



Специальные технологии баз данных и информационных систем

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН, Д.Ю. КУПРИЯНОВ. ЛЕКЦИЯ 5. СЕМЕСТР 2

Библиотеки

В данной лекции будут рассмотрены примеры с использованием следующих библиотек:

- NumPy – <https://numpy.org/>
- Pandas – <https://pandas.pydata.org/>
- Scikit-learn – <https://scikit-learn.org>
- Matplotlib – <https://matplotlib.org/>
- Keras - <https://keras.io/>

Часть 1

ЧТО ТАКОЕ НЕЙРОСЕТЬ

Как устроен человеческий мозг?

Человеческий мозг — это сеть нервных клеток. Нервные клетки принято называть нейронами. У нейрона есть тело, называемое сомой, внутри которого располагается ядро. Из сомы нейрона выходят отростки двух видов: многочисленные тонкие, густо ветвящиеся дендриты и более толстый, расщепляющийся на многочисленные нервные окончания — коллатералы, аксон.

Аксон — длинный отросток нейрона.

Дендриты — короткие и сильно разветвлённые отростки нейрона

Нейрон может иметь несколько дендритов и обычно только один аксон.

Один нейрон может иметь связи со многими (до 20 тысяч) другими нейронами и эффекторными клетками.

Синапс — место контакта между двумя нейронами или между нейроном и получающей сигнал эффекторной клеткой.

Структура нейрона

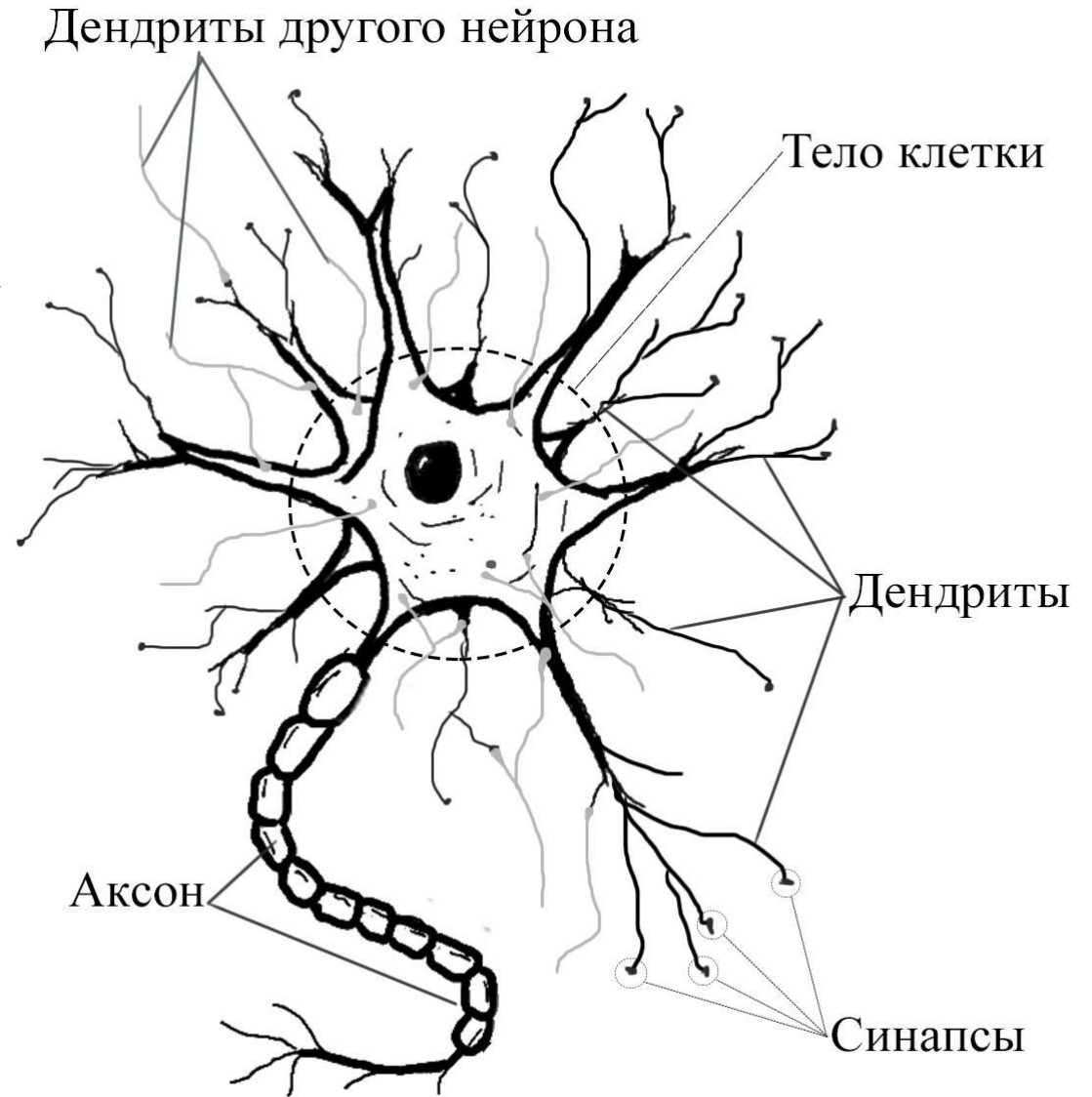


Рисунок взят с ресурса <https://www.pvsm.ru/programmirovanie/221453>

Как передается сигнал в мозге?

Все нейроны соединены со множеством других нейронов и могут передавать друг другу и эффекторным клеткам сигналы.

Нейрон реагирует на воздействие. Это может быть как поступление сигнал от других клеток (через синапсы на дендритах или теле клетки), так и в следствии давления, растяжения и т.д.

Выходной сигнал нейрона передается через аксон при помощи коллатералов. Коллатералы контактируют с телами клеток (сомой) и дендритами других нейронов. Места соединения коллатер с другими нейронами – это синапсы. Таким образом, коллатеры формируют выходные сигнала клетки.

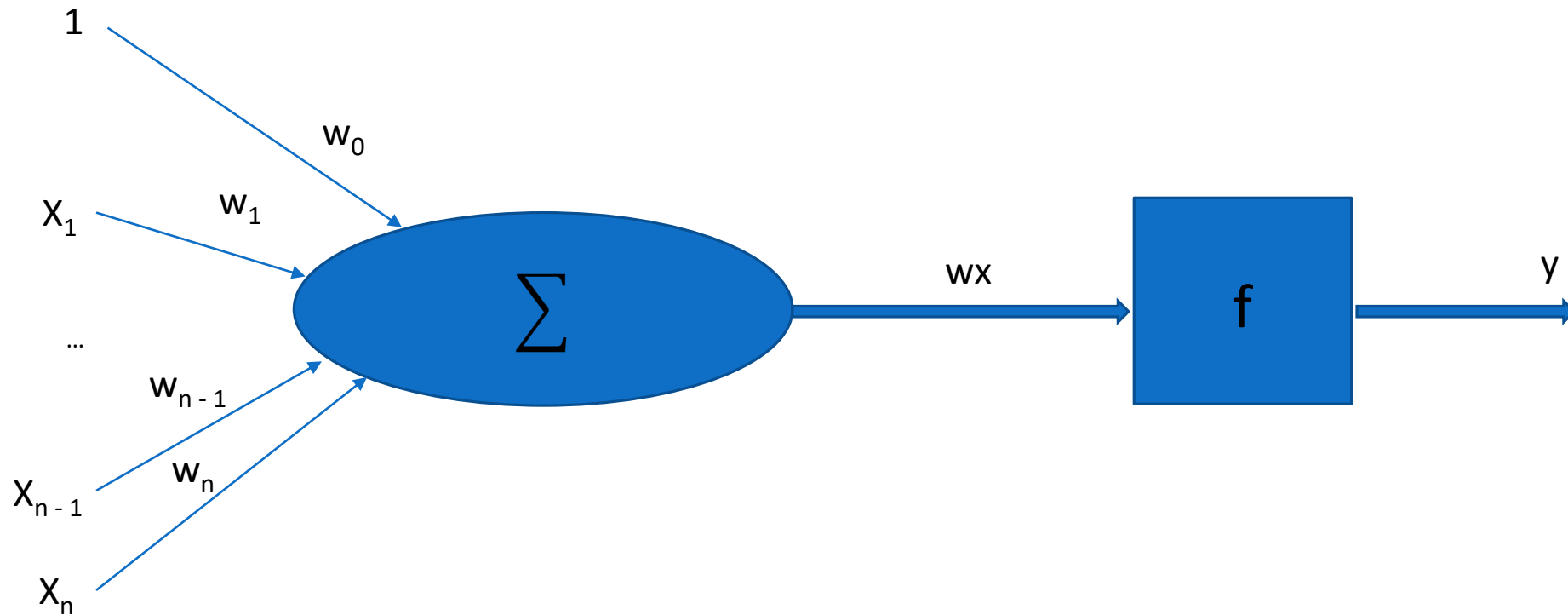
Много ли нейронов в мозге?

Приблизительно число нейронов в мозге человека оценивается, как 10^{11} . При этом они могут отличаться по форме и свойствам. В каждый нейрон может входить до 1000 соединений с другими нейронами.

Можно ли повторить человеческий мозг?

Сегодня это невозможно! Но подходы, выявленные в работе человеческого мозга можно применить в меньших масштабах для решения интеллектуальных задач.

