Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Отчет по лабораторной работе №1

по курсу «Нейроинформатика»

Студент: Полей-Добронравова Амелия Вадимовна

Группа: М8О-407Б, № по списку 20.

Преподаватель: Аносова Н.П.

Дата: 20.09.2021

Итоговая оценка:

Подпись преподавателя:

Москва, 2021

Тема лабораторной: “Персептроны. Процедура обучения Розенблатта.”

Целью работы является исследование свойств персептрона Розенблатта и его применение для решения задачи распознавания образов.

Основные этапы работы:

1. Для первой обучающей выборки построить и обучить сеть, которая будет правильно относить точки к двум классам. Отобразить дискриминантную линию и проверить качество обучения.

2. Изменить обучающее множество так, чтобы классы стали линейно неразделимыми. Проверить возможности обучения по правилу Розенблатта.

3. Для второй обучающей выборки построить и обучить сеть, которая будет правильно относить точки к четырем классам. Отобразить дискриминантную линию и проверить качество обучения.

Вариант 20

20.	$\begin{bmatrix} 4.3 & 2.3 & 3.6 & 4.8 & 2.8 & -3.3 \\ 2.2 & -4.4 & 4.3 & 3.5 & 0.1 & -1.1 \end{bmatrix}$ $\begin{bmatrix} -4.4 & 0.2 & 1.5 & -2.1 & -4.9 & -3.4 & -1.3 & -0.2 \\ -1.1 & -0.9 & 1.2 & -0.7 & 4.8 & -4 & -3.1 & -1.7 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$

Ход работы

Пресептрон Розенблатта - это сеть, состоящая из S, A и R нейронных элементов. Нейроны слоя S называются сенсорными и предназначены для формирования входных сигналов в результате внешних воздействий. Нейроны слоя A называются ассоциативными и предназначены для непосредственной обработки входной информации. Нейроны слоя R называются эффекторными. Они служат для передачи сигналов возбуждения к соответствующему объекту, например к мышцам.

Процедура обучения Розенблатта состоит из следующих шагов:

1. Весовые коэффициенты W нейронной сети инициализируются случайным образом или устанавливаются в нулевое состояние.

2. На входы сети поочередно подаются входные образы X из обучающей выборки, которые трансформируются в выходные сигналы нейронных элементов Y.

3. Если реакция нейронной сети y_j совпадает с эталонным значением t_j, т.е. если $y_j=t_j$, то весовой коэффициент w_{ij} не изменяется.

4. Если выходная реакция сети y_j не совпадает с эталонной, то производится модификация весовых коэффициентов по правилу
$w_{i+1}=w_i+speed * x_j * error$.
$0 < speed < 1$, $error = y_j - t_j$.

5 Шаги 2-4 повторяются до тех пор, пока не станет $y_j=t_j$, для всех входных образов, или не перестанут изменяться весовые коэффициенты.

В программе я создала три класса:

Preceptron - обычный пресептрон для бинарной классификации.
MulticlassPreceptron - пресептрон для многоклассовой классификации с числом выходов равных числу классов.
MulticlassPreceptron_1 - пресептрон для многоклассовой классификации с двумя выходами в соответствии с заданием.

Функция fit - обучение на обучающей выборке на заданном в конструкторе числе эпох. Функция predict - предсказание для одной точки.

Функция loss это подсчет среднего числа ошибок по выборке за 1 эпоху.

Исходный код

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
```

```
[55] m1 = np.array([[4.3, 2.3, 3.6, 4.8, 2.8, -3.3],[2.2, -4.4, 4.3, 3.5, 0.1, -1.1],[1, 0, 1, 1, 1, 0]])
```

```
[56] m1
```

```
array([[ 4.3,  2.3,  3.6,  4.8,  2.8, -3.3],
       [ 2.2, -4.4,  4.3,  3.5,  0.1, -1.1],
       [ 1. ,  0. ,  1. ,  1. ,  1. ,  0. ]])
```

```
[207] class Preceptron(object):
        def __init__(self, eps = 0.0001, iter = 15):
          self.epsilon = eps
          self.iterations = iter
          self.th = random.random()

        def predict_proba(self, X):
          return np.dot(X, self.w) + self.th

        def predict(self, X):
          r = np.dot(X, self.w) + self.th
```

```
          return r >= 0

        def fit(self, X, y):
            np.random.seed(42)
            self.w = np.random.rand(X.shape[1])
            loss = []
            for i in range(self.iterations):
              er = 0
              for xi, yi in zip(X, y):
                pred = self.predict(xi)
                #er += pred - yi
                if (pred != yi):
                  er += 1
                  self.w -= self.epsilon * np.dot(xi.transpose(), pred - yi)
                  self.th -= self.epsilon * (pred - yi)
              er = er / len(y)
              loss.append(er)
            return loss
```

