

ЧАСТЬ 1. Практические задания.

Занятие 1. БАЗОВЫЕ ОПЕРАЦИИ И СИНТАКСИС ЯЗЫКА PYTHON

Установка Python 3 на компьютер. Скачать установочный файл можно с официального сайта <https://www.python.org/>, на котором необходимо выбрать "latest python release" и версию Python 3. Внизу страницы с описанием выбранной версии найти кнопку "download page". Из появившегося списка файлов (если ваша ОС Windows) необходимо загрузить Windows x86 executable installer (если система 32-х битная) или Windows x86-64 executable installer (если система 64-х битная). Эти данные из системы можно узнать через настройки системы Windows. Обратите внимание, что загрузка версии Python по умолчанию с главной страницы всегда будет скачивать 32-битную версию, что не является оптимальным решением для 64-битных ОС.

Запустить установщик; выбрать, каким пользователем предоставить доступ к программе; выбрать папку для установки (или оставить по умолчанию); выбрать компоненты, которые будут установлены (оставьте компоненты по умолчанию, если не уверены; важно установить галочку напротив «PATH»). Проверьте, что путь к Python был добавлен в системную переменную PATH.

Далее кнопка «Finish». Python установлен. Также в установщик Python для Windows встроена среда разработки IDLE. Прямо сейчас вы можете написать свою первую программу!

Files					
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		bcd9f22cf531efc6f06ca6b9b2919bd4	23277790	SIG
XZ compressed source tarball	Source release		389d3ed26bd4d97c741d9e5423da1f43b	17389636	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	4b544fc0ac8c3cfd6b7dede23ddb79e	29305353	SIG
Windows help file	Windows		1094c8d9438ad1adc263ca57ceb3b927	8186795	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	60f77740b30030b22699dbd14883a4a3	7502379	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	7083fed513c3c9a4ea655211df9ade27	26940592	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	da0b17ae84d6579f8df3eb24927fd825	1348904	SIG
Windows x86 embeddable zip file	Windows		97c6558d479dc53bf448580b66ad7c1e	6659999	SIG
Windows x86 executable installer	Windows		1e6d31c98c68c723541f0821b3c15d52	25875560	SIG
Windows x86 web-based installer	Windows		22f68f09e533c4940fc006e035f08aa2	1319904	SIG

Рис 1.1.

Создание и открытие скрипта python с помощью IDLE. Каждая установка Python поставляется с простой средой разработки и обучения IDLE. Это класс приложений, которые помогают написать код более эффективно. После установки Вы можете использовать Python IDLE в качестве интерактивного интерпретатора или редактора файлов.

Программы Python – это файлы с расширением «.py», которые содержат строки кода. Python IDLE позволяет создавать и редактировать эти файлы, а также предоставляет несколько полезных функций (как в профессиональных IDE), таких как подсветка основного синтаксиса, автозавершение кода и автоотступ. Профессиональные IDE, как правило, имеют больший функционал.

Работа в консоли является режимом работы по умолчанию для Python IDLE. Когда Вы нажимаете на значок, чтобы открыть программу, консоль – это первое, что Вы видите.

Чтобы запустить новый файл Python, выберите File → New File в строке меню. Это откроет пустой файл в редакторе. В этом окне Вы можете написать новый файл Python. Вы также можете открыть существующий файл Python, выбрав File → Open... в строке меню. Это откроет браузер файлов и позволит найти файл Python, который Вы хотите открыть.

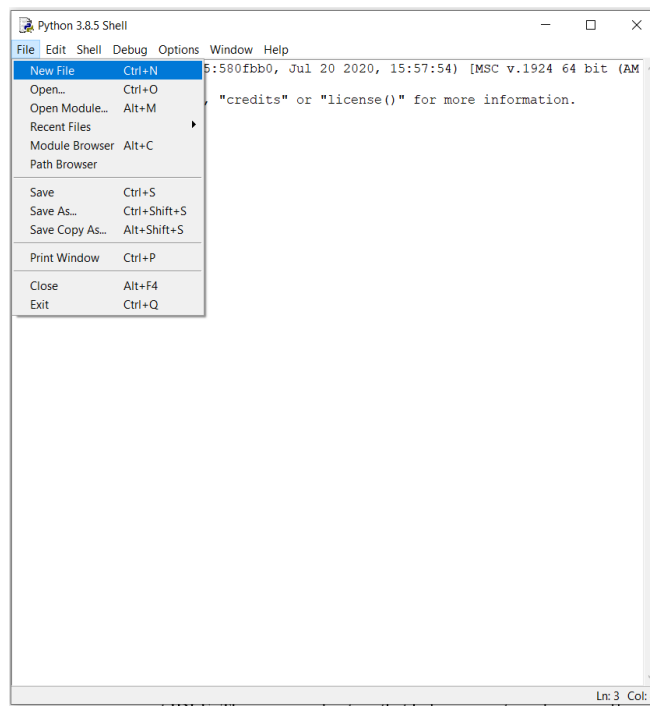


Рис 1.2.

После открытия файла в Python IDLE, Вы можете внести в него изменения. Содержимое файла отображается в открытом окне. Панель в верхней части окна содержит три важных элемента:

1. Имя файла, который Вы редактируете;
2. Полный путь к папке, где Вы можете найти этот файл на вашем компьютере;
3. Версия Python, которую использует IDLE.

Например, файл «myFile.py», который находится в папке «Documents». Версия Python 3.7.1.

В правом нижнем углу окна также есть две цифры:

1. Ln: показывает номер строки, на которой находится ваш курсор.
2. Col: показывает номер столбца, на котором находится ваш курсор.

Python IDLE использует обозначение астериксом («звездочка»), чтобы сообщить, что в файле есть несохраненные изменения. Файл можно сохранить с помощью стандартного сочетания клавиш или выбрать Файл → Сохранить в строке меню. Убедитесь, что Вы сохранили свой файл с расширением «.py», чтобы подсветка синтаксиса была включена.

Открытие в Python csv файла. Скачать файл по адресу:
https://archive.ics.uci.edu/ml/machine-learning-databases/00529/diabetes_data_upload.csv.

Это клинические данные испытуемых, у которых имелось или не имелось заболевание сахарным диабетом. Первая строчка содержит названия колонок.

```
1 Age, Gender, Polyuria, Polydipsia, sudden weight loss, weakness, Polyphagia, Ger
2 40, Male, No, Yes, No, Yes, No, No, No, Yes, No, Yes, No, Yes, Yes, Yes, Positive
3 58, Male, No, No, No, Yes, No, No, Yes, No, No, No, Yes, No, Yes, No, Positive
4 41, Male, Yes, No, No, Yes, Yes, No, No, Yes, No, Yes, No, Yes, Yes, No, Positive
5 45, Male, No, No, Yes, Yes, Yes, Yes, No, Yes, No, Yes, No, No, No, No, Positive
6 60, Male, Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Positive
7 55, Male, Yes, Yes, No, Yes, Yes, No, Yes, Yes, No, Yes, No, Yes, Yes, Yes, Positive
8 57, Male, Yes, Yes, No, Yes, Yes, Yes, No, No, No, Yes, Yes, No, No, No, Positive
9 66, Male, Yes, Yes, Yes, Yes, No, No, Yes, Yes, Yes, No, Yes, Yes, No, No, Positive
10 67, Male, Yes, Yes, No, Yes, Yes, Yes, No, Yes, Yes, No, Yes, Yes, No, Yes, Positive
11 70, Male, No, Yes, Yes, Yes, Yes, No, Yes, Yes, Yes, No, No, No, Yes, No, Positive
12 44, Male, Yes, Yes, No, Yes, No, Yes, No, No, Yes, Yes, No, Yes, Yes, No, Positive
13 38, Male, Yes, Yes, No, No, Yes, Yes, No, Yes, No, Yes, No, Yes, No, No, Positive
14 35, Male, Yes, No, No, No, Yes, Yes, No, No, Yes, Yes, No, No, Yes, No, Positive
15 61, Male, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, No, No, No, Yes, Yes, Positive
16 60, Male, Yes, Yes, No, Yes, Yes, No, Yes, Yes, No, Yes, Yes, No, No, No, Positive
17 58, Male, Yes, Yes, No, Yes, Yes, No, No, No, No, Yes, Yes, Yes, No, No, Positive
18 54, Male, Yes, Yes, Yes, Yes, No, Yes, No, No, No, Yes, No, Yes, No, No, Positive
19 67, Male, No, Yes, No, Yes, Yes, No, Yes, No, Yes, Yes, Yes, Yes, Yes, Yes, Positive
20 66, Male, Yes, Yes, No, Yes, Yes, No, Yes, No, No, No, Yes, Yes, No, No, Positive
21 43, Male, Yes, Yes, Yes, Yes, No, Yes, No, No, No, No, No, No, No, No, Positive
22 62, Male, Yes, Yes, No, Yes, Yes, No, Yes, No, Yes, No, Yes, Yes, No, No, Positive
23 54, Male, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, Yes, No, Yes, Yes, No, Positive
24 39, Male, Yes, No, Yes, No, No, Yes, No, Yes, Yes, No, No, No, Yes, No, Positive
25 48, Male, No, Yes, Yes, Yes, No, No, Yes, Yes, Yes, Yes, No, No, No, No, Positive
26 58, Male, Yes, Yes, Yes, Yes, Yes, No, Yes, No, No, Yes, Yes, Yes, No, Yes, Positive
27 32, Male, No, No, No, No, No, Yes, No, No, Yes, Yes, No, No, No, Yes, Positive
28 42, Male, No, No, No, Yes, Yes, No, No, No, Yes, No, No, Yes, No, No, Positive
29 52, Male, Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, No, Yes, Yes, Yes, No, No, Positive
30 38, Male, No, Yes, No, No, No, Yes, No, No, No, No, No, No, Yes, No, Positive
```

Рис 1.3.

Поместите файл в папке с проектом и адаптируйте пример ниже, чтобы прочитать файл в Python.

```
import csv
with open('testfile.csv', newline='') as csvfile:
    data = list(csv.reader(csvfile))
print(data)
```

Это загрузит данные в виде вложенного списка (list of lists).

Расчет среднего значения по колонке Age с помощью цикла for. В одной из колонок двумерного массива data (list of lists) хранится информация о возрасте испытуемых исследования. Модифицируйте код ниже, чтобы рассчитать среднее нужной колонки. Имейте в виду, что вы имеете дело с вложенным листом (nested list).

```
sum_num = 0
for t in num:
    sum_num = sum_num + t
avg = sum_num / len(num)
```

Ниже показан пример создания небольшого вложенного списка и итерация через его члены при помощи цикла for. Используйте данный пример и пример, представленный выше для решения данного задания:

```
my_list = [["one", "two"], ["1", "2"]]
for i in range(0, len(data)):
    print(my_list[i][1])
```

Расчет среднего значения по колонке Age отдельно для мужчин и женщин с помощью условий if-else. Основная идея: аналогично прошлому заданию ввести переменные для мужчин и женщин, создать счётчик для каждой выборки и тот же алгоритм повторить с разветвлением относительно пола. Для этого может помочь оператор ветвления (if-else statement):

```
gender = "Male"
if gender == "Male":
    print("Male")
if gender == "Female":
    print("Female")
```

Построение таблицы, показывающей связь диабета и ожирения в виде связи между классами. Для решения этой задачи необходимо проверить соответствие каждого пациента из списка обоим условиям с помощью оператора if-else с использованием цикла. Далее следует получить четыре массива со значениями порядковых номеров пациентов и разместить их в таблице.

		Diabetes	
		Positive (1)	Negative (0)
Obesity	Positive (1)		
	Negative (0)		

Полученную таблицу следует вывести в консоль.

Занятие 2. ЗАГРУЗКА И НАСТРОЙКА МОДУЛЕЙ NUMPY, PANDAS, SCIKIT-LEARN, TENSORFLOW, KERAS

В данном уроке мы научимся устанавливать и настраивать ключевые библиотеки Python для анализа данных.