Модель МакКаллока-Питса



Модель МакКаллока-Питса

В модели нейрона МакКаллока-Питса (1943 год) есть входной вектор признаков:

$$X(t) = (x_1(t), x_2(t), \dots x_n(t)).$$

0-й сигнал всегда равен 1. t означает момент времени. В каждый отдельный момент времени подаётся новый обучающий вектор признаков (новое наблюдение)

Вектор весов $W = (w_0, w_1, \dots w_n)$ – это основной механизм нейрона. Данные значения будут меняться от одного момента времени к другому и реализовывать обучение.

Выходной сигнал нейрона определяется взвешенной суммой:

$$y = f(w_0 + \sum_{i=1}^n w_i x_i(t)),$$

где f – функция активации.

Модель МакКаллока-Питса

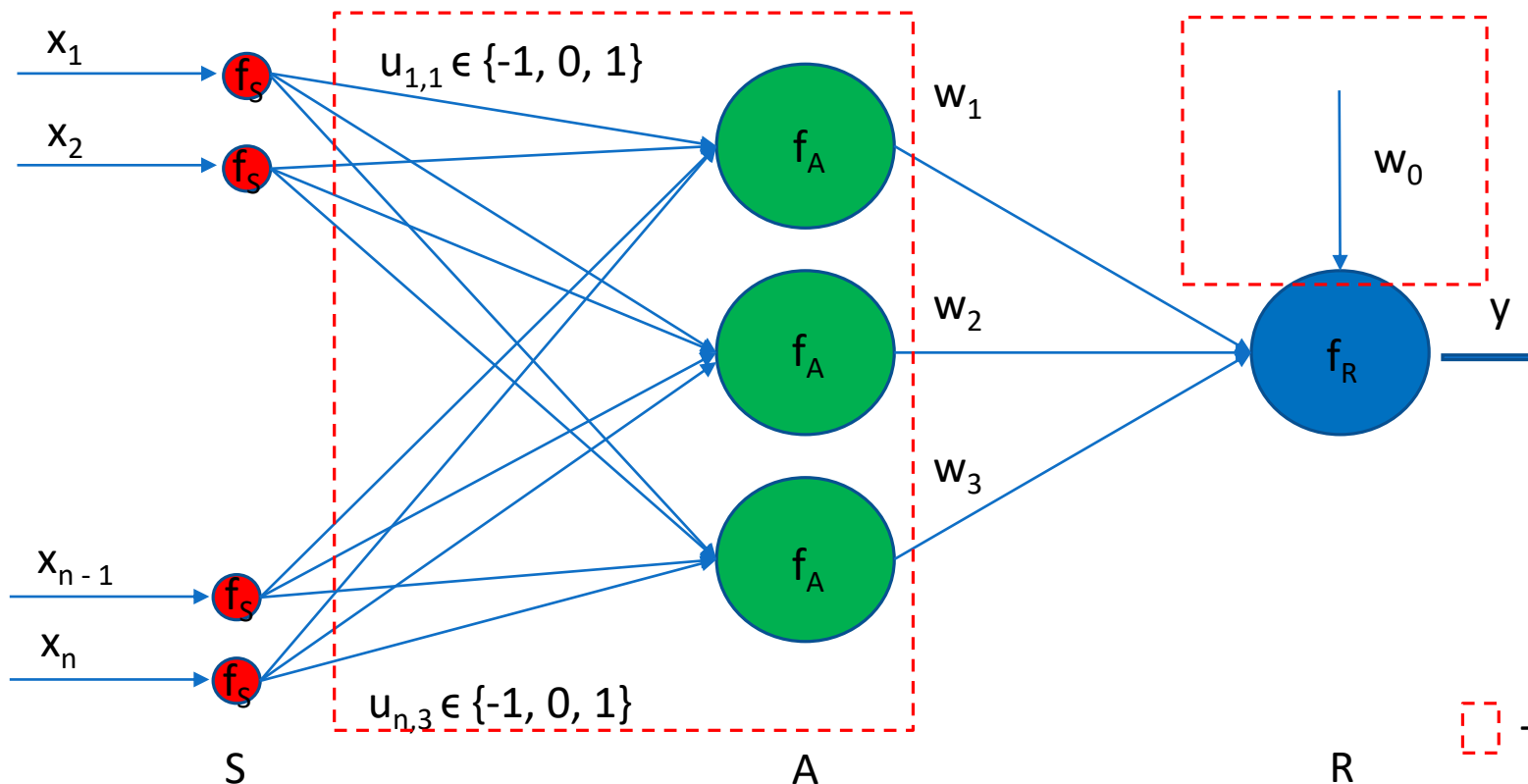
В модели МакКаллока-Питса состояние нейрона (вектор весов) изменяется дискретно, переходя от состояния в момент времени t к состоянию в момент времени $t + 1$.

Функция активации в модели МакКаллока-Питса пороговая:

$$f(u) = \begin{cases} 1 & \text{для } u > 0 \\ 0 & \text{для } u \leq 0 \end{cases}$$

Персептрон

В 1958 году Ф. Розенблатт ввел понятие персептрона – простейшей нейронной сети, обучаемой с учителем.



$$f_S(x_i) = \begin{cases} 1 & \text{для } x_i > P_S \\ 0 & \text{для } x_i \leq P_S \end{cases}$$

$$f_A(s_j) = \begin{cases} 1 & \text{для } s_j > P_A \\ 0 & \text{для } s_j \leq P_A \end{cases}$$

$$s_j = \sum_{i=1}^n f_S(x_i) u_{i,j}$$

$$f_R(S) = \begin{cases} 1 & \text{для } S > 0 \\ -1 & \text{для } S \leq 0 \end{cases}$$

$$S = w_0 + \sum_{j=1}^3 f_A(s_j) w_j$$

 – может отсутствовать

Персептрон

На каждом шаге времени в персептроне меняется только вектор весов W .

$$w_j(t + 1) = w_j(t) + \Delta w_j,$$

где

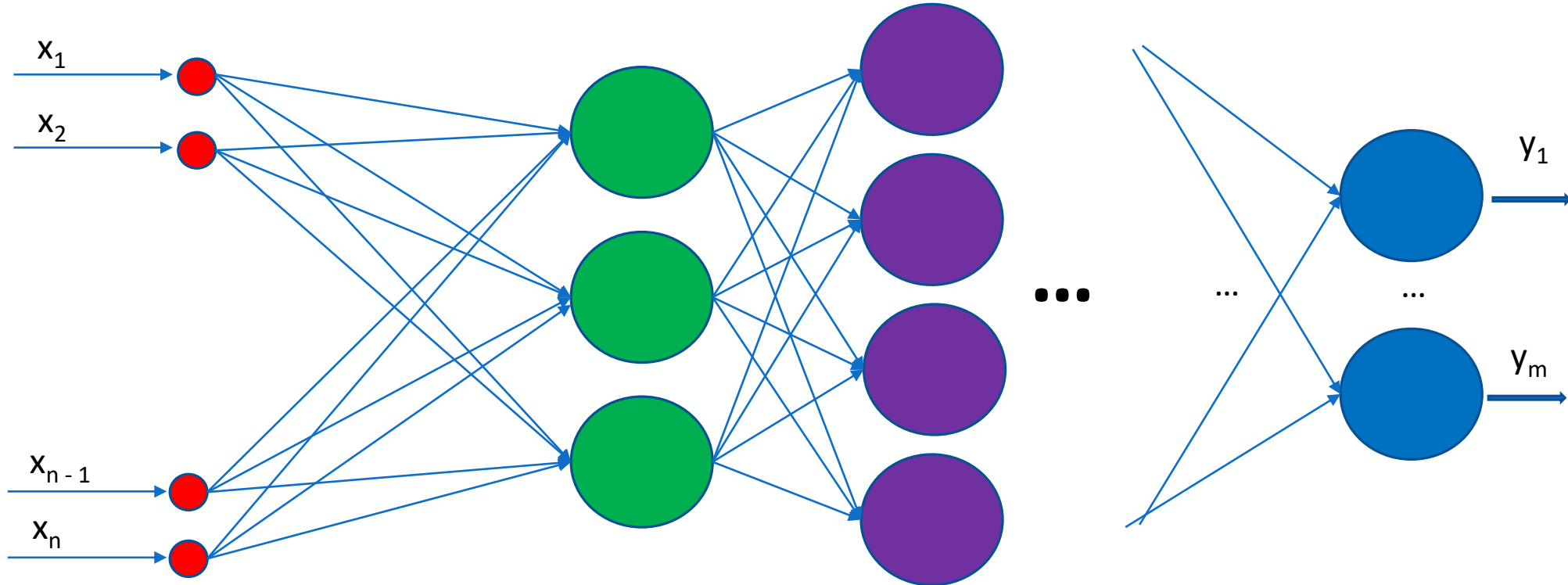
$$\Delta w_j = f_A(s_j(t)) (d(t) - y(t)),$$

где $f_A(s_j(t))$ – значение функции активации для j -го узла слоя A для входного набора в момент времени t , $d(t)$ – правильный ответ, ожидаемый для входного набора в момент времени t , $y(t)$ – полученный ответ в момент времени t . Фактически, $d(t) - y(t)$ – это функция потерь.

Для того, чтобы персептрон работал Δw_j должно уменьшаться со временем и стремиться к минимуму. Но при таком простейшем подходе это вовсе не обеспечено! Поэтому, персептрон в таком виде может и не работать!