```
[221] p = Preceptron(iter = 5)
```

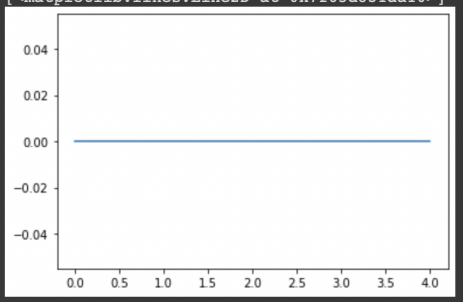
```
[222] X = m1[:-1].transpose()
      print(X.shape)
      y = m1[2].transpose()
      print(y.shape)
```

```
(6, 2)
(6,)
```

```
[223] l = p.fit(X, y)
```

```
plt.plot(l)
```

```
[<matplotlib.lines.Line2D at 0x7f65d831da10>]
```



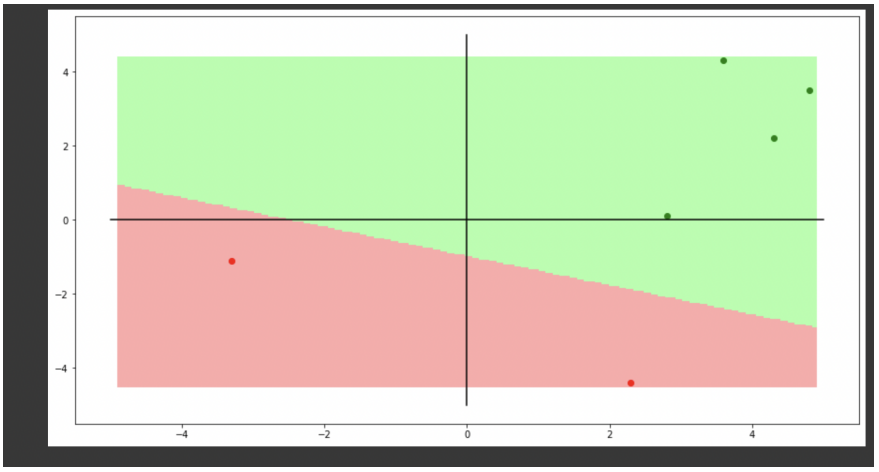

```
[215] #проверка на рандомных точках
      xx = np.array([[7],[2]]).transpose()
      print(xx)
      print(p.predict(xx))
```

```
xx = np.array([[7],[-6]]).transpose()
print(xx)
print(p.predict(xx))
```

```
[[7 2]]
[ True]
[[ 7 -6]]
[False]
```

+ Код + Текст

```
[216] from matplotlib.colors import ListedColormap

    plt.figure(figsize=(15,8))
    eps = 0.1
    xx, yy = np.meshgrid(np.linspace(np.max(X[:,0]) * (-1) - eps, np.max(X[:,0]) + eps, 200),
                         np.linspace(np.min(X[:,1]) - eps, np.max(X[:,1]) + eps, 200))
    Z = p.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    plt.scatter([4.3, 3.6, 4.8, 2.8],[2.2, 4.3, 3.5, 0.1], c = 'green')
    plt.scatter([2.3, -3.3], [-4.4, -1.1], c = 'red')
    plt.plot([0, 0],[-5, 5], c = "black")
    plt.plot([-5, 5],[0, 0], c = "black")
    plt.show()
```



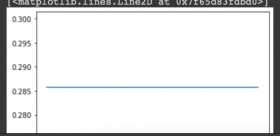

```
#изменение обучающего множества чтобы сделать линейно неразделимым - поменяю класс у 4 точки на противоположный

m2 =  np.array([[4.3, 2.3, 3.6, 4.8, 2.8, -3.3, 3.0],[2.2, -4.4, 4.3, 3.5, 0.1, -1.1, 0.5],[1, 0, 1, 0, 1, 0, 0]])
X2 = m2[:-1].transpose()
print(X2.shape)
y2 = m2[2].transpose()
print(y2.shape)
```