Обзор библиотек. NumPy – это расширение языка Python, добавляющее поддержку больших многомерных массивов и матриц вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами.

Pandas – это программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Эта библиотека в первую очередь работает с данными в табличной форме, так называемыми дата фреймами, построенными по аналогии с дата-фрейм объектами в языке программирования R.

Scikit-learn – это библиотека для машинного обучения на языке программирования Python с открытым исходным кодом. С помощью нее можно реализовать различные алгоритмы классификации, регрессии и кластеризации, которые построены на взаимодействии библиотек NumPy и SciPy с Python.

TensorFlow – это открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

Keras – это библиотека для Python с открытым исходным кодом, которая позволяет легко создавать нейронные сети. Она объединяет эффективные библиотеки численных вычислений Theano и TensorFlow и позволяет определять и обучать модели нейронных сетей в несколько коротких строк кода.

Знакомство с командной строкой. Для установки модулей используется команда:

```
$ pip install tabulate
```

Удаление пакета выполняется таким образом:

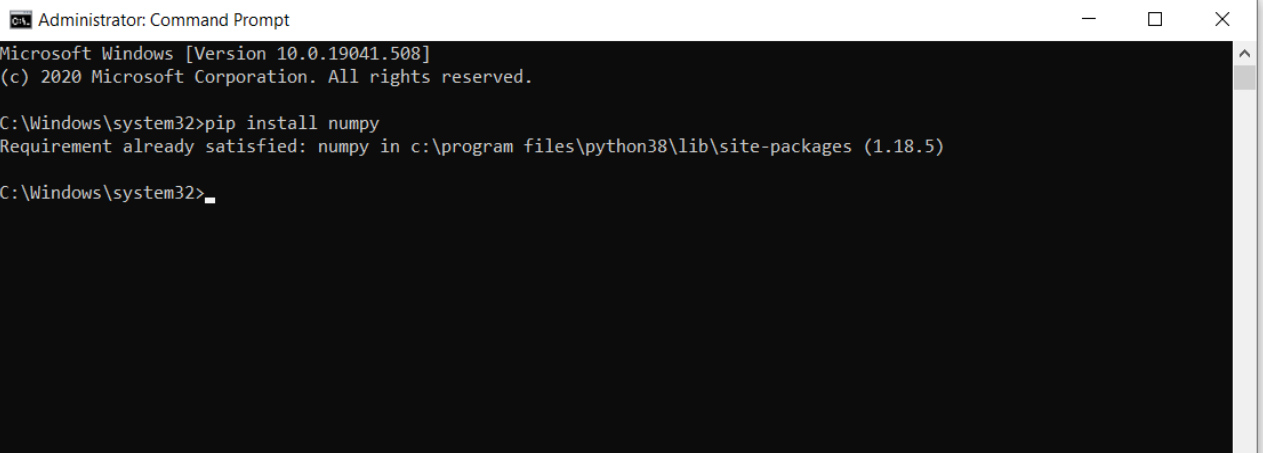
```
$ pip uninstall tabulate
```

Кроме того, иногда необходимо обновить пакет:

```
$ pip install --upgrade tabulate
```

Возможно, вам потребуется загрузить командную строку с правами администратора, чтобы установить библиотеки на компьютер.

Если последняя версия библиотеки уже установлена, вы увидите следующее сообщение.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>pip install numpy
Requirement already satisfied: numpy in c:\program files\python38\lib\site-packages (1.18.5)

C:\Windows\system32>
```

Рис. 2.1.

Выполните команды `pip install` в `cmd` для каждой из представленных выше библиотек:

```
pip install numpy
```

```
pip install pandas
```

```
pip install sklearn
```

```
pip install tensorflow
```

```
pip install keras
```


Пошаговое написание кода. Попробуем применить полученные теоретические знания написав самостоятельно небольшую программу.

1. Создайте и откройте файл со скриптом «.py» с помощью IDLE.
2. Затем напишите код в .py файле, который импортирует каждую из библиотек (import pandas as pd и т.п).

```
import numpy as np
import pandas as pd
import sklearn
import tensorflow as tf
import keras
```

Импортировать можно как сам модуль (команды выше), так и отдельные части модуля: from <имя_модуля> import <список определений>.

Например, from sklearn.linear_model import LogisticRegression.

3. Теперь напишите код в .py файле, который выведет в консоль версии каждой из установленных библиотек.

Узнать версию всех установленных пакетов можно через cmd:

```
pip freeze
```

Чтобы узнать версию конкретного пакета, примените одну из представленных ниже команд:

```
pip freeze | findstr numpy
pip freeze | findstr pandas
pip freeze | findstr sklearn
pip freeze | findstr tensorflow
pip freeze | findstr keras
```

Найдите возможность узнать версию библиотеки при помощи команды в скрипте Python. Напишите эту команду для каждой библиотеки.

4. Откройте в Python csv файл, загруженный по адресу: https://archive.ics.uci.edu/ml/machine-learning-databases/00529/diabetes_data_upload.csv с помощью специальной команды Pandas:

```
import pandas as pd
table = pd.read_csv('diabetes_data_upload.csv')
```

Как мы видим, pandas позволяет намного быстрее и проще осуществлять открытие csv файлов.

5. Рассчитайте среднее по колонке Age, используя средства numpy/pandas для всех людей и отдельно для мужчин и женщин. Выведите полученные значения в консоль. Сравните данные расчеты с теми, которые были сделаны с использованием встроенных средств Python.

6. Используя следующие методы модуля time рассчитайте и выведите в консоль время выполнения операции из пункта 7, а также аналогичного расчета из упражнения 1.

```
import time
start = time.time()
print("hello world") # ваш исполняемый код
end = time.time()
print(end - start)
```

Занятие 3. NUMPY ARRAYS AND FUNCTIONS

Работа с массивами данных. NumPy – это расширение языка Python, добавляющее поддержку больших многомерных массивов и матриц вместе с большой библиотекой высокоуровневых и очень быстрых математических функций для операций с этими массивами. Основным объектом NumPy является однородный многомерный массив элементов одного типа `numpy.ndarray`.

1. Создайте двумерный массив `numpy`, длина которого равна вашему номеру в списке группы, а ширина – номеру первой буквы вашей фамилии.

В NumPy существуют разные способы создать массив:

а) используя функцию `numpy.array()`:

```
import numpy as np
a=np.array([1,2,3,4,5,6,7])
```

Можно также переопределить тип в момент создания:

```
b = np.array([[1.5, 2, 3], [4, 5, 6]], dtype=np.complex)
```

б) иногда элементы массива бывают неизвестны, а массив, в котором они будут храниться, уже нужен.

Имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива `float64`).

Функция `zeros()` создает массив из нулей, а функция `ones()` – массив из единиц:

```
np.zeros((3, 5))
np.ones((4, 2))
```

Оценить размер массива можно с помощью функции `ndarray.size`:

```
v=a.size
```

2. Создайте массивы определенного в пункте 1 размера для равномерно и нормально распределенных случайных чисел с помощью случайных генераторов в NumPy.

rand (d0, d1, ..., dn)	Массив случайных значений
randn (d0, d1, ..., dn)	Массив нормально распределённых случайных значений
randint (low[, high, size, dtype])	Массив случайных целых чисел от low (включая) до high (не включая)
random_integers (low[, high, size])	Массив случайных целых чисел типа np.int от low (включая) до high (не включая)
random_sample ([size])	Массив случайных чисел с плавающей запятой в полуоткрытом интервале [0.0, 1.0).
random ([size])	Массив случайных чисел с плавающей запятой в полуоткрытом интервале [0.0, 1.0).
ranf ([size])	Массив случайных чисел с плавающей запятой в полуоткрытом интервале [0.0, 1.0).
sample ([size])	Массив случайных чисел с плавающей запятой в полуоткрытом интервале [0.0, 1.0).
choice (a[, size, replace, p])	Массив случайных чисел из заданного одномерного массива
bytes (length)	Массив случайных байтов

В таблице приведены генераторы массивов случайных чисел с разным распределением вероятностей ...

Функции NumPy для изучения данных. Главный объект NumPy – это однородный многомерный массив. Это таблица элементов (обычно чисел) одного типа, проиндексированных набором неотрицательных целых чисел. В NumPy измерения называются *осями*.

Например, координаты точки в трехмерном пространстве [1, 2, 1] имеют одну ось. Эта ось имеет 3 элемента, поэтому мы говорим, что она имеет длину 3. В примере, изображенном ниже, массив имеет 2 оси. Первая ось имеет длину 2, вторая ось имеет длину 3.

```
[[ 1., 0., 0.],  
 [ 0., 1., 2.]]
```

Класс массива NumPy называется `ndarray`. Он также известен по массиву псевдонимов. Обратите внимание, что `numpy.array` – это не то же самое, что класс `array.array` стандартной библиотеки Python, который обрабатывает только одномерные массивы и предлагает меньшую функциональность.

Наиболее важные атрибуты объекта `ndarray`:

- `ndarray.ndim` – Количество осей (размерность) массива.
- `ndarray.shape` – Размер массива. Это кортеж целых чисел, указывающий размер массива в каждом измерении. Для матрицы с *n* строками и *m* столбцами форма будет (*n*, *m*). Таким образом, длина кортежа массива – это количество осей, `ndim`.
- `ndarray.size` – Общее количество элементов массива, равное произведению элементов `shape`.
- `ndarray.dtype` – Объект, описывающий тип элементов в массиве. Можно создать или указать `dtype`, используя стандартные типы Python. Кроме того, NumPy предоставляет собственные типы, например, `numpy.int32`, `numpy.int16` и `numpy.float64`.
- `ndarray.itemsize` – Размер в байтах каждого элемента массива. Например, массив элементов типа `float64` имеет размер элемента 8 (= 64/8), а массив элементов типа `complex32` имеет размер элемента 4 (= 32/8). Это эквивалент `ndarray.dtype.itemsize`.
- `ndarray.data` – Буфер, содержащий фактические элементы массива. Обычно нам не нужно использовать этот атрибут, потому что мы будем обращаться к элементам в массиве с помощью средств индексирования.

Изучите ваш массив случайных чисел с помощью данных функций. Выведите результаты применения данных функций в консоль.

Создание функций в Python с помощью def. Функция – это блок кода, который начинается с ключевого слова `def`, названия функции и двоеточия:

```
def add(x, y):  
    return x + y
```

Инструкция `return` говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму `x` и `y`.

- функции создаются с помощью зарезервированного слова `def`;
- за `def` следуют имя функции и круглые скобки;
- внутри скобок могут указываться параметры, которые функция принимает;
- после круглых скобок идет двоеточие и с новой строки, с отступом, идет блок кода, который выполняет функция;
- первой строкой, опционально, может быть комментарий, так называемая `docstring`;
- в функциях может использоваться оператор `return`, он используется для прекращения работы функции и выхода из нее;
- чаще всего оператор `return` возвращает значение какой-либо переменной.

Когда функция только создана, она ещё ничего не выполняет. Действия, которые в ней перечислены, будут выполнены только после вызова. При вызове функции нужно указать её имя и передать аргументы, если функция имеет входные аргументы.

Знакомство с анализом данных. Анализ данных прежде всего начинается с определения размерностей и типов входных данных, для этого потребуется:

1. Создать в Python функцию, которая принимает NumPy массив и выводит в консоль результаты запуска соответствующих команд:

```
a.shape  
a.ndim
```

a.dtype.name

a.itemsize

a.size

Запустите эту функцию из другого места (консоль или другой скрипт).

2. Откройте файл по ссылке https://archive.ics.uci.edu/ml/machine-learning-databases/00529/diabetes_data_upload.csv с помощью библиотеки pandas, определите тип каждого из столбцов данной таблицы средствами numpy и запишите полученные типы колонок в отдельный csv файл.

3. Многие реализации алгоритмов машинного обучения требуют представление качественных входных переменных в виде чисел (0 – класс 0, 1 – класс 1 и т.д.) вместо буквенных обозначений. Переконвертируйте средствами numpy входную таблицу таким образом, чтобы таблица не содержала буквенных обозначений классов. Сохраните полученную таблицу в формате csv и повторите для нее пункт 6 еще раз.

Задание 4. РАБОТА С ТАБЛИЧНЫМИ ДАННЫМИ В БИБЛИОТЕКЕ PANDAS

Структура данных в библиотеке Pandas. Библиотека pandas предоставляет две структуры: Series и DataFrame для быстрой и удобной работы с данными.

Series – это маркированная одномерная структура данных, ее можно представить как таблицу с одной строкой. С Series можно работать как с обычным массивом (обращаться по номеру индекса), и как с ассоциированным массивом (использовать ключ для доступа к элементам данных).

DataFrame – это двумерная маркированная структура. Идейно она очень похожа на обычную таблицу, что выражается в способе ее создания и работе с элементами. Изначально с DataFrame начали работать на языке программирования R, и это оказалось настолько удобным, что оттуда функционал был скопирован в Python.

Импортируемые типы данных. Pandas поддерживает импорт данных из различных файлов. Поддерживаемые форматы указаны в таблице.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather

Продолжение таблицы

binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

Для загрузки файлов используется команда `pd.read_csv()`:

```
import pandas as pd
```

```
df = pd.read_csv('diabetes_data_upload.csv')
```

Откройте файл по ссылке https://archive.ics.uci.edu/ml/machine-learning-databases/00529/diabetes_data_upload.csv с помощью библиотеки `pandas`.

Исследование массивов средствами *Pandas*. Массив данных можно исследовать с помощью базовых функций:

`df.dtypes` – тип данных каждого столбца;

`df.head(n=5)` – вывод первых `n` строк массива, по умолчанию `n=5`.

Это полезно для быстрой проверки, содержит ли ваш объект данные нужного типа. Для отрицательных значений `n` эта функция возвращает все строки, кроме последних `n` строк, что эквивалентно `df[:n]`;

```
>>> df.head()
   Age Gender Polyuria Polydipsia ... muscle stiffness Alopecia Obesity  class
0   40   Male      No        Yes  ...      Yes      Yes      Yes  Positive
1   58   Male      No        No   ...      No      Yes      No  Positive
2   41   Male     Yes        No   ...      Yes      Yes      No  Positive
3   45   Male      No        No   ...      No      No      No  Positive
4   60   Male     Yes        Yes  ...      Yes      Yes      Yes  Positive

[5 rows x 17 columns]
```

Рис. 4.1.

`df.tail(3)` – аналогична предыдущей, выводит последние строки массива, по умолчанию – пять. Для отрицательных значений `n` эта функция возвращает все строки, кроме первых `n` строк, что эквивалентно `df[n:]`;

```
>>> df.tail(3)
   Age Gender Polyuria ... Alopecia Obesity  class
517  58  Female     Yes  ...      No      Yes  Positive
518  32  Female     No   ...      Yes      No  Negative
519  42   Male     No   ...      No      No  Negative

[3 rows x 17 columns]
```

Рис. 4.2.

`df.index` – индекс (метки строк).

```
>>> df.index
RangeIndex(start=0, stop=520, step=1)
--
```

Рис. 4.3.

`df.to_numpy()` – перевод массива в NumPy.

```
>>> df.to_numpy()
array([[40, 'Male', 'No', ..., 'Yes', 'Yes', 'Positive'],
       [58, 'Male', 'No', ..., 'Yes', 'No', 'Positive'],
       [41, 'Male', 'Yes', ..., 'Yes', 'No', 'Positive'],
       ...,
       [58, 'Female', 'Yes', ..., 'No', 'Yes', 'Positive'],
       [32, 'Female', 'No', ..., 'Yes', 'No', 'Negative'],
       [42, 'Male', 'No', ..., 'No', 'No', 'Negative']], dtype=object)
```

Рис. 4.4.

`df.describe()` – описательная статистика. Анализирует как числовые, так и объектные серии, а также наборы столбцов DataFrame смешанных типов данных;

```
>>> df.describe()
Age
count    520.000000
mean     48.028846
std      12.151466
min      16.000000
25%      39.000000
50%      47.500000
75%      57.000000
max      90.000000
--
```

Рис. 4.5.

df.T – транспонирование таблицы.

```
>>> df.T
      0      1      2      ...      517      518      519
Age      40      58      41      ...      58      32      42
Gender    Male    Male    Male      ...    Female    Female    Male
Polyuria      No      No      Yes      ...      Yes      No      No
Polydipsia     Yes      No      No      ...      Yes      No      No
sudden weight loss  No      No      No      ...      Yes      No      No
weakness       Yes      Yes      Yes      ...      Yes      Yes      No
Polyphagia     No      No      Yes      ...      Yes      No      No
Genital thrush  No      No      No      ...      No      No      No
visual blurring  No      Yes      No      ...      Yes      Yes      No
Itching        Yes      No      Yes      ...      No      Yes      No
Irritability   No      No      No      ...      No      No      No
delayed healing  Yes      No      Yes      ...      No      Yes      No
partial paresis  No      Yes      No      ...      Yes      No      No
muscle stiffness  Yes      No      Yes      ...      Yes      No      No
Alopecia       Yes      Yes      Yes      ...      No      Yes      No
Obesity        Yes      No      No      ...      Yes      No      No
class      Positive Positive Positive ... Positive Negative Negative

[17 rows x 520 columns]
```

Рис. 4.6.

Напишите функцию, которая принимает на вход DataFrame pandas, и выводит в консоль информацию об индексах, типах данных, описательную статистику и первые 5 строк для первых N+2 столбцов матрицы (здесь и далее N – ваш номер в списке группы). Используйте данную функцию для описания дата фрейма, загруженного в пункте 1.

Работа с ячейками. Выделение определенных ячеек. df['Age'] возвращает столбец Age.

```
>>> df['Age']
0      40
1      58
2      41
3      45
4      60
..
515    39
516    48
517    58
518    32
519    42
Name: Age, Length: 520, dtype: int64
```

Рис. 4.7.

`df[0:3]` возвращает строки от 0 до 3 (не включительно).

```
>>> df[0:3]
   Age Gender Polyuria Polydipsia ... muscle stiffness Alopecia Obesity  class
0   40  Male      No          Yes ...           Yes      Yes      Yes Positive
1   58  Male      No          No  ...           No      Yes      No  Positive
2   41  Male     Yes          No  ...           Yes      Yes      No  Positive

[3 rows x 17 columns]
```

Рис. 4.8.

`df.loc[0]` возвращает строку с индексом в виде Series, `df[[0]]` возвращает строку с индексом в виде DataFrame. 0 рассматривается как метка строки.

```
>>> df.iloc[3]
Age      45
Gender    Male
Polyuria  No
Polydipsia No
sudden weight loss  Yes
weakness      Yes
Polyphagia    Yes
Genital thrush  Yes
visual blurring No
Itching        Yes
Irritability   No
delayed healing  Yes
partial paresis No
muscle stiffness No
Alopecia        No
Obesity         No
class          Positive
Name: 3, dtype: object
>>> df.iloc[[3]]
   Age Gender Polyuria Polydipsia ... muscle stiffness Alopecia Obesity  class
3   45  Male      No          No  ...           No      No      No  Positive

[1 rows x 17 columns]
```

Рис. 4.9.

`df.loc[:, ['A', 'B']]` возвращает все строки в столбцах A и B.

```
>>> df.loc[:,['Age','Polydipsia']]
      Age Polydipsia
0       40        Yes
1       58        No
2       41        No
3       45        No
4       60        Yes
..     ...      ...
515     39        Yes
516     48        Yes
517     58        Yes
518     32        No
519     42        No

[520 rows x 2 columns]
```

Рис. 4.10.

`df.iloc[3]` возвращает третью строку в виде Series. `[[3]]` возвращает результат в виде DataFrame.

```
>>> df.iloc[3]
Age                45
Gender             Male
Polyuria           No
Polydipsia         No
sudden weight loss  Yes
weakness           Yes
Polyphagia         Yes
Genital thrush     Yes
visual blurring    No
Itching            Yes
Irritability       No
delayed healing    Yes
partial paresis    No
muscle stiffness   No
Alopecia           No
Obesity            No
class              Positive
Name: 3, dtype: object
>>> df.iloc[[3]]
      Age Gender Polyuria Polydipsia ... muscle stiffness Alopecia Obesity      class
3     45   Male      No        No ...                No        No      No  Positive

[1 rows x 17 columns]
```

Рис. 4.11.

`df[df['A'] > 0]` возвращает DataFrame с теми элементами, которые удовлетворяют условиям.

```
>>> df[df['Age']>50]
   Age  Gender Polyuria  ... Alopecia Obesity  class
1    58   Male      No  ...      Yes      No  Positive
4    60   Male      Yes  ...      Yes      Yes  Positive
5    55   Male      Yes  ...      Yes      Yes  Positive
6    57   Male      Yes  ...      No       No  Positive
7    66   Male      Yes  ...      No       No  Positive
..  ...   ...      ...  ...      ...      ...   ...
510  67   Male      No  ...      Yes      No  Negative
511  66   Male      No  ...      Yes      No  Negative
513  62  Female      Yes  ...      No      Yes  Positive
514  54  Female      Yes  ...      No      No  Positive
517  58  Female      Yes  ...      No      Yes  Positive

[207 rows x 17 columns]
```

Рис. 4.12.

`df.loc['2013-01-02':'2013-01-04']` возвращает строки в диапазоне от 02.10.2013 до 04.01.2013.

Создайте два дата фрейма, полученных путем определения в один из них только строк со значением Yes в колонке N+1, а в другой – только со значением No.

Сортировка данных. В pandas реализованы методы сортировки данных.

`df.sort_index(axis=1, ascending=False)` – сортирует значения в таблице по первой строке.

```
>>> df.sort_index(axis=1, ascending=False)
   weakness visual blurring sudden weight loss  ... Gender Alopecia Age
0      Yes          No          No          No  ...   Male      Yes  40
1      Yes          Yes          No          No  ...   Male      Yes  58
2      Yes          No          No          No  ...   Male      Yes  41
3      Yes          No          Yes          Yes  ...   Male      No  45
4      Yes          Yes          Yes          Yes  ...   Male      Yes  60
..  ...      ...      ...      ...      ...   ...      ...   ..
515     No          No          Yes          Yes  ...  Female      No  39
516     Yes          No          Yes          Yes  ...  Female      No  48
517     Yes          Yes          Yes          Yes  ...  Female      No  58
518     Yes          Yes          No          No  ...  Female      Yes  32
519     No          No          No          No  ...   Male      No  42

[520 rows x 17 columns]
```

Рис. 4.13.

`df.sort_values(by='Age')` – сортирует значения в таблице по первому столбцу.

