Однослойный Персептрон Розенблатта. Простейший нейро-сетевой классификатор.

Прежде чем мы приступим к разбору однослойного Персептрона необходимо понимать, что задачи, которые он помогает решать, могут быть решены и без использования методов нейронных сетей. Однако нейронные сети из таких персептронов способны решать действительно нетривиальные задачи.

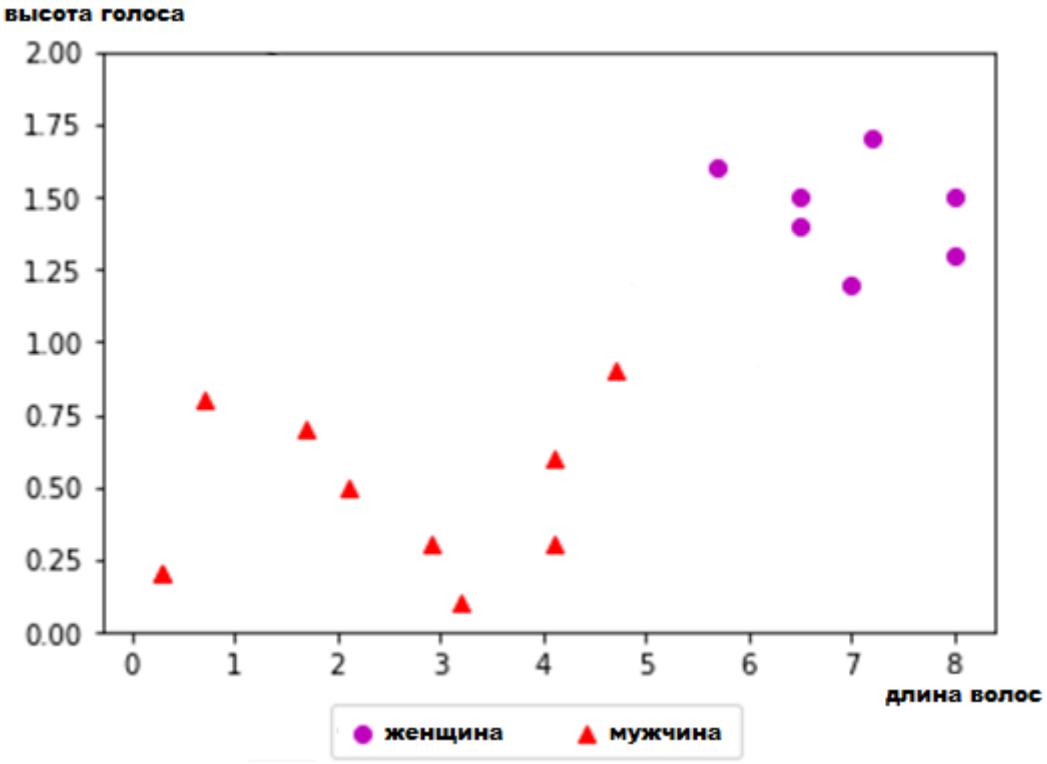
Теоретическая часть

Представим следующую задачу. Нам предоставили список людей, о которых мы ничего не знаем. Нам доступны только 2 параметра о каждом человеке: длина волос и высота голоса. Задача: *опираясь на данные о каждом человеке, с какой-то допустимой погрешностью, оценить пол каждого из них*.

Итак, мы имеем n человек, по каждому из которых имеем показатели по 2-м признакам.