Многослойный персептрон

Можно взять много слоев и получить многослойный персептрон. А много выходных нейронов дадут возможность получать ответ-вектор.



Часть 2

ПЕРСЕПТРОН И КЛАССИФИКАЦИЯ

Многослойный персептрон, классификация и Scikit-Learn

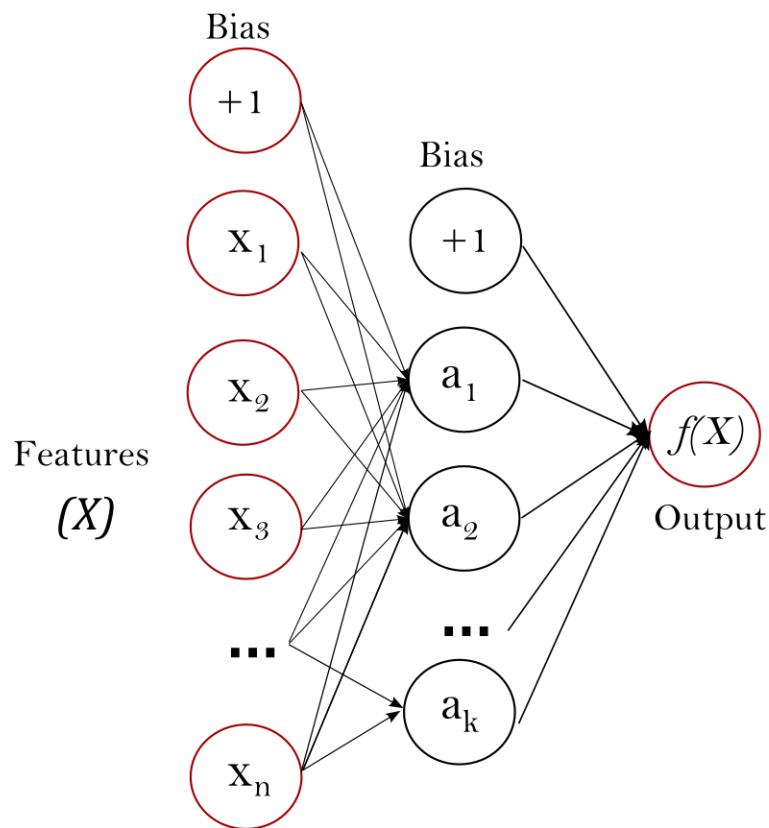
Многослойные персептроны успешно применяются для классификации наблюдений. Это алгоритмы, основанные на обучении с учителем.

В библиотеке Scikit-Learn есть готовая реализация многослойного персептрона (MLP) с различными алгоритмами пересчета весов. Данные алгоритмы, в отличие от простого персептрона, обеспечивают минимизацию функции Δw_j , что позволяет гарантировать их сходимость.

Структура MLP в Scikit-Learn

На рисунке показан MLP с одним скрытым слоем. Scikit-learn позволяет создавать много скрытых слоев, указав количество узлов в них с помощью параметра-списка **hidden_layer_sizes**.

2 слоя есть всегда (выделены красным). Их размерность указывать не надо.



Изображение взято с ресурса

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

MLP в Python

Для применения метода MLP в Python используется класс `MLPClassifier` модуля `neural_network` библиотеки Scikit-Learn [2].

Данная реализация предлагает на выбор три алгоритма минимизации функции Δw_j [3]:

- стохастический градиентный спуск (*Stochastic gradient descent, SGD*);
- метод адаптивной оценки моментов (Adam);
- алгоритм Бройдена — Флетчера — Гольдфарба — Шанно с ограниченным использованием памяти (L-BFGS or LM-BFGS).

Большее число алгоритмов можно найти в библиотеке Keras [5].

Эти алгоритмы довольно нетривиальны. Мы не будем разбирать их подробно, а поговорим только о параметрах их настройки.

SGD

Стохастический градиентный спуск включается указанием параметра

```
solver = 'sgd'
```

Пересчёт весов осуществляется по формуле:

$$w(t + 1) = w(t) - \eta \nabla Q(w(t)),$$

где η – параметр скорости обучения (learning rate), Q – функция потерь.

Параметр η может быть постоянным, уменьшающимся или адаптивным. В первом случае он не меняется, во втором уменьшается со временем обучения, в третьем подстраивается под особенности обучения (может расти или уменьшаться). Для выбора поведения скорости обучения используется параметр **learning_rate**. Возможные значения: 'constant', 'invscaling', 'adaptive'.

Если скорость обучения постоянна, то её величину можно задать параметром **learning_rate_init**. По умолчанию значение равно величине 0,001.

Если используется уменьшающаяся скорость обучения, то можно управлять показателем степени данного уменьшения (показатель степени экспоненты). Для этого доступен параметр **power_t**.

SGD

Для предотвращения переобучения доступен параметр регуляризации α . Он задаётся через одноимённый параметр **alpha**. Значение по умолчанию: 0.0001. Фактически, это параметр L2 регуляризации.

L2 регуляризация добавляет к функции потерь некоторое значение $\alpha \sum w_j^2$.

Большие значения α приведут к значительному уменьшению весов, что может увеличить смещение модели, в то время как слишком маленькие значения могут оставить модель подверженной переобучению. Путём уменьшения величины весов, L2 регуляризация помогает снизить дисперсию модели, что делает её менее чувствительной к отдельным точкам данных, и таким образом уменьшает риск переобучения.

L2 регуляризация уменьшает веса постепенно, делая модель более «гладкой» и менее подверженной влиянию шума в данных. L2 регуляризация полезна в ситуациях, когда количество признаков в данных велико или когда они сильно коррелированы. Она помогает справляться с мультиколлинеарностью, сохраняя все признаки, но уменьшая их влияние.

SGD

Одним из известных недостатков метода градиентного спуска является проблема выбора локального минимума в качестве оптимальной точки в случае невыпуклой целевой функции. Данная проблема является довольно распространенной в задачах машинного обучения, т. к. зачастую функция ошибок является многоэкстремальной, что приводит к ошибочным результатам в результате работы классического или стохастического градиентного спуска.

Проблему решают импульсные методы за счет накопления значений предыдущих градиентов. В Scikit-learn реализованы метод моментов и метод Нестерова. Они управляются параметрами:

momentum = число от 0 до 1 (по умолчанию – 0,9)

и

nesterovs_momentum = True или False (только для **momentum** > 0, по умолчанию True)

Число эпох

В алгоритмах, подобных SGD эпоха – это одно использование всех наблюдений для обучения. Одни и те же данные можно использовать для обучения несколько раз. Таким образом, мы получаем несколько эпох.

Параметр **max_iter** задает максимальное число эпох.

Подходы к обучению

Пакетный подход (full-batch) – параметры сети (веса и градиенты) обновляются после целой эпохи. Иногда его называют offline.

Мини-пакетный подход (mini-batch) – пакеты меньше всей выборки по размеру, но больше 1. Обычно размер подбирают из кратности общего размера или по степеням двойки (16, 32, 64 и т.д.). Параметры сети обновляются после каждого пакета.

Стохастический – единичный подход (online) – параметры сети обновляются после каждой записи.

SGD в sklearn умеет работать в online или mini-batch режимах. Переключается параметром **batch_size**. По умолчанию он «auto», но можно указать целое число.

Пример

Выборка 200 записей.

Пусть пакет 50 записей.

Число эпох пусть у нас 500.

Тогда для каждой эпохи сеть обновится 4 раза.

Всего параметры сети будут пересчитаны 2000 раз.

Adam

Adam похож на SGD в том смысле, что он является стохастическим оптимизатором, но он может автоматически регулировать количество обновляемых параметров на основе адаптивных оценок моментов низшего порядка.

Adam

Метод адаптивной оценки моментов (Adam) допускает часть таких же настроечных параметров. В том числе, допустимы:

learning_rate_init

alpha

Кроме того, он подразумевает ряд своих параметров, определяемых формулой алгоритма:

beta_1, beta_2 принимают значения от 0 до 1 не включая саму 1.

Также допустим параметр численной устойчивости **epsilon** (маленькое положительное число).

Аналогично SGD, Adam в sklearn умеет работать в online или mini-batch режимах.

L-BFGS

Алгоритм Бroyдена — Флетчера — Гольдфарба — Шанно с ограниченным использованием памяти (L-BFGS or LM-BFGS) допускает только один параметр, не относящийся к параметрам остановки. Это параметр регуляризации **alpha**.

L-BFGS — это алгоритм, аппроксимирующий матрицу Гессе, представляющую частную производную функции второго порядка. Далее он аппроксимирует обратную матрицу Гессе для обновления параметров сети.

Sklearn использует алгоритм из библиотеки `scipy`.

L-BFGS использует только full-batch подход.

Недостатки MLP

1. Функция минимизации ошибки может иметь несколько минимумов. Есть вероятность схождения к неверному результату. Для решения данной проблемы запускают MLP несколько раз с разными начальными значениями.
2. Множество настроечных параметров, которые надо подбирать итеративно.
3. Высокая чувствительность к размерности параметров. Для решения проблемы применяем стандартизацию.