```
>>> df.sort_values(by=['Age'])
      Age  Gender Polyuria  ... Alopecia Obesity  class
133    16   Male      Yes  ...      No      No  Positive
68     25  Female      No  ...      Yes      No  Positive
108    25   Male      Yes  ...      Yes      No  Positive
235    26   Male      No  ...      No      No  Negative
286    27   Male      No  ...      No      No  Negative
..     ..   ..      ...  ...      ...      ...      ...
113    79   Male      No  ...      No      No  Positive
185    85   Male      Yes  ...      No      No  Positive
101    85   Male      Yes  ...      No      No  Positive
102    90  Female      No  ...      Yes      No  Positive
186    90  Female      No  ...      Yes      No  Positive

[520 rows x 17 columns]
```

Рис. 4.14.

Выполните сортировку вашей исходной таблицы по нескольким ключам: первый ключ – колонка N+1, второй класс – колонка – N+2, третий ключ – возраст (Age). Сохраните результат в виде отдельного DataFrame.

Обработка исключений. Работа с пропущенными данными реализуется с помощью функций:

`df.dropna(how='any')` – удаляет столбцы, где пропущен хоть один элемент;

`df.fillna(value=5)` – отсутствующие значения заменяются указанным значением;

`pd.isna(df)` – обнаруживает неуказанные значения, на выходе получается таблица со значениями True, False:

```
>>> pd.isna(df)
   Age  Gender  Polyuria  ...  Alopecia  Obesity  class
0  False  False    False  ...    False    False  False
1  False  False    False  ...    False    False  False
2  False  False    False  ...    False    False  False
3  False  False    False  ...    False    False  False
4  False  False    False  ...    False    False  False
...    ...    ...    ...  ...    ...    ...    ...
515  False  False    False  ...    False    False  False
516  False  False    False  ...    False    False  False
517  False  False    False  ...    False    False  False
518  False  False    False  ...    False    False  False
519  False  False    False  ...    False    False  False

[520 rows x 17 columns]
```

Рис. 4.15.

Проверьте, есть ли в ваших данных пропущенные значения и удалите строки, в которых есть хотя бы один пропуск.

Визуализация данных. Методами pandas можно построить графики и гистограммы.

Для рисования графиков требуется подключить библиотеку `matplotlib.pyplot`, добавив следующую строку в начале программы:

```
import matplotlib.pyplot as plt
```

Для рисования простых двумерных графиков используйте функцию:

```
plt.plot(x, y, linewidth=2.0),
```

где `x` отвечает за координаты по горизонтальной оси, `y` – соответственно, за вертикальную ось, а параметр `linewidth` позволяет определить внешний вид маркера.

`plt.plot([1,2,3],[1,2,3], 'r--')` – красная прерывистая линия;
`plt.plot([1,2,3],[1,2,3], 'bs')` – синие квадратики; `plt.plot([1,2,3],[1,2,3], 'g^')` – зеленые галочки.

Виды маркеров можно посмотреть здесь:

https://matplotlib.org/api/markers_api.html

Для подписей осей используются следующие функции:

```
plt.ylabel('Название вертикальной оси'); plt.xlabel('Название горизонтальной оси').
```

Чтобы добавить название на график, используйте:

```
plt.title('Название графика').
```

Если требуется ограничить область отображения графика, можно воспользоваться функцией

```
plt.axis([x_min, x_max, y_min, y_max])
```

Для отображения полученного графика в отдельном окне используйте

```
plt.show()
```

Часто требуется отобразить не один график, а несколько, расположив их рядом или один под другим. Для этого используют метод:

```
plt.figure(),
```

где в качестве параметра прописывается порядковый номер формы (окна).

Чтобы красиво компоновать графики, используйте метод

```
plt.subplot(ijk),
```

где *i* – количество строк с графиками, *j* – столбцов с графиками и *k* – порядковый номер графика в текущем окне (figure).

Для построения гистограммы используется функция:

```
plt.hist()
```

Код программы, необходимый для построения первых трех столбцов исследуемой диаграммы:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('diabetes_data_upload.csv')
df1 = df.iloc[:,0:3]
plt.hist(df1.iloc[:,0])
plt.show()
plt.hist(df1.iloc[:,1])
plt.show()
plt.hist(df1.iloc[:,2])
```

```
plt.show()
```

Постройте гистограммы распределения Age для двух таблиц, получившихся при разделении данных в пункте 3 (по возможности на соседних полях одного окна, используя функцию subplot). Проанализируйте данные графики и сделайте выводы.

Ящик с усами и скатерограмма. Порой требуется визуализировать не только кривую данных во времени, но и посмотреть разброс относительно среднего, оценить остатки линейной регрессии, для чего используются ящик с усами и скатерограмма.

1. Методами pandas можно построить ящик с усами (boxplot).

Для построения ящика с усами можно воспользоваться следующей функцией:

```
plt.boxplot()
```

Тогда код для первых пяти столбцов будет выглядеть так:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('diabetes_data_upload.csv')
df2 = df.iloc[:,0:5]
plt.boxplot(df2.iloc[:,0])
plt.show()
```

Постройте boxplot распределения Age для двух таблиц, получившихся при разделении данных в пункте 3 (оба ящика с усами должны быть на одном полотне, друг рядом с другом). Проанализируйте данные графики и сделайте выводы.

2. Методами pandas можно также построить scatter matrix. Пример ниже иллюстрирует построение scatter matrix по последним пяти столбцам.

```
pd.plotting.scatter_matrix(df.iloc[:, -5:-1])
plt.show()
```

Постройте scatter matrix для колонок Age, колонки N+1, и колонки N+2, закодировов цветом переменную class (Positive – красным, Negative – синим цветом).

Упражнение 5. МАШИННОЕ ОБУЧЕНИЕ С БИБЛИОТЕКОЙ SCIKIT-LEARN

Введение в работу с модулями машинного обучения. Scikit-learn – свободная библиотека для машинного обучения в Python. Scikit-learn включает различные алгоритмы классификации, регрессии и кластеризации, включая метод опорных векторов, случайный лес (random forest), градиентный бустинг, k-средних и DBSCAN. Разработан для взаимной работы с численными и научными библиотеками NumPy и SciPy. Библиотека Scikit-learn – самый распространенный выбор для решения задач классического машинного обучения.

В данном упражнении мы рассмотрим последовательность шагов для создания модели-классификатора, определяющей наличие у человека сахарного диабета по анамнезу человека.

Убедитесь, что библиотека установлена на компьютере. Создайте новый ru-файл и импортируйте в нем библиотеку scikit-learn:

```
import sklearn
```

Как правило, мы будем импортировать отдельные модули библиотеки, например:

```
from sklearn import preprocessing
```

Предобработка данных. Для многих алгоритмов машинного обучения необходимо сделать предобработку данных. Для начала загрузите файл по ссылке https://archive.ics.uci.edu/ml/machine-learning-databases/00529/diabetes_data_upload.csv с помощью библиотеки Pandas.

1. Методы библиотеки Scikit-learn работают с NumPy массивами. Преобразовать дата фрейм Pandas в NumPy array можно при помощи метода values:

```
df.values
```

2. Для дальнейшей работы нужно разделить переменные в массивах на входные (X) и выходные (Y). В данном случае выходная переменная – наличие у человека сахарного диабета, а входные – все остальные. Используйте изученные вами методы NumPy или Pandas для разделения таблицы с переменными на X и Y.

3. Для численных переменных может понадобиться стандартизация (на практике мы часто игнорируем форму распределения и просто преобразовываем данные, удаляя среднее значение каждого объекта, а затем масштабируем его, деля переменные на их стандартное отклонение). Адаптируйте код ниже и используйте его, чтобы стандартизировать значения переменной Age в ваших данных:

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1., 2.],
...                     [ 2., 0., 0.],
...                     [ 0., 1., -1.]])
>>> X_scaled = preprocessing.scale(X_train)
>>> X_scaled
array([[ 0. ..., -1.22..., 1.33...],
       [ 1.22..., 0. ..., -0.26...],
       [-1.22..., 1.22..., -1.06...]])
```

4. Для логистической регрессии необходимо, чтобы качественные переменные были приведены к виду бинарных переменных (dummy-переменные, в которых 1 – наличие определенного класса, 0 – его отсутствие) посредством операции one-hot encoding. Преобразуйте приведенный ниже код для того, чтобы закодировать категориальные переменные в ваших данных, а также перевести их в бинарные качественные переменные:

```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses
Firefox']]
>>> enc.fit(X)
OneHotEncoder(categorical_features=None, categories=None, drop=None,
              dtype=<... 'numpy.float64'>, handle_unknown='error',
              n_values=None, sparse=True)
>>> enc.transform([['female', 'from US', 'uses Safari'],
...               ['male', 'from Europe', 'uses Safari']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

Сохраните полученные после данных преобразований данные в виде двух csv таблиц: X.csv, Y.csv любым известным вам методом.

Подготовка тестовой и обучающей выборки. В scikit-learn реализовано множество методов разделения данных на обучающую и тестовую выборки. Разделите ваши данные на обучающие (X_train, Y_train) и тестовые (X_test, Y_test) в пропорции 80 %/20 % при помощи следующей функции, указав random_state равным вашему номеру в списке группы:

```
>>> X_train, X_test, Y_train, Y_test = train_test_split(  
...   X, Y, test_size=test_size, random_state= random_state)
```

Проверьте размерности всех полученных массивов при помощи метода NumPy shape.

Обучение модели. Определяя значение качественной переменной (наличие заболевания), мы решаем задачу классификации. В библиотеки scikit-learn имеются реализации большого количества классификаторов. Рассмотрим один из простейших классификаторов – логистическую регрессию и натренируем модель при помощи команды fit() на обучающей выборке:

```
from sklearn.linear_model import LogisticRegression  
model = LogisticRegression(random_state=0).fit()
```

Подставьте нужные аргументы в метод fit().

8. Используя метод predict, предскажем наличие диабета у пациентов из тестовой выборки:

```
Y_pred = model.predict()
```

Подставьте нужные аргументы в метод predict().

Оценка точности модели. Оценить качество предсказания можно при помощи различных метрик (scores), которые реализованы в библиотеке. Оцените, насколько хорошо ваша модель справляется с задачей при помощи accuracy_score и еще одной метрики, подходящей для бинарной классификации.

```
from sklearn.metrics import accuracy_score  
print(accuracy_score(Y_test, Y_pred))
```

Оцените значения полученных метрик. Можно ли считать данную метрику достаточной для данного датасета? При желании можно также проанализировать другие метрики, построить ROC-кривую и матрицу ошибок (confusion matrix).

Улучшение модели. Попробуйте улучшить качество классификации путем задания в модели (аргументом функции `LogisticRegression()`) параметра регуляризации модели `C`. Можно брать его из следующего интервала:

```
param_range = [100, 10, 1, 0.1, 0.01, 0.001]
```

При каком `C` была получена наилучшая точность?

При желании можно осуществить подбор параметра автоматизировано с помощью k-fold кросс-валидации (используя метод `GridSearchCV()`). Можно также объединить все операции с моделью (включая предобработку) в единый pipeline (при помощи команды `make_pipeline()`).

Упражнение 6. СОЗДАНИЕ КЛАССИФИКАТОРА С ПОМОЩЬЮ БИБЛИОТЕКИ PYTORCH

Введение в облачные вычисления. PyTorch – фреймворк машинного обучения для языка Python с открытым исходным кодом, созданная на базе Torch. Используется для решения различных задач: компьютерное зрение, обработка естественного языка.

В данном упражнении мы научимся строить простейшие нейронные сети для решения задачи регрессии. Это упражнение лучше выполнять на удаленном сервере <https://colab.research.google.com/>, поскольку вычисления на центральном процессоре (CPU) могут занять значительное количество времени, а настройка драйверов видеокарты (GPU) для вычислений может быть сложной задачей.

Google Colab — это бесплатный облачный сервис на основе Jupyter Notebook. Google Colab предоставляет необходимые средства для создания и оценки моделей машинного обучения браузером и бесплатный доступ к быстрым GPU и TPU.

Практическая реализация модели машинного обучения. Выполнив следующие шаги, мы попытаемся создать и обучить модель машинного обучения на базе Google Colab.

1. Создайте новый блокнот по ссылке <https://colab.research.google.com/>.
2. Проверьте, имеется ли на сервере доступ к видеокарте и переведите pytorch в режим вычислений на GPU, если GPU доступна при помощи следующего кода.

```
import torch
train_on_gpu = torch.cuda.is_available()
if not train_on_gpu:
    print('CUDA is not available. Training on CPU ...')
else:
    print('CUDA is available! Training on GPU ...')
torch.cuda.is_available()
```

3. Сгенерируйте тренировочные данные и отобразите их с помощью matplotlib. При этом в качестве N здесь и в дальнейшем коде необходимо установить ваш номер в списке группы.