$$G1(x1, y1), \ldots, Gn(xn, yn)$$

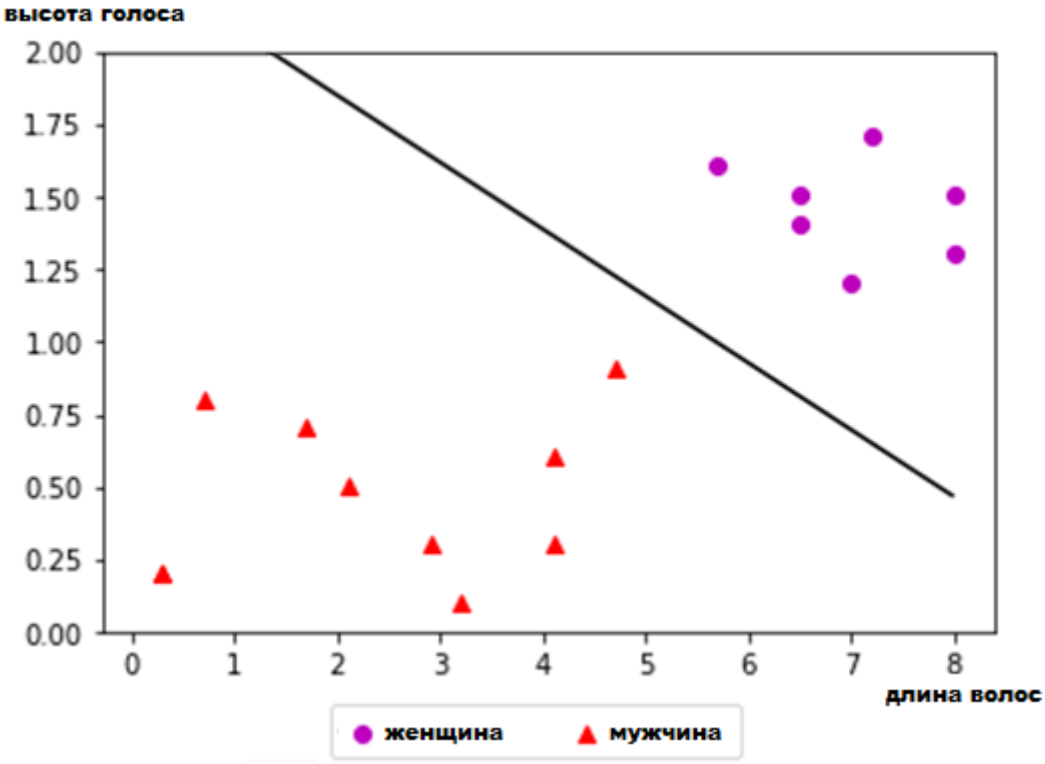
В таком случае мы имеем **двумерное пространство признаков**, на котором каждый человек будет описываться точкой.



Задача персептрона оптимально разделить 2 класса множеств на пространстве признаков одной прямой.

Предполагается, что задача является **линейно разделимой**. Это означает, что возможно одной прямой разделить множество на два класса (в нашем случае два пола). *Однослойный персептрон возможно использовать только для решения задач в линейно разделимом пространстве признаков* (в данном случае, для объяснения, мы используем допущение,

что множество мужчин и женщин можно с нулевой погрешностью линейно разделить по длине волос и высоте голоса).



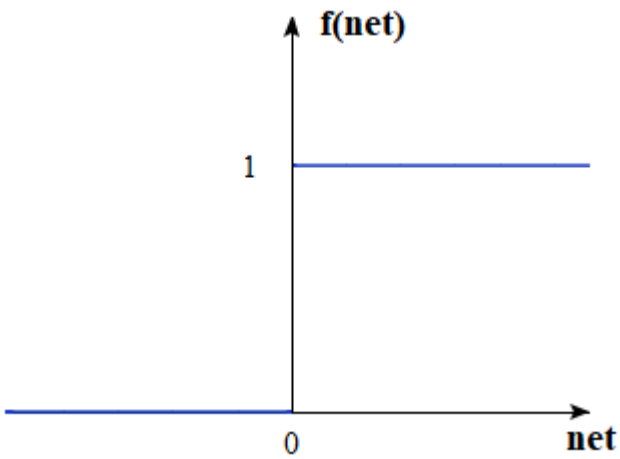
Эта прямая есть нейрон, описанный в персептроне

$$net = xw + yv + b$$

$$y = -x\frac{w+b}{v}, \text{ где } net = 0$$

Теперь становится понятно для чего была необходима искусственная единица с весом b. В случае отсутствия коэффициента b в уравнении прямой степени свободы этой прямой будет достаточно только для управления углом в точке $(0; 0)$.

