.merge), которая соединяет два набора данных (аналог join в SQL):

res = df2.merge(df1, 'left', on='shop')

 Набор данных перед финальной операцией выглядит так:



Рис. 11 – пример использования функции merge

Таким образом, мы убедились, что pandas является крайне мощным инструментов для быстрой и эффективной обработки большого количества данных. Библиотека имеет намного больший функционал, чем было показало. Например, библиотека позволяет сразу строить графики по полученным данным и многое другое.

# **5. Scikit-learn**

Библиотека scikit-learn предоставляет реализацию целого ряда алгоритмов для обучения с учителем (Supervised Learning) и обучения без учителя (Unsupervised Learning) через интерфейс для языка программирования Python. Данная библиотека распространяется под лицензией "Simplified BSD License" и имеет дистрибутивы для множества различных версий Linux, поощрая тем самым академическое и коммерческое использование scikit-learn.

Scikit-learn построена поверх [SciPy](http://www.scipy.org/) (Scientific Python), который должен быть установлен перед использованием scikit-learn.

Одна из основных концепций библиотеки scikit-learn - библиотека с уровнем надежности и поддержки, который необходим для продакшн-систем, а это значит, что большое внимание уделяется вопросам удобства использования, качества кода, документации и оптимизации скорости работы библиотеки.

Несмотря на то что весь интерфейс библиотеки представлен на Python, но использование библиотек, написанных на C во внутренней реализации некоторых частей scikit-learn, позволяет значительно повысить скорость работы, например, использование NumPy для работы с массивами и для операций с матрицами.

Библиотека scikit-learn ориентирована в первую очередь на моделирование данных, а не на загрузку, манипуляцию и обобщение данных. Для таких целей, как раз-таки, лучше использовать NumPy и Pandas.

Вот несколько популярных функциональных областей, в которых scikit-learn помогает решать поставленные задачи:

* Кластеризация (Clustering): для группировки неразмеченных данных, например, метод k-средних (k-means)
* Перекрестная проверка (Cross Validation): для оценки эффективности работы модели на независимых данных
* Наборы данных (Datasets): для тестовых наборов данных и для генерации наборов данных с определенными свойствами для исследования поведенческих свойств модели
* Сокращение размерности (Dimensionality Reduction): для уменьшения количества атрибутов для визуализации и отбора признаков (Feature Selection), например, метод главных компонент (Principal Component Analysis)
* Алгоритмические композиции (Ensemble Methods): для комбинирования предсказаний нескольких моделей
* Извлечение признаков (Feature Extraction): определение атрибутов в изображениях и текстовых данных
* Отбор признаков (Feature Selection): для выявления значимых атрибутов на основе которых будет построена модель
* Оптимизация параметров алгоритма (Parameter Tuning): для получения максимально эффективной отдачи от модели
* Множественное обучение (Manifold Learning): для нелинейного сокращения размерности данных
* Алгоритмы обучения с учителем (Supervised Models): огромный набор методов не ограничивается обобщенными линейными моделями (Generalized Linear Models), дискриминантным анализом (Discriminate Analysis), наивным байесовским классификатором (Naive Bayes), нейронными сетями (Neural Networks), методом опорных векторов (Support Vector Machines) и деревьями принятия решений (Decision Trees).

Следующий небольшой пример показывает, как легко пользоваться библиотекой scikit-learn. В данном примере мы используем достаточно популярный алгоритм на основе деревьев принятия решений под названием Classification and Regression Tress (CART) и применяем его к набору данных Iris. Данный набор данных используется для различных примеров и поставляется вместе с библиотекой. Классификатор проходит процесс настройки и выполняется процесс предсказания. Все это происходит на тренировочном наборе данных. И в самом конце мы выводим на экран оценку качества работы нашей модели и матрицу неточностей (Confusion Matrix).