```
from sklearn.datasets import make_moons
from matplotlib import pyplot as plt
X, y = make_moons(n_samples=5000, random_state=1, noise=0.1)
plt.figure(figsize=(16, 10))
plt.title("Dataset")
plt.scatter(X[:, 0], X[:, 1], c=y, cmap="summer")
plt.show()
```

4. Разделите данные на тестовую и тренировочную выборки с помощью sklearn.

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=N)
```

5. Загрузим датасет в PyTorch. В PyTorch загрузка данных как правило происходит без загрузки в оперативную память. Для этого используются две сущности Dataset и DataLoader. Dataset загружает каждый объект по отдельности. DataLoader группирует объекты из Dataset в батчи (отдельные части датасета определенного размера, равного batch_size). Это важно при работе с изображениями и большими объемами данных.

```
from torch.utils.data import TensorDataset, DataLoader
X_train_t = torch.FloatTensor(X_train)
y_train_t = torch.FloatTensor(y_train)
X_val_t = torch.FloatTensor(X_val)
y_val_t = torch.FloatTensor(y_val)
train_dataset = TensorDataset(X_train_t, y_train_t)
val_dataset = TensorDataset(X_val_t, y_val_t)
train_dataloader = DataLoader(train_dataset, batch_size=128)
val_dataloader = DataLoader(val_dataset, batch_size=128)
```

6. Создадим простейший классификатор на основе логистической регрессии для задачи бинарной классификации. Мы создаем данных класс на основе класса nn.Module из PyTorch.

```
class LinearRegression(nn.Module):
```



```

def __init__(self, in_features: int, out_features: int, bias: bool = True):
    super().__init__()
    self.weights = nn.Parameter(torch.Tensor(in_features, out_features))
    self.bias = bias
    if bias:
        self.bias_term = nn.Parameter(torch.randn(out_features))
def forward(self, x):
    x = x@self.weights
    if self.bias:
        x += self.bias_term
    return x

```

7. Определим параметры обучения нашего классификатора. В качестве функции потерь используем `nn.BCEWithLogitsLoss` (<https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html>), в качестве оптимизатора возьмем стандартный градиентный спуск (<https://pytorch.org/docs/stable/optim.html>). В качестве аргументов `LinearRegression` нужно задать количество входных (первый аргумент) и выходных (второй аргумент) переменных функции (по аналогии со слоем нейронной сети).

```

from sklearn.model_selection import train_test_split
linear_regression = LinearRegression(2, 1)
loss_function = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(linear_regression.parameters(), lr=0.01*N)
list(linear_regression.parameters())

```

8. Создадим основной цикл тренировки модели, включающий отображение информации о процессе обучения (логирование). На каждой итерации мы рассчитываем градиенты с помощью оптимизатора и потери с помощью функции потерь. В соответствии с этими данными мы осуществляем изменение весов модели в сторону снижения потерь.

```

losses = []
max_epochs = 5*N
prev_weights = torch.zeros_like(linear_regression.weights)
stop_it = False
for epoch in range(max_epochs):

```

```

for it, (X_batch, y_batch) in enumerate(train_dataloader):
    optimizer.zero_grad()
    outp = linear_regression.forward(X_batch).squeeze(1)
    loss = loss_function(outp, y_batch)
    loss.backward()
    losses.append(loss.detach().flatten()[0])
    optimizer.step()
    probabilities = linear_regression.forward(X_batch)
    preds = (probabilities > 0.5).type(torch.long)
    batch_acc = (preds.flatten() ==
y_batch).type(torch.float32).sum()/y_batch.size(0)
    if it % 5 == 0:
        print(f"Iteration: {it + epoch*len(train_dataset)}\nBatch accuracy:
{batch_acc}")
        current_weights = linear_regression.weights.detach().clone()
        if (prev_weights - current_weights).abs().max() < tol:
            print(f"\nIteration: {it + epoch*len(train_dataset)}.Convergence.
Stopping iterations.")
            stop_it = True
            break
        prev_weights = current_weights
    if stop_it:
        break

```

9. Постройте и проанализируйте график изменения значений функции потерь с помощью кода ниже.

```

plt.figure(figsize=(12, 8))
plt.plot(range(len(losses)), losses)
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()

```

10. Оцените качество прогнозирования на тестовой выборке. Напишите, какое значение точности (accuracy_score) вы получили.

```

import numpy as np

```

```

def predict(dataloader, model):
    model.eval()
    predictions = np.array([])
    for x_batch, _ in dataloader:
        outp = model(x_batch)
        probs = torch.sigmoid(outp)
        preds = (probs > 0.5).type(torch.long)
        predictions = np.hstack((predictions, preds.numpy().flatten()))
    predictions = predictions
    return predictions.flatten()

from sklearn.metrics import accuracy_score
print(accuracy_score(predict(val_dataloader, linear_regression), y_val))

```

11. Попробуйте улучшить качество прогнозирования, используя различные параметры скорости обучения (lr), количества эпох (epochs), либо применив нейронную сеть с различными функциями активации.

Мы изучили основы создания и моделирования моделей в PyTorch. Используя данный цикл разработки, мы сможем создавать и отлаживать нейронные сети для решения различных задач.

Упражнение 7. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ PYTORCH ДЛЯ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

Введение в нейронные сети. Сверточные нейронные сети — вид нейронных сетей, которые хотя бы на одном из своих слоев в качестве преобразования используют операцию «свертки».

В данном упражнении мы научимся строить простейшие сверточные нейронные сети для решения задачи классификации изображений. Это упражнение лучше выполнять на удаленном сервере <https://colab.research.google.com/>, поскольку вычисления на центральном процессоре (CPU) могут занять значительное количество времени, а настройка драйверов видеокарты (GPU) для вычислений может быть сложной задачей.

Google Colab — это бесплатный облачный сервис на основе Jupyter Notebook. Google Colab предоставляет необходимые средства для создания и оценки моделей машинного обучения браузером и бесплатный доступ к быстрым GPU и TPU.

Практическая реализация сверточной нейронной сети. Выполнив следующие шаги, мы попытаемся создать нейронную сеть на базе Google Colab.

1. Создайте новый блокнот по ссылке <https://colab.research.google.com/>.
2. Проверьте, имеется ли на сервере доступ к видеокарте и переведите pytorch в режим вычислений на GPU, если GPU доступна при помощи следующего кода.

```
import torch
train_on_gpu = torch.cuda.is_available()
if not train_on_gpu:
    print('CUDA is not available. Training on CPU ...')
else:
    print('CUDA is available! Training on GPU ...')
DEVICE = torch.device("cuda")
```

3. В качестве задачи классификации возьмем классификацию изображений цифр от 0 до 9. Для этого используем датасет MNIST, который содержит черно-белые изображения цифр размером 28 на 28 пикселей, его

можно загрузить при помощи встроенных в библиотеку catalyst функций. Первая строка в коде ниже устанавливает библиотеку catalyst, которая загружает датасет MNIST. Функция DataLoader используется для того, чтобы загружать датасет отдельными фрагментами – батчами по 128 изображений. В качестве N здесь и в дальнейшем коде необходимо установить ваш номер в списке группы.

```
!pip install -q catalyst
!pip install -q catalyst
import os
import torch
import torchvision
from torch.utils.data import TensorDataset, DataLoader
from catalyst import utils
from catalyst.contrib.datasets import MNIST
from torch import nn
N = 1
utils.set_global_seed(N)
train_dataset = MNIST(root=os.getcwd(), train=True, download=True)
val_dataset = MNIST(root=os.getcwd(), train=False)
train_dataloader = DataLoader(train_dataset, batch_size=128)
val_dataloader = DataLoader(val_dataset, batch_size=128)
4. Создадим простейшую сверточную нейронную сеть.
class Identical(nn.Module):
    def forward(self, x):
        return x
class Flatten(nn.Module):
    def forward(self, x):
        batch_size = x.size(0)
        return x.view(batch_size, -1)
activation = Identical
model = nn.Sequential(
    Flatten(),
    nn.Linear(28*28, 128),
```

```

activation(),
nn.Linear(128, 128),
activation(),
nn.Linear(128, 10)
)

```

5. Зададим настройки обучения нейронной сети. В качестве функции минимизации потерь зададим кросс-энтропию, в качестве алгоритма оптимизации Adam.

```

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())
loaders = {"train": train_dataloader, "valid": val_dataloader}

```

6. Ниже представлен код для главного цикла обучения нейронной сети. Он в целом аналогичен коду тренировки сети из упражнения 6. Разберитесь в функциях, которые представлены в коде. Запустите код, посмотрите на вывод, который выводит алгоритм. Какую точность удалось получить на первой и последней итерациях обучения нейронной сети?

```

max_epochs = N
accuracy = {"train": [], "valid": []}
for epoch in range(max_epochs):
    epoch_correct = 0
    epoch_all = 0
    for k, dataloader in loaders.items():
        for x_batch, y_batch in dataloader:
            if k == "train":
                model.train()
                optimizer.zero_grad()
                outp = model(x_batch.float().unsqueeze(1))
            else:
                model.eval()
                with torch.no_grad():
                    outp = model(x_batch.float().unsqueeze(1))
            preds = outp.argmax(-1)
            correct = (preds == y_batch).sum()

```

```

all = len(y_batch)
epoch_correct += correct.item()
epoch_all += all
if k == "train":
    loss = criterion(outp, y_batch)
    loss.backward()
    optimizer.step()
if k == "train":
    print(f"Epoch: {epoch+1}")
    print(f"Loader:{k}.Accuracy: {epoch_correct/epoch_all}")
    accuracy[k].append(epoch_correct/epoch_all)

```

7. Попробуйте улучшить качество прогнозирования, используя различные параметры скорости обучения (lr), количества эпох (epochs), либо применив нейронную сеть с различными функциями активации. Подробнее о функциях активации можно прочитать по ссылке:

<https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity>

Попробуйте также различные алгоритмы оптимизации:

<https://pytorch.org/docs/stable/optim.html>

8. Используйте преднастроенные нейронные сети со сложной архитектурой для улучшения точности моделей. Эта техника называется transfer learning. В torchvision есть различные преднастроенные нейронные сети. Код ниже выбирает одну из таких сетей и оставляет «замороженными» три первых преднастроенных слоя (это означает, что коэффициенты на этих слоях не будут меняться; как правило, первые слои сверточных содержат информацию о маленьких деталях изображений, которые зачастую схожи для разных задач). В данном примере используется сеть resnet18, имеющая соединения быстрого доступа для борьбы с затуханием градиента (подробнее об использовании этой и других архитектур нейронных сетей можно прочитать в <https://pytorch.org/docs/stable/torchvision/models.html>). Загрузка коэффициентов модели может занять некоторое время.

```

model=torchvision.models.resnet18(pretrained=True)
.to(DEVICE)

```

```
ct = 0
for child in model.children():
    ct += 1
    if ct < 4:
        for param in child.parameters():
            param.requires_grad = False
```

Постарайтесь получить максимальную точность при оценке качества классификации на тестовой выборке.

Мы изучили основы создания и моделирования моделей в PyTorch. Используя данный цикл разработки, мы сможем создавать и отлаживать нейронные сети для решения более сложных задач.

ЧАСТЬ 2. Лабораторные работы.

Лабораторная работа 1. ГЕНЕРАЦИЯ МОДЕЛЬНЫХ НАБОРОВ ДАННЫХ

Цель: получение навыков работы с numpy-массивами и написание функций на языке Python на примере генерации массивов произвольно распределенных данных.

Для изучения различных классификаторов, их свойств и особенностей, создайте модельные данные, форму распределения и смешанность которых можно регулировать вручную.

Наборы формируются так, чтобы первый набор удовлетворял условиям применения текущего метода классификации/кластеризации, второй – нет. Вы можете поэкспериментировать и создать больше тестовых наборов для исследования ограничений метода. Также вы увидите, какие оценки эффективности наилучшим образом отвечают на вопрос – подходит ли выбранный метод для анализа имеющихся данных.

Используемые модули: numpy, matplotlib

Вспомогательная документация:
<https://numpy.org/doc/stable/contents.html#numpy-docs-mainpage>,
<https://matplotlib.org/3.3.1/users/index.html>

Ход работы

Перед началом работы создайте новую папку проекта, в которой будет храниться исходный код программы. В работе необходимо сгенерировать два массива с элементами различных классов по 1000 объектов в каждом в 4-х мерном пространстве признаков, а также массив с метками класса для них.