Итак, при $net = 0$ мы имеем прямую, разделяющую это линейно разделимое множество на 2 класса. В таком случае при $net > 0$ определяется класс женщин, а при $net < 0$ класс мужчин. Значит, для определения этого класса мы будем использовать пороговую функцию, которая позволит определить принадлежность к классу для какой-либо точки. Эта функция называется **функцией активации**.



Исходя из этого, мужчина примет значение $f(net) = 0$, а женщина значение $f(net) = 1$.

Задача сводится к подбору оптимальных значений коэффициентов *(w, v, b)* прямой *net*, которые являются весами персептрона. Процесс, в результате которого определяются оптимальные значения весов называется **обучением**.

Обучение заключается в минимизации ошибки путем итерационного обновления весов по описанному ниже алгоритму. Множество всех точек, которое мы будем использовать для обучения называется **обучающим множеством**. В данной задаче оно представляет из себя набор точек (x, y), с подготовленными для них ответами t ($t = 0$ – мужчина, $t = 1$ – женщина). Множество, на котором мы будем проверять персептрон, называется **тестовым** (сформировано по тому же принципу)

Реализация и обучение персептрона:

#формирование нейрона

$$net = xw + yv + b$$

#функция активации в нейроне

Если $net > 0$, то $out = 1$. Иначе $out = 0$
(допущена некоторая погрешность для случая $net = 0$)

#определение ошибки

$$e = t - out,$$

где t-реальные данные для точки (x, y)

#обновление весов

$$w_x \mathrel{+}= e * n * x$$

$$w_y \mathrel{+}= e * n * y$$

$$w_1 \mathrel{+}= e * n * 1$$

До тех пор, пока ошибка для каждой точки (x, y) обучающего множества не будет равна 0, мы будем продолжать обучение путем повтора итераций по всему обучающему множеству. Каждый такой полный повтор обучающего множества называется **эпохой**.

Затем, в процессе тестирования, персептрон без обучения проверяется на соответствие ответов. И если ошибка оказалась нулевой, то персептрон можно считать обученным. Значения весов подобраны оптимально, и прямая разделила множество на 2 класса.

Мы имеем персептрон, способный по описанным признакам определить пол человека.