Задача 1

Решим классическую задачу кластеризации. Кластеризацию ирисов Фишера [1]. Ирисы Фишера – это набор данных, собранных американским ботаником Эдгаром Андерсоном. Каждая запись данного набора состоит из длины наружной доли околоцветника или чашелистника (англ. sepal length), ширины наружной доли околоцветника или чашелистника (англ. sepal width), длины внутренней доли околоцветника или лепестка (англ. petal length), ширина внутренней доли околоцветника или лепестка (англ. petal width) и указания вида ириса (класса). Всего рассмотрено три вида ирисов: setosa, versicolor, и virginica. Первый из них линейно отделим от других.

На данной задаче часто проверяют качество методов кластеризации, сравнивая полученные результаты с реальным делением на классы (по видам ирисов).

Ирисы Фишера на Википедии

Ирисы Фишера

Длина чашелистика ↕	Ширина чашелистика ↕	Длина лепестка ↕	Ширина лепестка ↕	Вид ириса ↕
5.1	3.5	1.4	0.2	<i>setosa</i>
4.9	3.0	1.4	0.2	<i>setosa</i>
4.7	3.2	1.3	0.2	<i>setosa</i>
4.6	3.1	1.5	0.2	<i>setosa</i>
5.0	3.6	1.4	0.2	<i>setosa</i>
5.4	3.9	1.7	0.4	<i>setosa</i>
4.6	3.4	1.4	0.3	<i>setosa</i>
5.0	3.4	1.5	0.2	<i>setosa</i>
4.4	2.9	1.4	0.2	<i>setosa</i>
4.9	3.1	1.5	0.1	<i>setosa</i>
5.4	3.7	1.5	0.2	<i>setosa</i>
4.8	3.4	1.6	0.2	<i>setosa</i>
4.8	3.0	1.4	0.1	<i>setosa</i>



Iris setosa



Iris virginica



Импорт данных

Скопируем данную таблицу в текстовый файл (для однозначности назовём его `irises.txt`). Затем импортируем его и визуально изучим.

Импорт и подготовка данных

```
example2-1.py - E:\Works\Victor\Students\STDB\Term2\Lecture5\example2-1.py (3.7.2)
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
plt.get_current_fig_manager().window.wm_geometry('1400x750+50+50')

pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)

table = pd.read_excel("../Lecture6/irises.xlsx")

from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()

x = scaler_std.fit_transform(table[['sepal_length', 'sepal_width',
                                     'petal_length', 'petal_width']])
table[['sepal_length', 'sepal_width',
        'petal_length', 'petal_width']] = x
print(table)
```

Ln: 19 Col: 12

Импорт и подготовка данных (текстом)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.get_current_fig_manager().window.wm_geometry('1400x750+50+50')
pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)
table = pd.read_excel("../Lec6/irises.xlsx")
from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']])
table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] = x
print(table)
```

Результат

```
*Python 3.7.2 Shell*
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: E:\Works\Victor\Students\STDB\Term2\Lec3\example2-3.py ====
      sepal_length  sepal_width  petal_length  petal_width  class_label
0      -0.900681      1.019004      -1.340227      -1.315444      setosa
1      -1.143017      -0.131979      -1.340227      -1.315444      setosa
2      -1.385353      0.328414      -1.397064      -1.315444      setosa
3      -1.506521      0.098217      -1.283389      -1.315444      setosa
4      -1.021849      1.249201      -1.340227      -1.315444      setosa
5      -0.537178      1.939791      -1.169714      -1.052180      setosa
6      -1.506521      0.788808      -1.340227      -1.183812      setosa
7      -1.021849      0.788808      -1.283389      -1.315444      setosa
8      -1.748856      -0.362176      -1.340227      -1.315444      setosa
Ln: 9 Col: 0
```

Подготовка данных

Для эффективной работы MLP необходимо убрать доминирование одних переменных над другими за счёт разницы абсолютных значений. Обычно для этого выполняют процедуру стандартизации данных. Данная задача в Python решается при помощи класса `StandardScaler` модуля `preprocessing` библиотеки `Scikit-Learn`.

Стандартизацию мы уже сделали сразу после импорта данных.

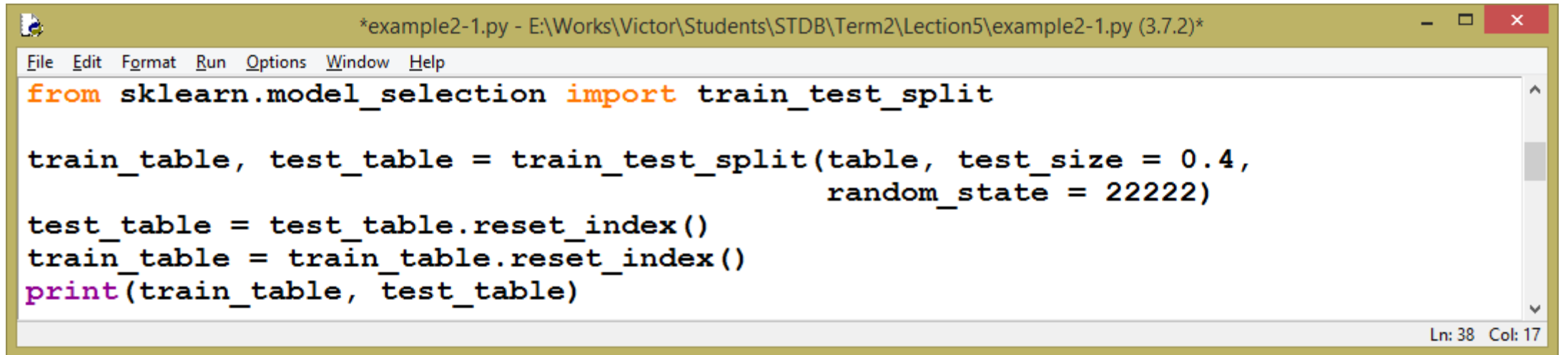
Стандартизация данных

Стандартизация данных – это такое биективное отображение данных из пространства действительных чисел в пространство действительных чисел, при котором данные оказываются распределёнными вокруг 0 со стандартным отклонением 1:

$$x' = \frac{x - M_x}{\sigma_x},$$

где M_x – математическое ожидание (среднее арифметическое) величины x , а σ_x – стандартное отклонение величины x .

Обучающая и тестовая выборки



```
*example2-1.py - E:\Works\Victor\Students\STDB\Term2\Lecture5\example2-1.py (3.7.2)*
File Edit Format Run Options Window Help
from sklearn.model_selection import train_test_split

train_table, test_table = train_test_split(table, test_size = 0.4,
                                           random_state = 22222)

test_table = test_table.reset_index()
train_table = train_table.reset_index()
print(train_table, test_table)
```

Ln: 38 Col: 17

Обучающая и тестовая выборки (текстом)

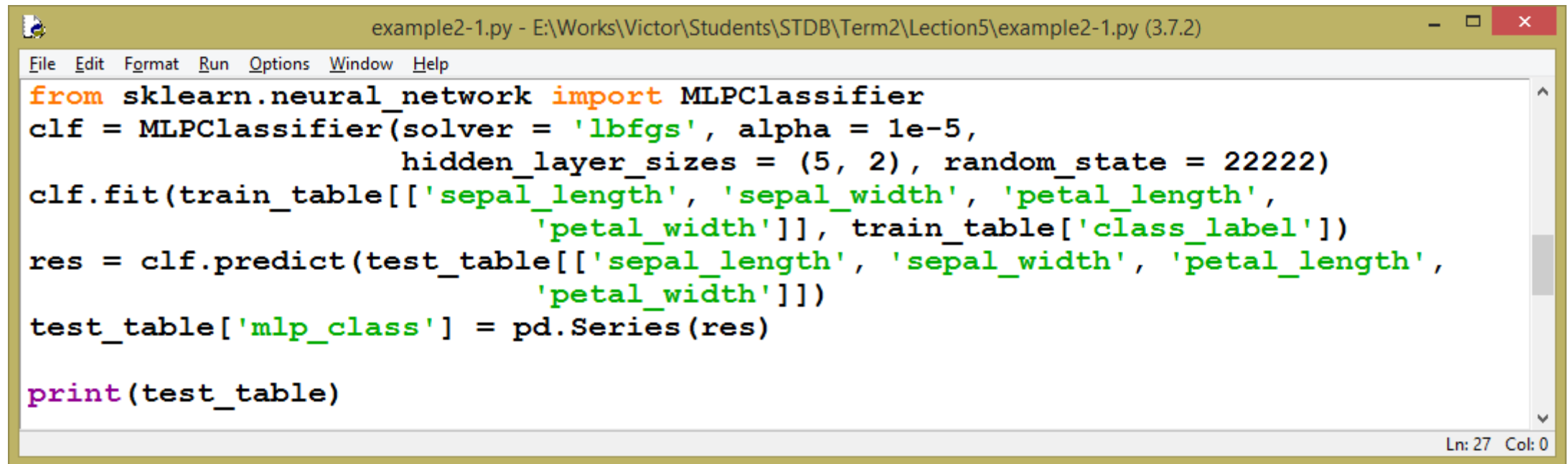
```
from sklearn.model_selection import train_test_split

train_table, test_table = train_test_split(table, test_size = 0.4,
                                           random_state = 22222)

test_table = test_table.reset_index()
train_table = train_table.reset_index()

print(train_table, test_table)
```

MLP

A screenshot of a Python IDE window titled "example2-1.py - E:\Works\Victor\Students\STDB\Term2\Lection5\example2-1.py (3.7.2)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code is written in a monospaced font with syntax highlighting. It imports MLPClassifier from sklearn.neural_network, initializes it with solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), and random_state=22222. It then fits the classifier on train_table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], predicts on test_table with the same features, and assigns the results to a new column 'mlp_class' in test_table. Finally, it prints the test_table. The status bar at the bottom right shows "Ln: 27 Col: 0".

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver = 'lbfgs', alpha = 1e-5,
                    hidden_layer_sizes = (5, 2), random_state = 22222)
clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length',
                    'petal_width']], train_table['class_label'])
res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length',
                    'petal_width']])
test_table['mlp_class'] = pd.Series(res)

print(test_table)
```