1. Создайте отдельный скрипт. Для начала, импортируйте модуль Numpy.

```
import numpy as np
```

2. Создадим переменные, распределение по нормальному закону с незначительно различными средними и дисперсиями. Эта форма

распределения характерна для случайных величин, подверженных большому числу случайных факторов и очень часто встречается в биологии и медицине. Задайте средние значения μ и стандартное отклонения σ для каждого из классов. Значения могут быть выбраны самостоятельно и отличаться от предложенных ниже.

```
mu0 = [0, 2, 3]
mu1 = [3, 5, 1]
sigma0 = [2, 1, 2]
sigma1 = [1, 2, 1]
```

3. Создайте переменные, соответствующие классам. Удобнее сделать это в цикле *for*, создавая столбец и конкатенируя (стыкуя) его с предыдущими.

```
N = 1000 # число объектов класса
col = len(mu0) # количество столбцов-признаков – длина массива средних
class0 = np.random.normal(mu0[0],sigma0[0],[N,1]) # инициализируем первый
столбец (в Python нумерация от 0)
class1 = np.random.normal(mu1[0],sigma1[0],[N,1])
```

```
for i in range(1,col): # подумайте, почему нумерация с 1, а не с 0
    v0 = np.random.normal(mu0[i],sigma0[i],[N,1])
    class0 = np.hstack((class0,v0))

    v1 = np.random.normal(mu1[i],sigma1[i],[N,1])
    class1 = np.hstack((class1,v1))
```

4. Создайте переменную, содержащую метки класса – логические 0 и 1.

Это можно сделать несколькими способами.

```
Y1 = np.ones((N, 1), dtype=bool)# массив логических единиц
```

```
Y0 = np.zeros((N, 1), dtype=bool)
```

Или:

```
Y1 = np.empty((N, 1), dtype=bool) # пустой массив размерности Nx1
```

```
Y1.fill(1) #заполнение пустого массива единицами
```

```
Y0 = np.empty((N, 1), dtype=bool)
```

```
Y0.fill(0)
```

Изучите самостоятельно оба подхода и используемые функции. Выберите любой вариант из представленных или предложите свой.

5. Сведите в единую переменную X объекты двух классов, а также метки двух классов в переменную Y . Не перепутайте порядок стыковки массивов, чтоб метки не поменялись на противоположные.

```
Y1 = np.ones((N, 1), dtype=bool)
```

```
Y0 = np.zeros((N, 1), dtype=bool)
```

```
X = np.vstack((class0,class1))
```

```
Y = np.vstack((Y0,Y1)).ravel() #ravel позволяет сделать массив плоским –  
одномерным, размера (N,), это необходимо для дальнейшего использования в  
классификаторах
```

6. Разделите данные на две части – это будут обучающая и тестовые подвыборки. Пропорции 70/30. Перед этим перемешайте («пошафлите») данные так, чтобы объекты из разных классов перемешались. Обратите внимание, что меняя местами элементы массива с данными X , необходимо точно также перемешать метки Y , чтобы не утратить соответствие между объектами и метками.

```
# перемешиваем данные
```

```
rng = np.random.default_rng()
```

```
arr = np.arange(2*N) #индексы для перемешивания
```

```
rng.shuffle(arr)
```

```
X = X[arr]
Y = Y[arr]
```

```
# разделяем данные на 2 подвыборки
trainCount = round(0.7*N*2) # не забываем округлить до целого
Xtrain = X[0:trainCount]
Xtest = X[trainCount:N*2+1]
Ytrain = Y[0:trainCount]
Ytest = Y[trainCount:N*2+1]
```

7. Визуализируйте результаты, чтобы оценить пересекаемость классов. Это можно сделать, используя гистограммы и диаграммы рассеяния (скатерограммы), средствами модуля `matplotlib`. Этот этап позволит увидеть, как распределены данные, и сделать выводы о возможностях их последующего анализа и необходимости коррекции свойств распределения.

```
import matplotlib.pyplot as plt # все импорты можно выносить в самое начало скрипта
```

```
# построение гистограмм распределения для всех признаков
for i in range(0, col):
    _ = plt.hist(class0[:, i], bins='auto', alpha=0.7) #параметр alpha позволяет задать
    прозрачность цвета
    _ = plt.hist(class1[:, i], bins='auto', alpha=0.7) plt.savefig('hist_'+str(i+1)+'.png')
# сохранение изображения в файл
plt.show()
```

```
# построение одной скатеррограммы по выбранным признакам
plt.scatter(class0[:,0], class0[:,2], marker=".", alpha=0.7)
plt.scatter(class1[:,0], class1[:,2], marker=".", alpha=0.7)
plt.savefig('scatter_'+str(i+1)+'.png')
plt.show()
```

8. Самостоятельно подпишите графики – название и оси. Как это сделать можно найти в документации.

9. Вынесите генерацию модельных данных из основного скрипта. Это позволит использовать ее в других проектах и обеспечит повторное использование кода без копирования. Создайте отдельный файл, назовите “DataGenerator.py”. Определим внутри него функцию *norm_dataset*, в которую перенесем весь код начиная с определения числа колонок, заканчивая шаблом переменных *X* и *Y*.

Файл DataGenerator.py начинается и заканчивается следующим образом:

```
import numpy as np

#объявляем функцию и определяем входные переменные
def norm_dataset(mu,sigma,N): # обозначение имя функции и входных
    аргументов
    # тело функции, обязательно с отступом!
    mu0 = mu[0]
    mu1 = mu[1]
    sigma0 = sigma[0]
    sigma1 = sigma[1]
    col = len(mu0) # количество столбцов-признаков
    .....
    X = X[arr]
    Y = Y[arr]
    return X, Y, class0, class1 # возвращаемые аргументы
```

10. Скопируйте в новый документ скрипт, с которым работали с начала лабораторной работы. Вместо генерации переменных *X* и *Y*, вставьте вызов новой функции. Для этого в начале импортируйте содержащий ее файл (DataGenerator.py), а также задайте переменные *mu* и *sigma* (можно использовать имеющиеся *mu0*, *mu1* и *sigma0*, *sigma1*).

N = 1000 # число объектов класса

```
col = len(mu0) # количество столбцов-признаков
```

```
import DataGenerator as dg
```

```
mu = [mu0, mu1]
```

```
sigma = [sigma0, sigma1]
```

```
X, Y, class0, class1 = dg.norm_dataset(mu,sigma,N)
```

Задание для самостоятельной работы

1. Выполните пункты 1-10.
2. Создайте новую функцию внутри файла DataGenerator.py, назовите ее *nonlinear_dataset_N*, где *N* – номер вашего варианта. Данная функция должна генерировать двумерный массив данных, распределенный в пространстве заданным образом. Форма приведена в приложении А. Для ее создания используйте любые математические методы генерации данных. Обеспечьте максимально близкое воспроизведение заданной формы.

Содержание отчета:

1. Титульный лист, описание цели работы. Основные теоретические положения.
2. Полный код программы с пояснениями на каждом из этапов, а также полученные графики.
3. Выводы по работе (объем 0,25-0,5 страницы А4).

Вопросы для самоконтроля:

1. Что представляет собой гистограмма?
2. Изобразите гистограмму нормально распределенных данных. Как по ней можно оценить среднее и СКО?
3. Как можно отобразить распределение двумерных данных? Приведите несколько вариантов.
4. Что значит, что «данные линейно разделимы»?
5. Какие классификаторы называются линейными?
6. С какой целью при разработке классификатора имеющаяся выборка разделяется на подвыборки?
7. Что это за подвыборки и какие варианты деления существуют?

8. Что подразумевается под «обучением классификатора»?
9. Чем классификация отличается от кластеризации?
10. Как можно отразить принадлежность объекта классу через массив меток, если классов больше, чем 2?

Лабораторная работа 2. КЛАССИФИКАТОР НА ОСНОВЕ ЛОГИСТИЧЕСКОЙ РЕГРЕССИИ С ГРАДИЕНТНЫМ СПУСКОМ

Цель: разработка модели классификатора на основе логистической регрессии, изучение его свойств и принципов работы, получение навыков программирования на Python и использования модуля scikit-learn.

Результат работы классификатора \hat{y} – вероятность наступления события для объекта X , где X – вектор-строка $\{x_1, x_2 \dots x_n\}$ в n -мерном пространстве, рассчитывается по формуле сигмоиды:

$$\hat{y} = \frac{1}{1 + e^{-z}}, \quad (2.1)$$
$$Z = b_0 + b_1x_1 + b_2x_2 + \dots b_nx_n$$

Метод логистической регрессии относится к методам обучения с учителем. Под обучением понимается оптимизационный алгоритм, минимизирующий ошибку предсказания. В результате обучения модели формируется набор коэффициентов, который используется для предсказаний на новых данных.

Ошибка предсказания описывается с помощью функции потерь (loss function). Она может определяться разными способами, наиболее часто используемой является следующая функция:

$$loss = -\frac{1}{m} \sum_{i=1}^m \log(\hat{y}_i) * y_i - (1 - y_i) * \log(1 - \hat{y}_i), \quad (2.2)$$

где m – число объектов в выборке, y_i – действительное значение класса i -го объекта (0 или 1), \hat{y}_i – выход сигмоидальной функции (результат предсказания, от 0 до 1).

Метод градиентного спуска – один из методов обучения модели. Более подробно о нем можно узнать в материалах курса.

Чувствительность (истинно положительная пропорция) отражает долю положительных результатов, которые правильно идентифицированы как таковые. Иными словами, чувствительность диагностического теста показывает вероятность того, что больной субъект будет классифицирован именно как больной. *Специфичность* (истинно отрицательная пропорция) отражает долю отрицательных результатов, которые правильно идентифицированы как таковые, то есть вероятность того, что не больные субъекты будут классифицированы именно как не больные.

Библиотека `scikit-learn` – одна из самых популярных библиотек для анализа данных. Она включает сотни функций, покрывающие почти все задачи базового анализа. В частности, в библиотеке реализован метод логрегрессии с возможностью настройки гиперпараметров модели, а также выбора оптимизационного алгоритма.

Для установки `scikit-learn` надо выполнить в командной строке:

```
pip install -U scikit-learn
```

Данная библиотека ориентирована на моделирование данных. Она не ориентирована на их загрузку, манипулирование и суммирование. Для решения этих задач, как правило, применяются библиотеки NumPy и Pandas.

Используемые модули: `numpy`, `scikit-learn`, `matplotlib`

Документация (обязательна к изучению): https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic#sklearn.linear_model.LogisticRegression

Ход работы

1. Создайте новый скрипт. В начале работы импортируйте модули `numpy` и модель логистической регрессии из модуля `scikit-learn` (в данной работе нам понадобится только одна модель).

```
import numpy as np
```



```
from sklearn.linear_model import LogisticRegression
```

2. Используя функцию и скрипты, разработанные в лабораторной работе №1, создайте четыре переменных: *Xtrain*, *Ytrain*; *Xtest*, *Ytest*. Здесь *X* – объекты двух классов, параметры которых распределены нормально, а объекты классов хорошо линейно разделимы. *Y* – метки класса. Для обучения и тестирования модели будут использованы разные выборки.

3. Ознакомьтесь с документацией по *LogisticRegression* для модуля *scikit-learn*.

4. Обучите модель на обучающей выборке используется метод *fit()* (полное описание доступных методов представлено в документации). Чтобы получать воспроизводимые результаты, зафиксируйте стартовое состояние, задав *random_state* равным номеру вашего варианта.