Практическая часть

Реализация и обучение персептрона. Тестирование и визуализация результатов

Этот код так же доступен по ссылке https://github.com/YSRoot/NeuralNetwork

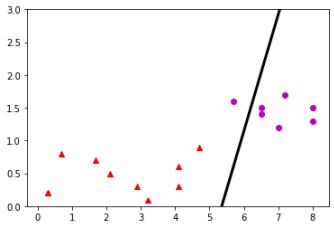
```
# библиотека для работы с матричными операциями
import numpy as np
# библиотека для работы с графиками
import matplotlib.pyplot as plt

# Функция тренировки нейронной сети
# inputs_list - входные данные,
# targets_list - целевые данные,
# weight - веса,
# learning_rate - скорость обучения
def train(inputs_list, targets_list, weight, learning_rate):
    # размерность матрицы входных данных (input_list)
    n, m = inputs_list.shape
    # количество эпох
    era = 0
    # Главный цикл обучения, повторяется пока глобальная ошибка не будет
равна 0
    while True:
        # Глобальная ошибка
        error = 0
        # побочный цикл, прогоняющий данные с input_list
        # функция enumerate(matrix) возвращает индекс и значение строк
        # которая сохраняется в переменные i, value
        # i - индекс строки input_list
        # value - переменная которая хранит в себе строки матрицы input_list
        for i,value in enumerate(inputs_list):
            # вычисляется net с помощью скалярного произведения (вектора на
вектор)
            net = np.dot(value, weight)
            # если условия выполняются out = 0 иначе out = 1
            if net < 0:
                out = 0
            else:
                out = 1
            # условие выше можно написать и по-другому
            # out = 0. if net < 0 else 1.

            # вычисляется отклонение результата от правильного ответа
            # это отклонение сохраняется в переменную local_error
            local_error = targets_list[i] - out
            # если локальная ошибка существует то
            if local_error != 0:
                # изменяются веса
                weight += value*learning_rate*local_error
                # и глобальная ошибка увеличивается на единицу
                error += 1
        # счетчик эпох
        era += 1
        # Просмотр за статусом обучения персептрона
#         query(inputs_list,targets_list,weight)

        # если глобальная ошибка равна 0, т.е. ошибок нет, выход из цикла
        if error == 0: break
    # вывод количества пройденных эпох
    print("Количество пройденных эпох - " + str(era))
    # возвращается веса в конце выполнения работы функции
    return weight
```

```
# функция для проверки обученной сети и вывода результата
def query(inputs_list, targets_list, weight):
    # размерность матрицы input_list
    n,m = inputs_list.shape
    # Глобальная ошибка
    error = 0
    # x - вектор хранящий в себе диапазон значений
    # от минимального значения до максимального, матрицы input_list с шагом
0.02
    x = np.arange(np.min(inputs_list), np.max(inputs_list),0.02)
    # Y функция y = a*x + b для отрисовки прямой разделяющей данные
    Y = np.vectorize(lambda x: ((-weight[0]-weight[1]*x)/weight[2]))
    # y получает значение "Y(x)" для каждого "x" (y — это вектор)
    y = Y(x)
    # цикл, прогоняющий данные с input_list
    # функция enumerate(matrix) возвращает индекс и значение строк
    # которая сохраняется в переменные i, value
    # i - индекс строки input_list
    # value - переменная которая хранит в себе строки матрицы input_list
    for i,value in enumerate(inputs_list):
        # вычисляется net с помощью скалярного произведения вектора на вектор
        net = np.dot(value, weight)
        # если условия выполняются out = 0 иначе out = 1
        if net < 0:
            out = 0
        else:
            out = 1

        # отрисовка точек в зависимости от класса (мужчины или женщины)
        if(targets_list[i] == 0):
            # plt.scatter(x,y, marker, color)
            # scatter() принимает на вход несколько аргументов:
            # "x", "y", вид маркера, цвет маркера
            # если условие выполняется - женщины
            plt.scatter(value[1], value[2], marker='o', color = 'm')
        else:
            # иначе - мужчины
            plt.scatter(value[1], value[2], marker='^', color = 'r')

        # вычисляется отклонение результата от правильного ответа
        # это отклонение сохраняется в переменную local_error
        local_error = targets_list[i] - out
        # если локальная ошибка существует, то глобальная ошибка
увеличивается
        error += local_error
    # plt.plot() рисует прямую по координатам
    # на вход идет вектор "x" и вектор "y"
    # так же цвет и толщина линии
    plt.plot(x, y, color = 'k', linewidth = 3)
    # в переменные x1, x2, y1, y2 сохраняются нижние и верхние границы
графика
    x1, x2, y1, y2 = plt.axis()
    # ставятся ограничения отображения графика по оси "y" от 0 до 3
    plt.axis((x1, x2, 0, 3))
    # показать график
    plt.show()
    # Вывод ошибки (погрешности)
    print("Погрешность - " + str(error))
    pass
#скорость обучения
lr = 0.7
#обучающее множество
inputs = np.array([
        [1.,0.3,0.2],
```

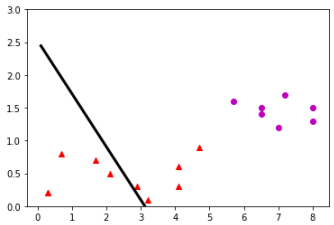
```
        [1.,0.3,0.2],
        [1.,8.0,1.5],
        [1.,0.7,0.8],
        [1.,2.1,0.5],
        [1.,7.2,1.7],
        [1.,1.7,0.7],
        [1.,4.7,0.9],
        [1.,2.9,0.3],
        [1.,6.5,1.5],
        [1.,5.7,1.6],
        [1.,4.1,0.3],
        [1.,3.2,0.1],
        [1.,7.0,1.2],
        [1.,8.0,1.3],
        [1.,4.1,0.6],
        [1.,6.5,1.4]])
# целевые данные
targets = np.array([1,1,0,1,1,0,1,1,1,0,0,1,1,0,0,1,0])
# np.random.seed ставит «семя» для функции random
# в питоне, как и в других высокоуровневых языках стоит псевдорандом
# возвращающее число, созданное на основе другого числа
# по умолчанию — время на данный момент
# мы меняем стандартное значение «seed» на свое
np.random.seed(2)
#случайные веса
W = np.random.random_sample(3)
# тренировка сети, возвращает веса и сохраняет в переменную W
W = train(inputs, targets, W, lr)
# Проверка данных на линейную разделимость
query(inputs, targets, W)
# Данные для проверки на необученных значениях
# раскомментируйте и проверьте на правильность обученной сети
#inpt=np.array([
#         [1,3.2,1.1],
#         [1,2.8,0.7],
#         [1,1.5,0.3],
#         [1,6.0,1.7],
#         [1,7.2,1.3]])
#tt = np.array([1,1,1,0,0])
#query(inpt, tt, W)
```