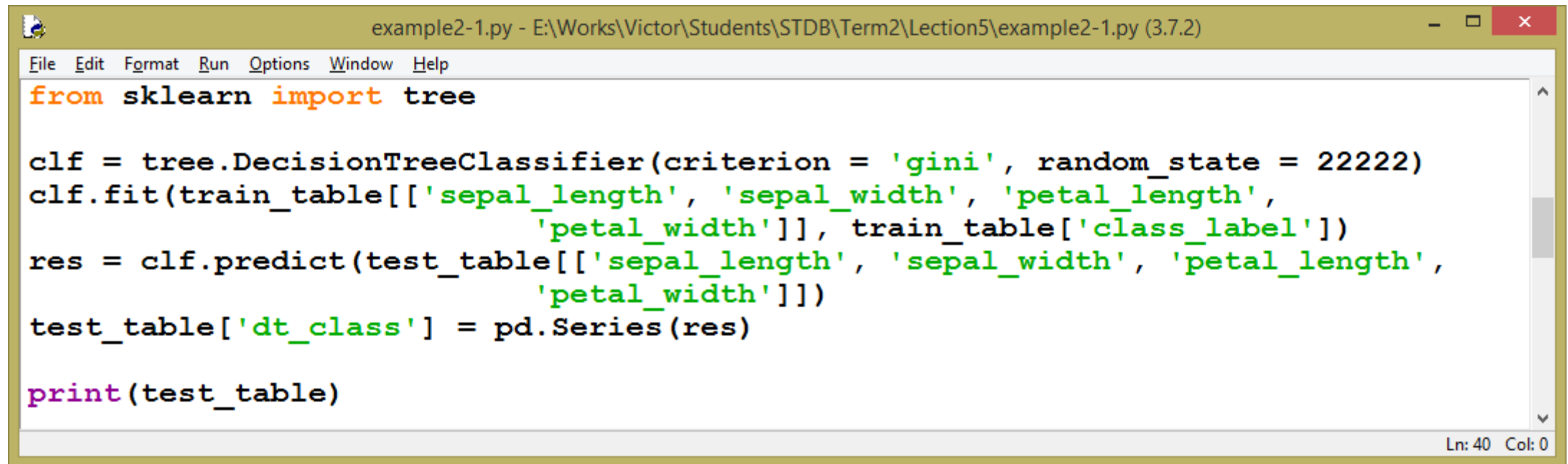
Ln: 27 Col: 0

MLP (ТЕКСТОМ)

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver = 'lbfgs', alpha = 1e-5,
                    hidden_layer_sizes = (5, 2), random_state = 22222)
clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length',
                    'petal_width']], train_table['class_label'])
res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length',
                              'petal_width']])
test_table['mlp_class'] = pd.Series(res)

print(test_table)
```


Сравним с Decision Tree



The image shows a screenshot of a Python IDE window titled "example2-1.py - E:\Works\Victor\Students\STDB\Term2\Lecture5\example2-1.py (3.7.2)". The window contains a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area displays the following Python code:

```
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length',
                    'petal_width']], train_table['class_label'])
res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length',
                              'petal_width']])
test_table['dt_class'] = pd.Series(res)

print(test_table)
```

The status bar at the bottom right indicates "Ln: 40 Col: 0".

Сравним с Decision Tree (текстом)

```
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length',
                    'petal_width']], train_table['class_label'])
res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length',
                              'petal_width']]))
test_table['dt_class'] = pd.Series(res)

print(test_table)
```

Карты цветов и заголовки для рисования

```
example2-1.py - E:\Works\Victor\Students\STDB\Term2\Lecture5\example2-1.py (3.7.2)
File Edit Format Run Options Window Help
colors_map = {'virginica': 'red', 'setosa': 'green', 'versicolor': 'blue'}
test_table['real_color'] = test_table['class_label']
test_table['dt_color'] = test_table['dt_class']
test_table['mlp_color'] = test_table['mlp_class']

for field in ['real_color', 'dt_color', 'mlp_color']:
    test_table[field] = test_table[field].apply(
        lambda x: colors_map[x])

y = test_table.iloc[:, 1]
x = test_table.iloc[:, 2]

titles = ['sepal length', 'sepal width',
          'petal length', 'petal width']
```

Ln: 29 Col: 0

Карты цветов и заголовки для рисования (текстом)

```
colors_map = {'virginica': 'red', 'setosa': 'green', 'versicolor': 'blue'}
test_table['real_color'] = test_table['class_label']
test_table['dt_color'] = test_table['dt_class']
test_table['mlp_color'] = test_table['mlp_class']
for field in ['real_color', 'dt_color', 'mlp_color']:
    test_table[field] = test_table[field].apply(
        lambda x: colors_map[x])
y = test_table.iloc[:, 1]
x = test_table.iloc[:, 2]
titles = ['sepal length', 'sepal width',
          'petal length', 'petal width']
```

Рисуем 3 карты-проекции

```
example2-1.py - E:\Works\Victor\Students\STDB\Term2\Lecture5\example2-1.py (3.7.2)
File Edit Format Run Options Window Help

ax = plt.subplot(1, 3, 1)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = test_table['real_color'], s = 10)
ax = plt.subplot(1, 3, 2)
ax.set_title("Decision Tree", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = test_table['dt_color'], s = 10)
ax = plt.subplot(1, 3, 3)
ax.set_title("MLP", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = test_table['mlp_color'], s = 10)
plt.show()