Выберите оптимизационный алгоритм *SAGA* – вариант стохастического градиентного спуска с регуляризацией. Чаще всего именно *SAGA* является наилучшим выбором при решении задачи классификации.

```
Nvar = 5
```

```
clf = LogisticRegression(random_state=Nvar,solver='saga').fit( Xtrain, Ytrain)
```

5. Получите предсказание для новых данных, используя методы *predict()* и *predict_proba()*. Самостоятельно изучите различия между этими методами.

```
Pred_test = clf.predict(Xtest)
```

```
Pred_test_proba = clf.predict_proba(Xtest)
```

6. Оцените точность полученных предсказаний, используя метод *score()*.

```
acc_train = clf.score(Xtrain, Ytrain)
```

```
acc_test = clf.score(Xtest, Ytest)
```

7. Точность также можно легко посчитать вручную.

```
acc_test = sum(Pred_test==Ytest)/len(Ytest)
```

8. Визуализируйте результаты классификации для обучающей и тестовой выборки, отобразив на гистограмме распределения вероятности, полученной на выходе из классификатора (рис. 2.1). Разные классы отобразите разными цветами. Проанализируйте полученные графики.

```
import matplotlib.pyplot as plt
plt.hist(Pred_test_proba[Ytest,1], bins='auto', alpha=0.7)
plt.hist(Pred_test_proba[~Ytest,1], bins='auto', alpha=0.7) # т.к массив с
вероятностями имеет два столбца, мы берем один - первый
plt.title("Результаты классификации, тест")
plt.show()
```

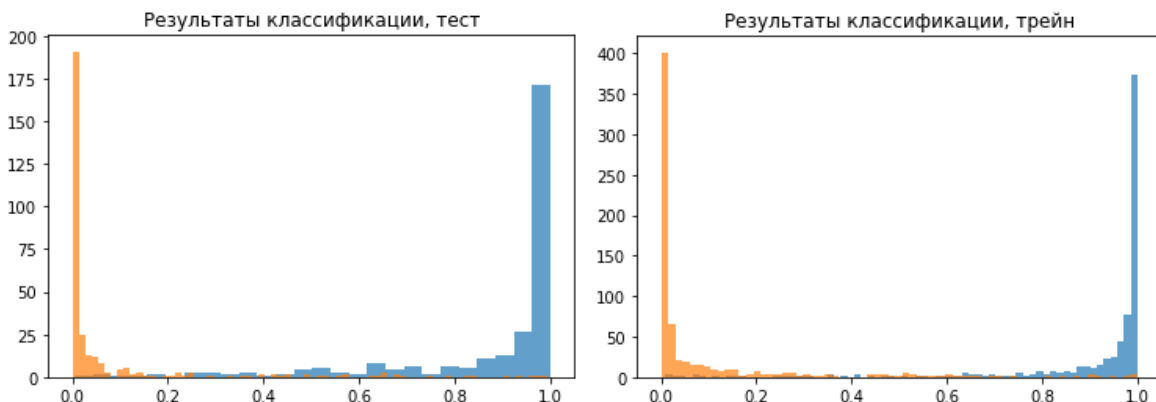


Рисунок 2.1 – Вероятности принадлежности объектов 1 классу для обучающей и тестовой выборки.

Задание для самостоятельной работы

1. Выполните пункты 1-8.
2. Считая, что класс 0 – это отсутствие признака, а класс 1 – наличие признака, самостоятельно рассчитайте чувствительность и специфичность алгоритма, используя известные вам формулы без использования методов библиотеки.
3. Сведите полученные результаты в таблицу:

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Train				
Test				

4. Изменяя параметры генерируемых данных (средние и СКО), добейтесь более плотного пересечения классов – в этом случае параметры эффективности классификатора ухудшатся.

5. Оцените эффективность классификатора на нелинейно-пересекаемых классах. Их можно сгенерировать, используя функцию из лаб. работы №1.

Содержание отчета:

1. Титульный лист, описание цели работы. Основные теоретические положения.
2. Полный код программы с пояснениями на каждом из этапов.
3. Гистограммы распределения вероятности на выходе классификатора и таблицы с данными об эффективности для 3-х выборок (нормальное распределение: хорошо и плохо разделимые (выборки А и Б), а также нелинейно-пересекаемые (выборка В)).
4. Таблицы с результатами оценки качества классификации (точность, чувствительность, специфичность).
5. Выводы по работе.

Вопросы для самоконтроля:

1. Что представляет из себя модель логистической регрессии?
2. Изобразите график сигмоиды.
3. Как рассчитывается выход классификатора? Напишите формулу и правило определения принадлежности объекта классу 0 или 1.
4. Какая связь между логистической и линейной регрессией?
5. В каких случаях использование лог. регрессии будет эффективно?
6. В чем заключается обучение модели лог. регрессии?
7. Опишите принцип градиентного спуска. Есть ли еще методы обучения, применимые к логистической регрессии?

8. Что такое регуляризация? Для чего она используется?
9. Почему в формуле потерь используется два слагаемых, а также логарифм?
10. Напишите формулы для расчета точности, чувствительности и специфичности классификатора.

Лабораторная работа 3. ДЕРЕВЬЯ И ЛЕСА РЕШЕНИЙ

Цель: реализация классификатора на основе дерева принятия решений и исследование его свойств.

Дерево решений (распознающее дерево, recognition или decision tree) – распознаватели вида, при котором для распознаваемого объекта проводится конечная последовательность сравнений значений его признаков с константами на равенство или неравенство, причем от результатов каждого сравнения зависят дальнейшие действия – продолжать сравнивать с чем-то еще или давать ответ распознавания. То есть распознавание можно представить как двоичное дерево вложенных операторов вида:

```
if x[j] ?? d[k] then ...  
    else...
```

завершающееся листьями:

```
return res[h]
```

где x -массив признаков, d -массив пороговых значений, res -массив возможных ответов, знаком $??$ обозначена операция сравнения.

Деревья строятся при помощи обучения с учителем. В качестве обучающего набора данных используется множество наблюдений, для которых предварительно задана метка класса.

Структурно дерево решений состоит из объектов двух типов — узлов (node) и листьев (leaf). В узлах расположены решающие правила и подмножества наблюдений, которые им удовлетворяют. В листьях содержатся классифицированные деревом наблюдения: каждый лист ассоциируется с одним из классов, и объекту, который распределяется в лист, присваивается соответствующая метка класса.

Под обучением дерева понимается определение его структуры, операций сравнения, пороговых величин и сравниваемых на каждом узле признаков, а также ответов в каждом листе. По своей сути, дерево может быть описано как

набор операций «разрезания» признакового пространства гиперплоскостями, проходящими через пороговые значения признаков (рис. 3.1). Именно поэтому **дерево решений является линейным классификатором**, несмотря на его достаточно сложную организацию. Стоит отметить, что в общем случае деревья применимы и для решения задач регрессии.

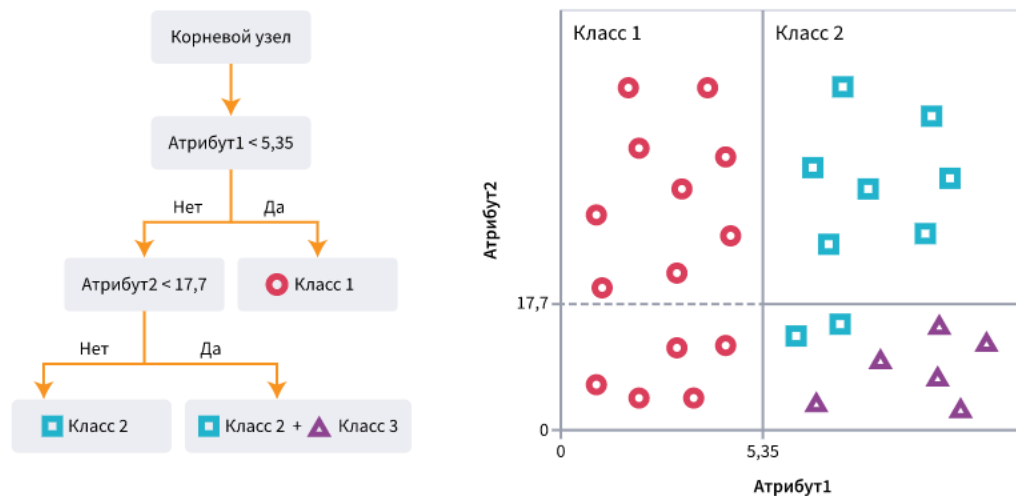


Рисунок 3.1 – Пример дерева решений

Алгоритмы построения деревьев решений относят к категории жадных алгоритмов. Алгоритм считается *жадным*, если допускает, что локально-оптимальные решения на каждом шаге (разбиения в узлах), приводят к оптимальному итоговому решению. В случае деревьев решений это означает, что если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать другой атрибут, который дал бы лучшее итоговое разбиение. Поэтому на этапе построения нельзя сказать обеспечит ли выбранный атрибут, в конечном итоге, оптимальное разбиение. Более подробно про алгоритмы обучения – в материалах курса, а также в дополнительных материалах.

Одно дерево не всегда может эффективно справиться с задачей классификации или регрессии. В таком случае возможным выходом является использование ансамблей. **Ансамбль** – это некоторая совокупность алгоритмов, объединенных в единое целое. Каждый алгоритм имеет свою вероятность ошибки, и объединяя выходы тысячи среднеточных моделей

можно добиться более точного сведенного результата «голосования», усреднив результаты.

Бэггинг (от Bagging - Bootstrap aggregation) — это один из первых и самых простых видов ансамблей. Он был придуман Лео Брэйманом в 1994 году. **Бэггинг** основан на статистическом методе бутстрэпа, который позволяет оценивать многие статистики сложных распределений, когда выборка дробится на множество подвыборок и на них оценивается бутстрэп статистика. Бэггинг позволяет снизить дисперсию обучаемого классификатора, уменьшая величину, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных, или другими словами, предотвращает переобучение. Эффективность бэггинга достигается благодаря тому, что базовые алгоритмы, обученные по различным подвыборкам, получают достаточно различными, и их ошибки взаимно компенсируются при голосовании, а также за счёт того, что объекты-выбросы могут не попадать в некоторые обучающие подвыборки.

Случайный лес (random forest) — бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков. Для задачи классификации итоговое решение выбирается по большинству результатов, выданных классификаторами, а в задаче регрессии — по их среднему значению.

Одной из главных проблем случайного леса является его склонность к переобучению и «рваным краям» разделяющих поверхностей. Алгоритмы построения и обучения лесов постоянно совершенствуются и дополняются.

Используемые библиотеки: numpy, scikit-learn, scikit-plot, matplotlib

Документация (обязательна к изучению): <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
<https://scikit-plot.readthedocs.io/en/stable/metrics.html>

Дополнительные материалы:

- 1) <https://tproger.ru/translations/python-random-forest-implementation/> - разработка деревьев решений на Python с пояснениями по теории
- 2) <https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-python-using-scikit-learn-38ad2d75f21c> - визуализация деревьев решений

- 3) <https://habr.com/ru/post/171759/> - энтропия как параметр для оптимизации в деревьях решений
- 4) <https://loginom.ru/blog/decision-tree-p1> - теория по деревьям решений, в том числе затронуты вопросы обучения деревьев
- 5) <https://habr.com/ru/company/ods/blog/324402/> - бэггинг и случайный лес – теория + практические примеры

Ход работы

1. Повторите пункты 1-3 из Лабораторной работы №2, но вместо логистической регрессии загрузив дерево решений (DecisionTreeClassifier)

```
from sklearn.tree import DecisionTreeClassifier
```

Создайте два массива данных, относящихся к двум классам со средней степенью пересечения. Разделите выборку на обучающую и тестовую.

2. Обучите модель, по аналогии с логистической регрессией, используя метод *fit()*, на обучающей выборке *Xtrain*, *Ytrain*. Задайте *random_state=0* для получения воспроизводимых результатов обучения.

3. Оцените точность, чувствительность и специфичность обученной модели на обучающей и тестовой выборках. Обратите внимание, что точность модели на обучающих данных равна (или почти равна) 1, а на тестовых она ощутимо ниже. Проанализируйте, в связи с чем наблюдается такой эффект.

4. Выполните пункты 1-3 для классификатора RandomForestClassifier. Сравните полученные результаты для леса и дерева.