Результаты

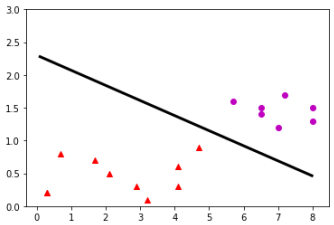
Этапы обучения:



1 эпоха

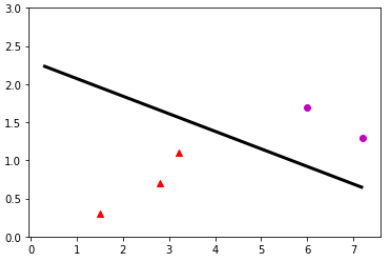


4 эпоха



7 эпоха

Тестирование:



Небольшая документация, которая сможет ответить на некоторые ваши вопросы по коду. Здесь приведены примеры как работают методы (функции) из кода.

Метод **print()** выводит в консоль объект внутри скобок

```
>>> print('Вывод текста в консоль')

Вывод текста в консоль

>>> print(10+5)

15
```

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
>>> i = 5

>>> while i < 10:

...     print(i)

...     i = i + 2

5

7

9
```

Цикл **for** уже чуточку сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла **while**. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
>>> for i in 'hello world':

...     print(i * 2, end='')

hheellllloo  wwoorrlldd
```

Условная инструкция if-elif-else - основной инструмент выбора в Python. Проще говоря, она выбирает, какое действие следует выполнить, в зависимости от значения переменных в момент проверки условия.

```
>>> if 5>2:

...     print('true')

... else:

...     print('false')
```

```
true
```

Для генерации диапазона нужно вызвать функцию range, передав ей от 1 до 3 целочисленных аргументов. В языке Python диапазон является самостоятельным объектом. Чаще всего она используется в циклах for.

```
>>> range(5)
range(0, 5)
>>> range(1,10,3)
range(1,10,3)
```

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

Метод **numpy.array()** трансформирует вложенные последовательности в многомерные массивы. Тип элементов массива зависит от типа элементов исходной последовательности

```
>>> numpy.array([1,2,3,4])
array([14, 32, 50])
>>> numpy.array([[1,2,3],[4,5,6],[7,8,9]])
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Метод **numpy.dot(a, b)**

Для двухмерных массивов это эквивалентно матричному умножению, а для 1-D массивов - скалярному произведению векторов (без комплексного сопряжения). Для размерностей N это суммирующий продукт по последней оси **“a”** и второй по последнему из **“b”**

```
>>> A = numpy.array([[1,2,3],[4,5,6],[7,8,9]])
>>> B = numpy.array([1,2,3,4])
>>> C = numpy.dot(A, B)
>>> print(C)
array([1, 2, 3])
```

$$c[i] = \sum_{i=1}^{n} a[i] * b[i],$$

где $a[i]$ — это строка матрицы $A, b[i]$ — элемент вектора B

Функция **np.random.random(), np.random.random_simple()** - без аргументов возвращает просто число в промежутке [0, 1), с одним целым числом - одномерный массив, с кортежем - массив с размерами, указанными в кортеже (все числа - из промежутка [0, 1)).