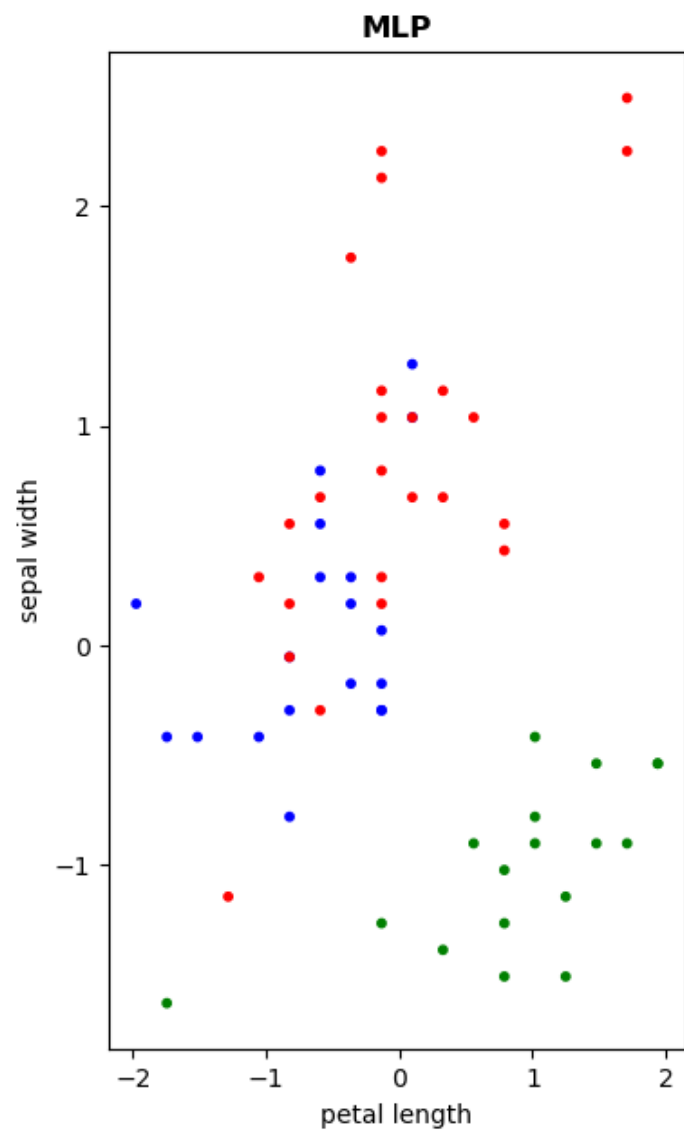
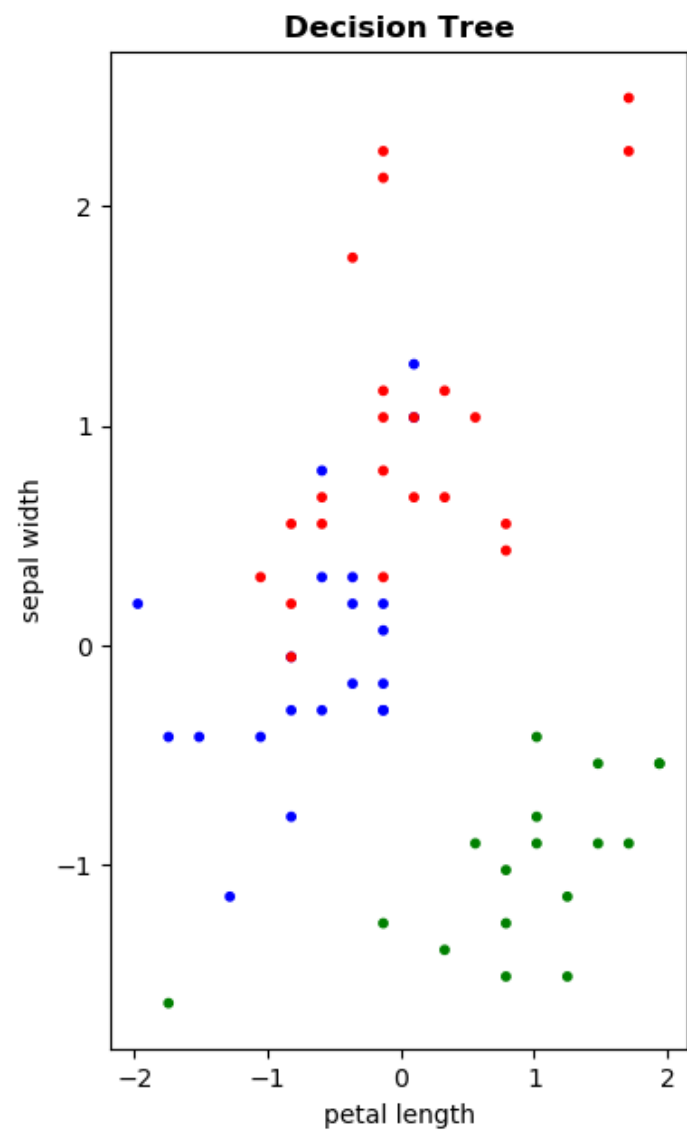
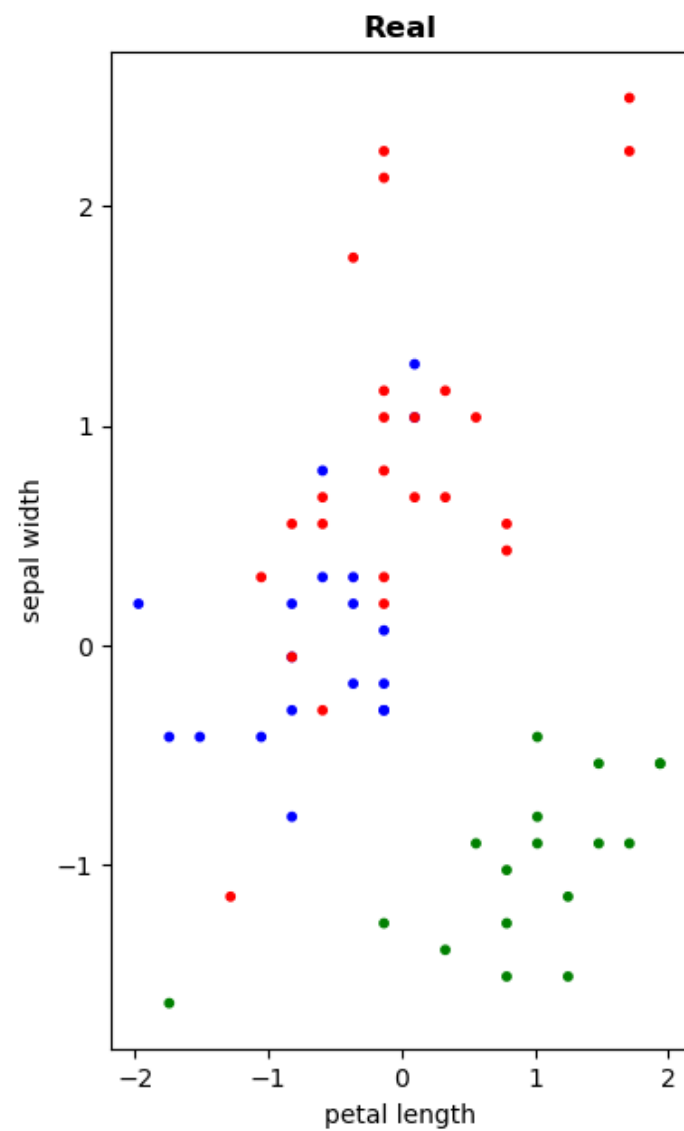
Ln: 51 Col: 0
```

Рисуем 3 карты-проекции (текстом)

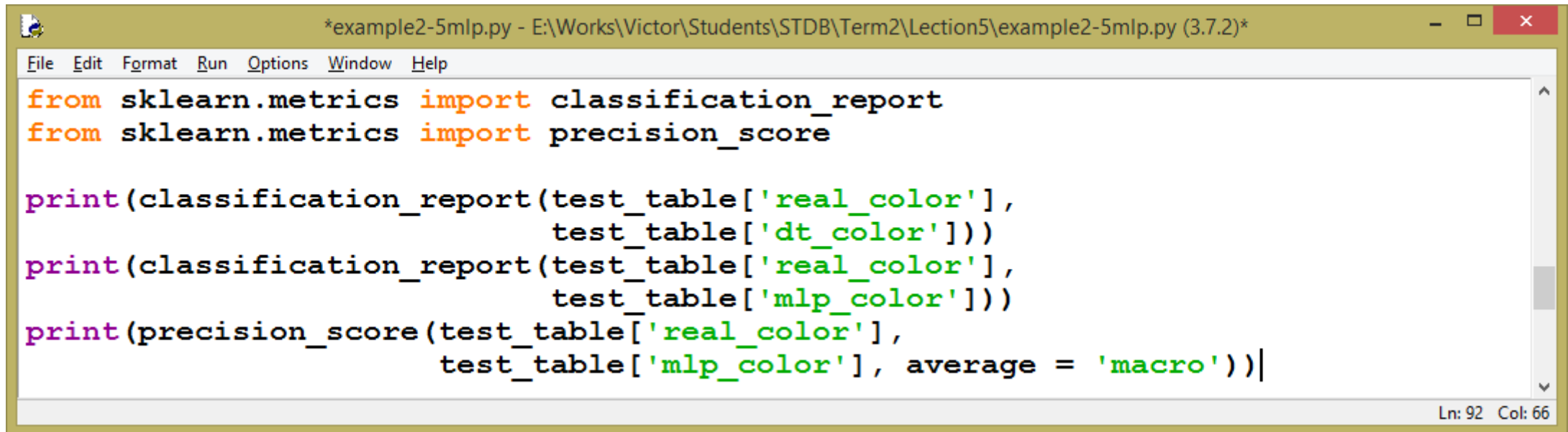
```
ax = plt.subplot(1, 3, 1)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = test_table['real_color'], s = 10)
ax = plt.subplot(1, 3, 2)
ax.set_title("Decision Tree", fontweight='bold')
ax.set_xlabel(titles[2])
```

```
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = test_table['dt_color'], s = 10)
ax = plt.subplot(1, 3, 3)
ax.set_title("MLP", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = test_table['mlp_color'], s = 10)
plt.show()
```

Figure 1



Оценим точность



```
*example2-5mlp.py - E:\Works\Victor\Students\STDB\Term2\Lecture5\example2-5mlp.py (3.7.2)*
File Edit Format Run Options Window Help
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score

print(classification_report(test_table['real_color'],
                           test_table['dt_color']))
print(classification_report(test_table['real_color'],
                           test_table['mlp_color']))
print(precision_score(test_table['real_color'],
                      test_table['mlp_color'], average = 'macro'))|
```

Ln: 92 Col: 66

Оценим точность(текстом)

```
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score

print(classification_report(test_table['real_color'],
                           test_table['dt_color']))
print(classification_report(test_table['real_color'],
                           test_table['mlp_color']))
print(precision_score(test_table['real_color'],
                      test_table['mlp_color'], average = 'macro'))
```

Результат

	precision	recall	f1-score	support
blue	0.86	0.95	0.90	19
green	1.00	1.00	1.00	17
red	0.95	0.88	0.91	24
accuracy			0.93	60
macro avg	0.94	0.94	0.94	60
weighted avg	0.94	0.93	0.93	60

	precision	recall	f1-score	support
blue	0.95	0.95	0.95	19
green	1.00	1.00	1.00	17
red	0.96	0.96	0.96	24
accuracy			0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

0.9685672514619883

```

solvers = ['lbfgs', 'sgd', 'adam']
alphas = [0.00001, 0.000005, 0.000001, 0.1, 0.01]
learning_rates = ['constant', 'invscaling', 'adaptive']
learning_rate_inits = [0.0001, 0.00001, 0.000005, 0.000001, 0.1, 0.01]

best = [0, None, None, None, None, None, None, None]
for slv in solvers:
    for alph in alphas:
        for lr in learning_rates:
            for lri in learning_rate_inits:
                for lay1 in [3, 4, 5, 6]:
                    for lay2 in [2, 3, 4]:
                        clf = MLPClassifier(solver = slv, alpha = alph,
                                           learning_rate = lr, learning_rate_init = lri,
                                           hidden_layer_sizes = (lay1, lay2), random_state = 22222)
                        clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length',
                                           'petal_width']], train_table['class_label'])
                        res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length',
                                           'petal_width']])
                        prec = precision_score(test_table['class_label'], pd.Series(res),
                                           average = 'macro', zero_division = 0)