|  |
| --- |
| # Decision Tree Classifier |
|  | from sklearn import datasets |
|  | from sklearn import metrics |
|  | from sklearn.tree import DecisionTreeClassifier |
|  |  |
|  | # загрузка данных iris |
|  | dataset = datasets.load\_iris() |
|  | # процесс настройки модели CART по выборке данных |
|  | model = DecisionTreeClassifier() |
|  | model.fit(dataset.data, dataset.target) |
|  | print(model) |
|  | # предсказание |
|  | expected = dataset.target |
|  | predicted = model.predict(dataset.data) |
|  | # оценка качества работы предсказательной модели |
|  | print(metrics.classification\_report(expected, predicted)) |
|  | print(metrics.confusion\_matrix(expected, predicted)) |
|  |  |

Всем хорошо знакомо, что большинство градиентных методов (на которых по-сути и основаны почти все алгоритмы машинного обучения) сильно чуствительны к шкалированию данных. Поэтому перед запуском алгоритмов чаще всего делается либо **нормализация**, либо так называемая **стандартизация**. Нормализация предполагает замену номинальных признаков так, чтобы каждый из них лежал в диапазоне от 0 до 1. Стандартизация же подразумевает такую предобработку данных, после которой каждый признак имеет среднее 0 и дисперсию 1. В Scikit-Learn уже есть готовые для этого функции:

from sklearn import preprocessing

*# normalize the data attributes*

normalized\_X = preprocessing.normalize(X)

*# standardize the data attributes*

standardized\_X = preprocessing.scale(X)

Не секрет, что зачастую самым важным при решении задачи является умение правильно отобрать и даже создать признаки. В англоязычной литературе это называется **Feature Selection** и **Feature Engineering**. В то время как Future Engineering довольно творческий процесс и полагается больше на интуицию и экспертные знания, для Feature Selection есть уже большое количество готовых алгоритмов. «Древесные» алгоритмы допускают расчета информативности признаков:

from sklearn import metrics

from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()

model.fit(X, y)

*# display the relative importance of each attribute* print(model.feature\_importances\_)

Пример реализации логистической регрессии:

from sklearn import metrics

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X, y) print(model)

*# make predictions*

expected = y

predicted = model.predict(X)

*# summarize the fit of the model* print(metrics.classification\_report(expected, predicted)) print(metrics.confusion\_matrix(expected, predicted))

Таким образом, мы видим, что библиотека предоставляет обширный базовый набор для решения большого количества задач машинного обучения.

# **ЗАКЛЮЧЕНИЕ**

Таким образом, можно сказать, что язык программирования Python имеет очень обширную библиотечную поддержку для решения задач машинного обучения и анализа данных. Весь выше описанный стек позволяет упростить такие задачи как: начальная загрузка данных, визуализация данных, решение типовых задач линейной алгебры, статистики, готовые реализации многих методов машинного обучения, что позволяет сэкономить значительное количество времени и сосредоточиться на непосредственной задаче.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Почему Python используется для машинного обучения [Электронный ресурс] – Режим доступа: <https://medium.com/nuances-of-programming/%D0%BF%D0%BE%D1%87%D0%B5%D0%BC%D1%83-python-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D1%83%D0%B5%D1%82%D1%81%D1%8F-%D0%B4%D0%BB%D1%8F-%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE-%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D1%8F-bc1a2d8f5963> – Дата доступа: 14.11.2019

[2] Jupyter notebook для начинающих: учебник[Электронный ресурс] – Режим доступа: <https://webdevblog.ru/jupyter-notebook-dlya-nachinajushhih-uchebnik/> – Дата доступа: 13.11.2019

[3] Нескучный туториал по NumPy[Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/469355/> – Дата доступа

11.11.2019

[4] Построение графиков в Matplotlib при помощи Python [Электронный ресурс] – Режим доступа: <https://python-scripts.com/matplotlib> – Дата доступа 09.11.2019

[5] Основы SciPy [Электронный ресурс] – Режим доступа: <https://www.slideshare.net/tm_ssau/scipy-75716240> – Дата доступа 13.11.2019

[6] Введение в анализ данных с помощью Pandas [Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/196980/> - Дата доступа 12.11.2019

[7] Введение в машинное обучение с помощью Python и Scikit-Learn [Электронный ресурс] – Режим доступа: <https://habr.com/ru/company/mlclass/blog/247751/> - Дата доступа 12.11.2019