```
>>> np.random.sample()
0.6336371838734877
>>> np.random.sample(3)
array([ 0.53478558,  0.1441317 ,  0.15711313])
>>> np.random.sample((2, 3))
array([[ 0.12915769,  0.09448946,  0.58778985],
       [ 0.45488207,  0.19335243,  0.22129977]])
```

Метод **numpy.random.seed()** инициализирует генератора случайных чисел. В Python, как и в любом другом языке, используется т.н. генератор псевдо случайных чисел. Т.е. random выдает не случайное число, а число которое вычисляется алгоритмом на основе другого числа, по умолчанию это текущее время. random.seed позволяет изменить число, которое передается в random для генерации случайного числа, а т.к. "случайные" числа выдаются одним и тем же алгоритмом, то при одинаковом параметре в random.seed будут и одинаковые "случайные" числа.

```
>>> np.random.random(5)
array([0.89629309, 0.12558531, 0.20724288, 0.0514672 , 0.44080984])
>>> np.random.random(5)
array([0.02987621, 0.45683322, 0.64914405, 0.27848728, 0.6762549 ])
#а если перед вызовом рандома вызвать seed()
>>> np.random.seed(3)
>>> np.random.random(5)
array([0.5507979 , 0.70814782, 0.29090474, 0.51082761, 0.89294695])
>>> np.random.seed(3)
>>> np.random.random(5)
array([0.5507979 , 0.70814782, 0.29090474, 0.51082761, 0.89294695])
#как видно последовательность рандома та же
```

Библиотека **matplotlib** — это библиотека двумерной графики для языка python с помощью которой можно создавать высококачественные рисунки различных форматов.

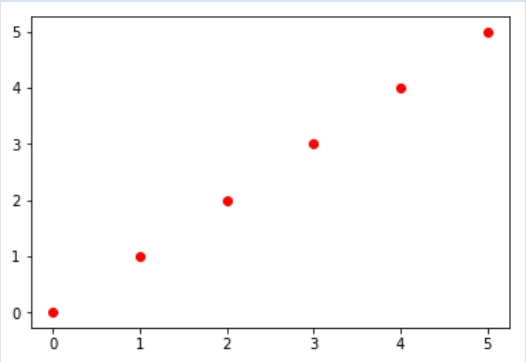
Эта библиотека содержит очень много возможностей для инфографики но мы воспользуемся только pyplot.plot(), pyplot.scatter(), pyplot.legend(), pyplot.show()

plt.show() – показать итоговый график

plt.scatter() - маркер или точечное рисование, аргументы:

- s - размер маркера, как для 1 значения, так и для списка
- color - цвет маркера, как для 1 значения, так и для списка
- ...

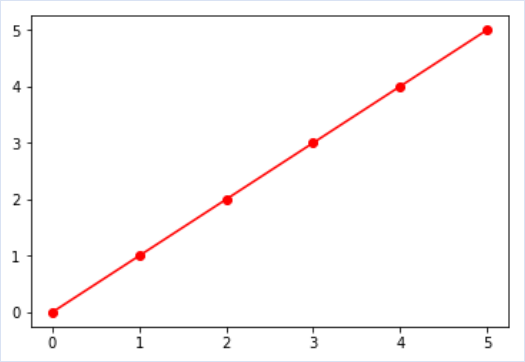
```
>>> import matplotlib.pyplot as plt

>>> x = np.arange(6)

>>> y = np.arange(6)

>>> plt.scatter(x,y,color='red')

>>> plt.show()
```



plt.plot() - ломаная линия на вход принимает аргументы:

- color - цвет линии
- label - строка легенды
- line_format - идет сразу после координат, тип линии, цвет линии, маркер точек, задается строкой
- linestyle - стиль линии
- linewidth - ширина линии
- marker - маркер точек
- markersize - размер маркера
- ...

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

>>> x = np.arange(6)

>>> y = np.arange(6)

>>> plt.plot(x,y,color='red',marker='o')

>>> plt.show()
```



plt.legend() – добавление легенды в график, на вход принимает аргументы:

- borderaxespad - величина зазора между осями и легендой
- legend_names - список названии легенд, лучше задавать при построении графика
- loc - местоположение вывода данных легенды, можно задать как числом, так и строкой, а также кортежом позиции
- ncol - количество столбцов для легенды
- ...

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

>>> x = np.arange(6)

>>> y = np.arange(6)

>>> plt.plot(x,y,color='red',marker='o', label = "example" )

>>> plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.10), ncol=1)

>>> plt.show()
```