                        if prec > best[0]:
                            best[0] = prec
                            best[1] = clf
                            best[2] = slv
                            best[3] = alph
                            best[4] = lr
                            best[5] = lri
                            best[6] = lay1
                            best[7] = lay2
                        print("%s %f %s %f %d %d = %f" % (slv, alph, lr, lri, lay1, lay2, prec))

print(best)

```

Подберем параметры. Часть 1 (текстом)

```
solvers = ['lbfgs', 'sgd', 'adam']
alphas = [0.00001, 0.000005, 0.000001, 0.1, 0.01]
learning_rates = ['constant', 'invscaling', 'adaptive']
learning_rate_inits = [0.0001, 0.00001, 0.000005, 0.000001, 0.1, 0.01]

best = [0, None, None, None, None, None, None, None]

for slv in solvers:
    for alph in alphas:
        for lr in learning_rates:
            for lri in learning_rate_inits:
                for lay1 in [3, 4, 5, 6]:
                    for lay2 in [2, 3, 4]:
                        clf = MLPClassifier(solver = slv, alpha = alph, learning_rate = lr, learning_rate_init = lri,
                                             hidden_layer_sizes = (lay1, lay2), random_state = 22222)
```

Подберем параметры. Часть 2 (текстом)

```
clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], train_table['class_label'])
res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']])
prec = precision_score(test_table['class_label'], pd.Series(res), average = 'macro', zero_division = 0)
if prec > best[0]:
    best[0] = prec
    best[1] = clf
    best[2] = slv
    best[3] = alph
    best[4] = lr
    best[5] = lri
    best[6] = lay1
    best[7] = lay2
print("%s %f %s %f %d %d = %f" % (slv, alph, lr, lri, lay1, lay2, prec))

print(best)
```

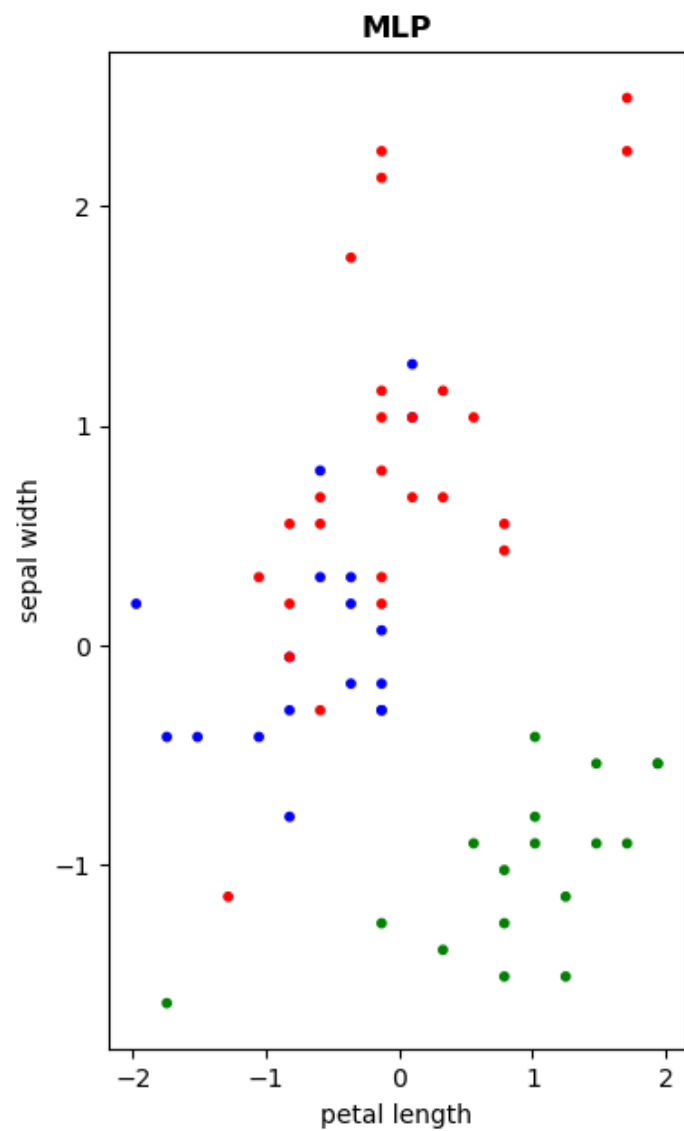
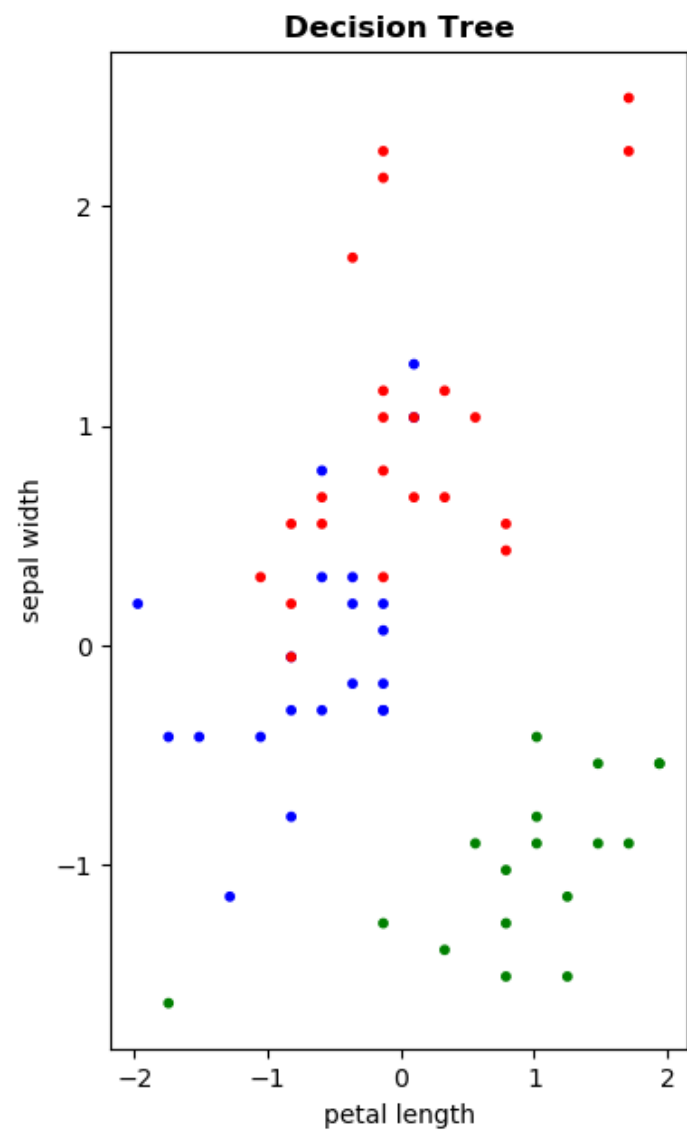
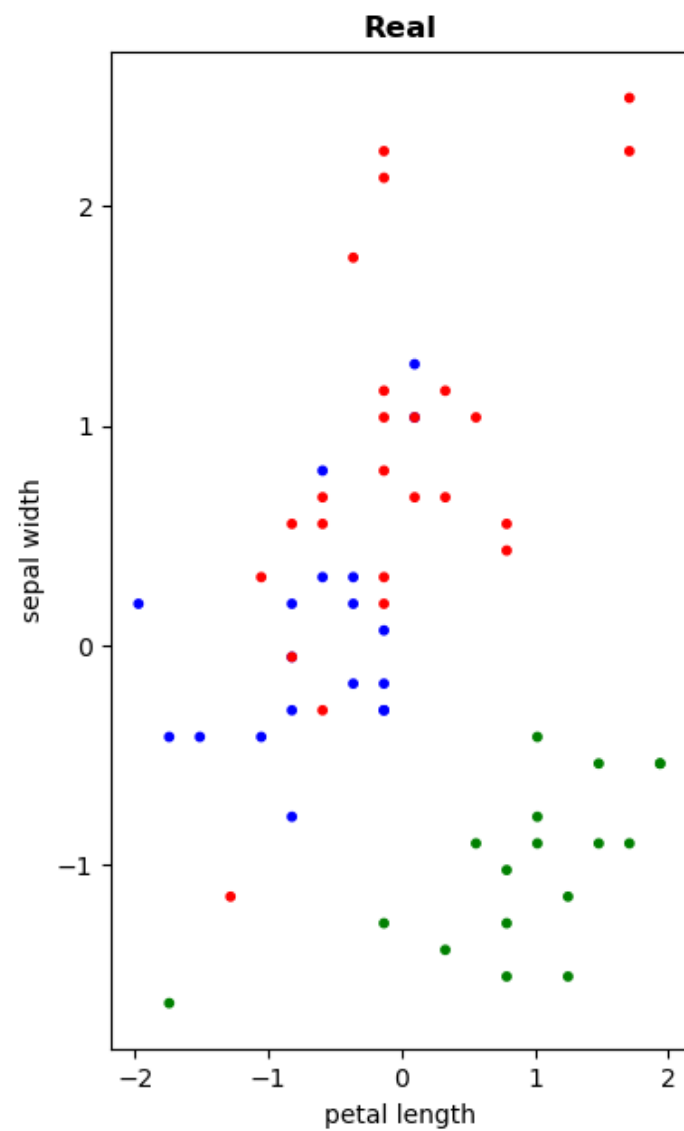
Результат

Много-много предупреждений спустя...