5. Постройте ROC-кривые для дерева и для леса, используя результаты классификации тестового набора данных. Рассчитайте площадь под кривой. Для этого используйте библиотеку scikit-plot. После ее установки через *pip* импортируйте ее в рабочий скрипт и используя модуль *metrics* постройте ROC-кривую. Расчет площади под кривой удобнее производить, используя модуль *sklearn.metrics*.

```
import scikitplot as skplt  
import matplotlib.pyplot as plt
```



```
skplt.metrics.plot_roc_curve(Ytest, Pred_test_proba, figsize = (10,10))
plt.show()
```

```
from sklearn.metrics import roc_auc_score
```

```
# Расчет площади под кривой
```

```
AUC = roc_auc_score(Ytest, Pred_test_proba[:,1])
```

```
print("AUC tree:"+str(AUC))
```

6. Постройте гистограммы распределения результатов классификации (используя вероятности) для случайного леса (для обучающей и тестовой выборок). Проанализируйте результаты.

Задание для самостоятельной работы

1. Выполните пункты 1-6.
2. Считая, что класс 0 – это отсутствие признака, а класс 1 – наличие признака, самостоятельно рассчитайте чувствительность и специфичность алгоритмов, используя известные вам формулы.
3. Сведите полученные результаты в таблицу:

	Число объектов	Точность, %	Чувствительность, %	Специфичность, %
Train				
Test				

4. Оцените эффективность решающих деревьев на нелинейно-пересекаемых классах. Их можно сгенерировать, используя функцию из лаб. работы №1.

5. Изучив документацию классификатора дерева решений, попробуйте подобрать такие гиперпараметры (в частности, глубину дерева), которые снизят переобучение модели.

6. Используя цикл *for* и перебор, подберите такое значение количества деревьев в лесе (в диапазоне от 1 до 300 с шагом в 10), которое даст наибольшее значение площади под кривой на тестовой выборке.

Содержание отчета:

1. Титульный лист, описание цели работы. Основные теоретические положения.
2. Полный код программы с пояснениями на каждом из этапов.
3. Гистограммы распределения вероятности на выходе классификатора и таблицы с данными об эффективности для 3х выборок (нормальное распределение: хорошо и плохо разделимы, а также нелинейно-пересекаемые).
4. Таблицы с результатами оценки качества классификации (точность, чувствительность, специфичность).
5. Результаты подбора наилучших гиперпараметров моделей.
6. Выводы по работе.

Вопросы для самоконтроля:

1. Дерево принятия решений – это линейный или нелинейный классификатор?
2. Каким образом в двумерном пространстве признаков можно визуализировать разделяющую плоскость дерева, которая формируется набором решающих правил?
3. Какие задачи могут решать деревья помимо классификации?
4. Из каких элементов состоит дерево принятия решений?
5. Какие недостатки имеют деревья принятия решений и каким образом их можно сгладить?
6. В чем преимущество использования ансамблей, в частности случайного леса?
7. Что такое бэггинг?
8. Что такое гиперпараметры модели?
9. Как выбрать наилучшую модель для имеющихся данных?

Лабораторная работа 4. НЕЙРОННЫЕ СЕТИ

Цель: создание простейшей нейронной сети на Python без использования специализированных библиотек.

Простейшая нейронная сеть состоит из слоя искусственных нейронов (ИН). ИН по своей сути представляет из себя алгоритм вычисления выхода из входных данных по известной формуле, с использованием набора весовых коэффициентов w и смещений b .

На вход нейрона поступают сигналы x_i , каждый умножается на соответствующий коэффициент w_i и суммируется (рис. 4.1). Полученная взвешенная сумма поступает на функцию активации, результат вычисления которой и является выходом нейрона.

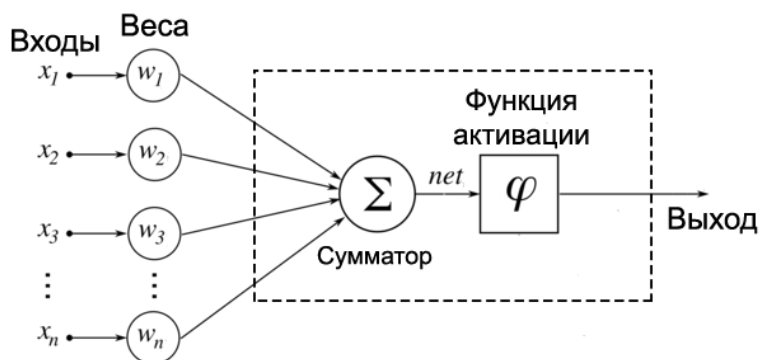


Рисунок 4.1 – Структура искусственного нейрона.

Простейшей функцией активации является функция единичного скачка, принимающая только два значения – 0, если net ниже порогового значения, 1 если превышает. Самой распространенной функцией активации является сигмоида, с которой вы уже знакомы. На выходе нейрон отдает значение от 0 до 1. Существует множество других видов функций активации, с которыми вы познакомитесь в рамках курса.

Нейронные сети состоят из набора нейронов, сгруппированных по слоям. В однослойных нейронных сетях сигналы с входного слоя сразу подаются на выходной слой (рис. 4.2). Он производит необходимые вычисления, результаты которых сразу подаются на выходы.

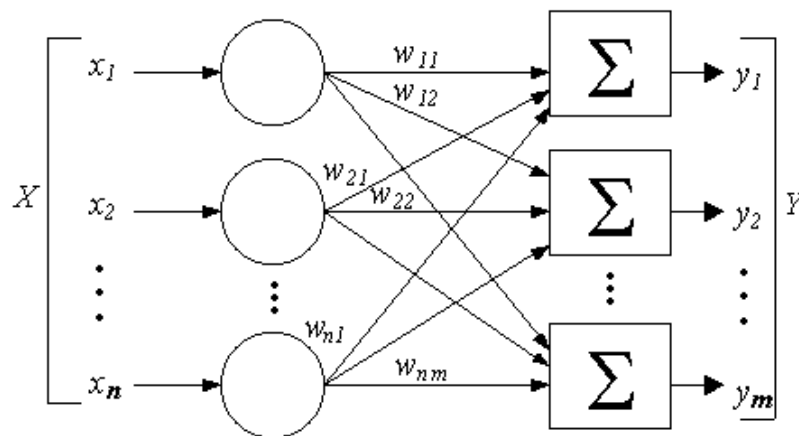


Рисунок 4.2 – Однослойная нейронная сеть.

Входы не считаются за входной слой сети и обозначены кружками. Справа (квадраты) расположены нейроны основного слоя.

Под обучением нейронной сети понимается поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется с минимальной ошибкой в нужный нам выходной.

Используемые библиотеки: numpy, matplotlib

Дополнительные материалы:

1. <https://neuralnet.info/chapter/персептроны> - история развития персептронов и основные понятия
2. <https://neuralnet.info/chapter/основы-инс-> простым языком про ИНС

Ход работы

1. Создайте новый скрипт. Импортируйте numpy.
2. Для начала создадим ряд функций для основных вычислений, происходящих внутри сети. Активационную функцию используем сигмоиду.

```
#Активационная функция сигмоиды
def sigmoid(Z):
    return 1/(1+np.exp(-Z))
```

3. Для расчетов также потребуется производная сигмоиды

```
def sigmoid_derivative(p):
    return p * (1 - p)
```

4. Сама нейронная сеть будет представлять из себя класс с набором методов – инициализация (*init*), прямое распространение (*feedforward*), обратное распространение (*backprop*) и обучение (*train*).

```
class NeuralNetwork:
```

```
    def __init__(self, x,y):
        self.input = x
        n_inp = self.input.shape[1] # кол-во входов
        n_neuro = 4 # число нейронов на главном слое
        # инициализация весов рандомными значениями
        self.weights1= np.random.rand(n_inp,n_neuro)
        self.weights2 = np.random.rand(n_neuro,1)
        self.y = y
        self.output = np. zeros(y.shape)
```

```
    def feedforward(self):
```

```
        #ВЫХОДЫ СЛОЁВ ВЫЧИСЛЯЮТСЯ ПО СИГМОИДЕ
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2
```

```
    def backprop(self):
```

#здесь происходит коррекция весов по известному вам из курса алгоритму.

Подумайте, что значит .T после NN.weights2

```
        d_weights2 = np.dot(self.layer1.T, 2*(self.y -
self.output)*sigmoid_derivative(self.output))
        d_weights1 = np.dot(self.input.T, np.dot(2*(self.y -
self.output)*sigmoid_derivative(self.output),
self.weights2.T)*sigmoid_derivative(self.layer1))
```

```
#обновляем веса
self.weights1 += d_weights1
self.weights2 += d_weights2
```

```
def train(self, X, y):
#весь процесс обучения прост – высчитываем выход с помощью прямого
распространения, а после обновляем веса
self.output = self.feedforward()
self.backprop()
```

5. Чтобы запустить созданную сеть сгенерируйте выборку, используя ранее написанный генератор данных. Единственное, надо будет изменить размер массива меток Y и из плоского сделать его одномерным. Т.к. в этой работе делается самый простой вариант НС, делить выборку на обучение и тест не будем.

```
mu0 = [0, 2, 3] # параметры выборки даны для примера, задайте свои и
можно выбрать более 3х столбцов
```

```
mu1 = [3, 5,1]
sigma0= [2, 1, 2]
sigma1 = [1, 2, 1]
```

```
N = 1000 # число объектов класса
col = len(mu0) # количество столбцов-признаков
```

```
import datagenerator as dg
mu = [mu0, mu1]
sigma = [sigma0, sigma1]
X, Y,class0, class1 = dg.norm_dataset(mu,sigma,N)
Y = np.reshape(y,[2000,1])
```

6. В цикле обучите сеть на достаточном количестве эпох

```

NN = NeuralNetwork(X,Y) # инициализируем сетку на наших данных
N_epoch = 50
for i in range(N_epoch):
    print ("for iteration # " + str(i) + "\n")
    print ("Loss: \n" + str(np.mean(np.square(y - NN.feedforward())))) #
потери рассчитайте как среднеквадратичные
    NN.train(X, Y) #собственно, обучение сети

```

7. Для получения предсказания необходимо вызвать метод `feedforward`. Заметьте, что он выдает не номер класса, а значение от 0 до 1

```
pred = NN.feedforward()
```

Задание для самостоятельной работы

1. Выполните пункты 1-7.
2. Рассчитайте итоговую точность полученной НС.
 *(доп. задание «со звездочкой») Конечно, оценивать точность на обучающей выборке – плохая практика, так что вы можете доработать класс NN и включить в него метод *test* для получения предсказания на тестовых данных.
3. Вычисляя на каждой эпохе обучения значение точности и среднеквадратичной потери, постройте графики их зависимости от номера эпохи.
4. Проанализировав графики, подберите оптимальное число нейронов и эпох обучения.
5. Вынесите в отдельную переменную значения весов нейронов на каждом из слоев.

Содержание отчета:

1. Титульный лист, описание цели работы. Основные теоретические положения.
2. Полный код программы с пояснениями на каждом из этапов.
3. Структура используемой нейронной сети в виде блок-схемы.
4. Значения весов НС на каждом из слоев.

5. Результаты обучения: графики функции потерь и изменения точности, а также итоговую точность модели.
6. Выводы по работе.

Вопросы для самоконтроля:

1. Что такое искусственный нейрон?
2. Что такое нейронная сеть?
3. Как рассчитывается взвешенная сумма?
4. Как рассчитать выход с перцептрона с функцией активации сигмойдой при известных весах и входах?
5. Для чего визуализировать графики изменения функции потерь и точности, как их анализировать?
6. Как обучается стандартная НС без скрытых слоев?
7. Как получить предсказание предобученной НС на новых данных?
8. Какие параметры и характеристики модели необходимо экспортировать, чтобы после обучения использовать ее в другом приложении?
9. Какие значения могут быть получены на выходе из НС?

ПРИЛОЖЕНИЕ А

Варианты модельных распределений к лаб.работе №1