```
Warning (from warnings module):
  File "C:\Users\Radygins\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\normalization\_multilayer_perceptron.py", line 571
    % self.max_iter, ConvergenceWarning)
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
adam 0.010000 adaptive 0.010000 6 4 = 0.937888
[0.9866666666666667, MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
    beta_2=0.999, early_stopping=False, epsilon=1e-08,
    hidden_layer_sizes=(4, 2), learning_rate='constant',
    learning_rate_init=0.01, max_fun=15000, max_iter=200,
    momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
    power_t=0.5, random_state=22222, shuffle=True, solver='sgd',
    tol=0.0001, validation_fraction=0.1, verbose=False,
    warm_start=False), 'sgd', 1e-05, 'constant', 0.01, 4, 2]
```

Ln: 62608 Col: 54

Figure 1



Часть 3

ПОСТРОЕНИЕ НЕЙРОННОЙ СЕТИ
С ПОМОЩЬЮ БИБЛИОТЕКИ KERAS

Keras в Python

Keras в Python — это открытая библиотека для языка программирования Python, которая предназначена для глубокого машинного обучения. Она реализует высокоуровневый API, написанный на Python и способный работать поверх TensorFlow, Theano или CNTK.

Keras позволяет создавать и настраивать модели — схемы, по которым распространяется и подсчитывается информация при обучении.

Keras поддерживает как сверточные, так и рекуррентные сети, в том числе и их комбинации.

Keras позволяет работать как на основном процессоре (CPU), так и на графическом процессоре (GPU).

Особенности библиотеки Keras

Ориентированность на пользователя. Библиотека предоставляет простой и согласованный API, минимизируя количество действий, которые нужно выполнить для решения стандартных задач.

Модульность. Все компоненты Keras, нейронные слои, функции активации, оптимизаторы и другие элементы, легко комбинируются и настраиваются.

Расширяемость. Пользователи могут легко добавлять новые классы и функции. Это делает библиотеку отличным инструментом для научных исследований и экспериментов в области искусственного интеллекта.

TensorFlow

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов.

TensorFlow применяется как для исследований, так и для разработки собственных продуктов Google.

Основной API для работы с библиотекой реализован для Python, также существуют реализации для R, C#, C++, Haskell, Java, Go, JavaScript и Swift.

Подготовка к использованию

Перед установкой Keras, необходимо установить один из его движков: TensorFlow, Theano или CNTK. Разработчики рекомендуют TensorFlow.

Требуется последняя версия pip

```
pip install --upgrade pip
```

Установка текущей стабильной версии для использования CPU и GPU

```
pip install tensorflow
```

Запуск контейнера с TensorFlow

Сначала необходимо скачать и установить Docker Desktop,
[<https://docs.docker.com/get-started/get-docker/>],
а затем запустить контейнер с TensorFlow

```
docker pull tensorflow/tensorflow:latest # Загрузка последнего стабильного образа
```

```
docker run -it -p 8888:8888 tensorflow/tensorflow:latest-jupyter # Запуск Jupyter сервера
```

Использование Keras для обучения нейронных сетей

Установка Keras

```
pip install keras
```

Импорт модулей

```
from keras.models import Sequential  
from keras.layers import Dense
```

Определение архитектуры нейронной сети

Необходимо создать объект **Sequential**, который будет представлять модель, и добавить слои в модель с помощью метода **add()**.

Использование Keras для обучения нейронных сетей

Создание модели (компиляция)

С помощью метода **compile()** указывают оптимизатор, функцию потерь и метрики для оценки качества модели.

Обучение нейронной сети на данных

Модель обучают на данных с помощью метода **fit()**.

Использование обученной нейронной сети для предсказаний

Для предсказаний используют метод **predict()**.

Оценка качества предсказания

Для оценки качества предсказаний используют метод **evaluate()**.

Построение модели с помощью Keras

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import preprocessing
```


Построение модели с помощью Keras

```
pd.set_option('display.max_columns', 2000)
```

```
pd.set_option('display.width', 2000)
```

```
table = pd.read_excel("irises.xlsx")
```

```
scaler_std = preprocessing.StandardScaler()
```

```
x = scaler_std.fit_transform(table[['sepal_length', 'sepal_width', 'petal_length',  
'petal_width']])
```

```
table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] = x
```

Построение модели с помощью Keras

```
label_codes = {'virginica': 0, 'setosa': 1, 'versicolor': 2}
table['class_label'] = table['class_label'].apply(
    lambda x: label_codes[x])

train_table, test_table = train_test_split(table, test_size = 0.4,
                                           random_state = 22222)

test_table = test_table.reset_index()
train_table = train_table.reset_index()
```

Построение модели с помощью Keras

```
x_train = np.array(train_table[['sepal_length', 'sepal_width', 'petal_length',  
'petal_width']])  
y_train = np.array(train_table['class_label'])  
y_train = to_categorical(y_train)  
y_train.shape  
print(y_train)  
  
x_test = np.array(test_table[['sepal_length', 'sepal_width', 'petal_length',  
'petal_width']])  
y_test = np.array(test_table['class_label'])  
y_test = to_categorical(y_test)
```

Построение модели с помощью Keras

```
model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu')) ## спрямленный линейный блок
model.add(Dense(16, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax')) ## функция распределения вероятностей

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

model.fit(x_train, y_train, batch_size=8, epochs=240, steps_per_epoch=10, verbose=2)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
print(loss_and_metrics)
```

Подготовка категориальных данных для каждого класса

```
= RESTART: C:/Users/dukme/Desktop/Lectons STBD/example-keras.py  
[[1. 0. 0.]  
 [1. 0. 0.]  
 [1. 0. 0.]  
 [1. 0. 0.]  
 [0. 1. 0.]  
 [1. 0. 0.]  
 [0. 0. 1.]  
 [0. 0. 1.]  
 [0. 1. 0.]  
 [0. 0. 1.]  
 [0. 1. 0.]  
 [0. 0. 1.]  
 [0. 0. 1.]  
 [0. 1. 0.]  
 [1. 0. 0.]  
 [0. 0. 1.]  
 [0. 0. 1.]
```

Описание модели по слоям

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	80
dense_1 (Dense)	(None, 16)	272
dense_2 (Dense)	(None, 3)	51

Total params: 403 (1.57 KB)
Trainable params: 403 (1.57 KB)
Non-trainable params: 0 (0.00 B)

Обучение модели по эпохам и этапам

```
10/10 - 0s - 3ms/step - accuracy: 0.6625 - loss: 0.9200
Epoch 6/240
10/10 - 0s - 4ms/step - accuracy: 0.7000 - loss: 0.8743
Epoch 7/240
10/10 - 0s - 3ms/step - accuracy: 0.6750 - loss: 0.8535
Epoch 8/240
10/10 - 0s - 4ms/step - accuracy: 0.6000 - loss: 0.8800
Epoch 9/240
10/10 - 0s - 4ms/step - accuracy: 0.6500 - loss: 0.8033
Epoch 10/240
10/10 - 0s - 4ms/step - accuracy: 0.8000 - loss: 0.7462
Epoch 11/240
10/10 - 0s - 4ms/step - accuracy: 0.7000 - loss: 0.7519
Epoch 12/240
10/10 - 0s - 4ms/step - accuracy: 0.7000 - loss: 0.6570
Epoch 13/240
10/10 - 0s - 3ms/step - accuracy: 0.7125 - loss: 0.6846
Epoch 14/240
10/10 - 0s - 4ms/step - accuracy: 0.8000 - loss: 0.6848
```

Обучение модели по эпохам и этапам

```
Epoch 234/240
10/10 - 0s - 4ms/step - accuracy: 1.0000 - loss: 0.0157
Epoch 235/240
10/10 - 0s - 3ms/step - accuracy: 0.9875 - loss: 0.0421
Epoch 236/240
10/10 - 0s - 3ms/step - accuracy: 1.0000 - loss: 0.0139
Epoch 237/240
10/10 - 0s - 3ms/step - accuracy: 0.9875 - loss: 0.0434
Epoch 238/240
10/10 - 0s - 3ms/step - accuracy: 1.0000 - loss: 0.0041
Epoch 239/240
10/10 - 0s - 3ms/step - accuracy: 0.9875 - loss: 0.0345
Epoch 240/240
10/10 - 0s - 4ms/step - accuracy: 1.0000 - loss: 0.0683
```


Оценка результата в соответствии с выбранной метрикой

```
Epoch 238/240
10/10 - 0s - 3ms/step - accuracy: 1.0000 - loss: 0.0041
Epoch 239/240
10/10 - 0s - 3ms/step - accuracy: 0.9875 - loss: 0.0345
Epoch 240/240
10/10 - 0s - 4ms/step - accuracy: 1.0000 - loss: 0.0683
[1m1/1][0m ][32m _____][0m][37m][0m ][
[1m1/1][0m ][32m
s][0m 257ms/step - accuracy: 0.9500 - loss: 0.1313
[0.13134457170963287, 0.949999988079071]
[0.13134457170963287, 0.949999988079071]
```

Интернет ресурсы и литература

1. https://ru.wikipedia.org/wiki/%D0%98%D1%80%D0%B8%D1%81%D1%8B_%D0%A4%D0%B8%D1%88%D0%B5%D1%80%D0%B0 – википедия об ирисах Фишера.
2. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
3. https://scikit-learn.org/stable/modules/neural_networks_supervised.html
4. <http://www.vestnik.vsu.ru/pdf/analiz/2018/04/2018-04-15.pdf>
5. <https://keras.io/>
6. <https://habr.com/ru/companies/otus/articles/787626/>