```
(7, 2)
(7,)
```

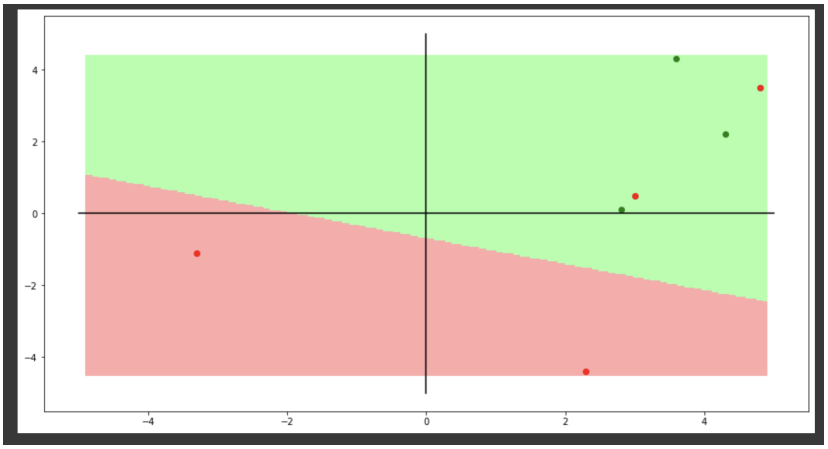
```
[218] p2 = Preceptron(iter = 50)
```

```
[219] l2 = p2.fit(X2, y2)
    plt.plot(l2)
```

```
[<matplotlib.lines.Line2D at 0x7f65d83fdbd0>]
```

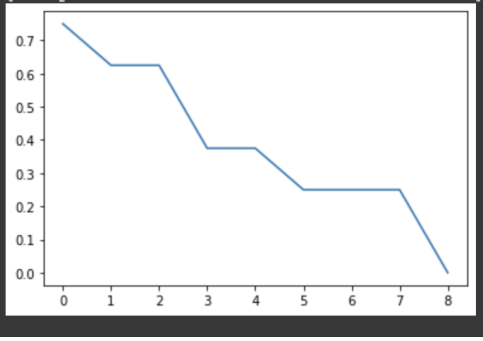



```
[220] from matplotlib.colors import ListedColormap

    plt.figure(figsize=(15,8))
    eps = 0.1
    xx, yy = np.meshgrid(np.linspace(np.max(X2[:,0]) * (-1) - eps, np.max(X2[:,0]) + eps, 200),
                         np.linspace(np.min(X2[:,1]) - eps, np.max(X2[:,1]) + eps, 200))
    Z = p2.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    plt.scatter([4.3, 3.6, 2.8],[2.2, 4.3, 0.1], c = 'green')
    plt.scatter([2.3, -3.3, 4.8, 3.0], [-4.4, -1.1, 3.5, 0.5], c = 'red')
    plt.plot([0, 0],[-5, 5], c = "black")
    plt.plot([-5, 5],[0, 0], c = "black")
    plt.show()
```



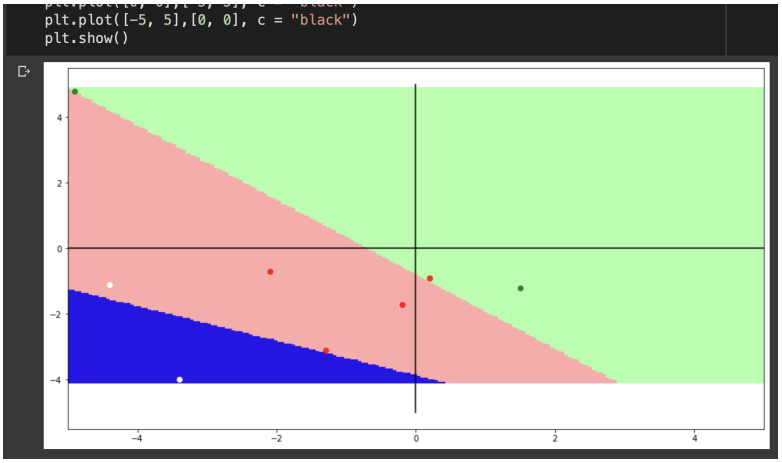
```
[284] class Multiclass_Preceptron_1(object):
        def __init__(self, class_count = 2, eps = 0.01, iter = 15):
          self.epsilon = eps
          self.iterations = iter
          self.k = class_count

        def predict(self, X):
          t = []
          for u in range(self.k):
            t.append(np.dot(X, self.w[u]) + self.th[u] >= 0)
          return np.array(t).transpose()

        def fit(self, X, y):
            np.random.seed(42)
            self.w = np.zeros((self.k, X.shape[1]))
            self.th = np.zeros(self.k)
            loss = []
            for i in range(self.iterations):
              l1 = 0
              for xi, yi in zip(X, y):
                pred = self.predict(xi)
                if (np.any(pred != yi)):
                  l1 += 1
                if (pred[0] != yi[0]):
                  self.w[0] -= self.epsilon * np.dot(xi, pred[0] - yi[0])
```

```
[323] p4 = Multiclass_Preceptron_1(iter = 9)

[324] l4 = p4.fit(X4,y4)

      plt.plot(l4)

      [<matplotlib.lines.Line2D at 0x7f65d40e3550>]
```



```
plt.figure(figsize=(15,8))
eps = 0.1
xx, yy = np.meshgrid(np.linspace(np.min(X4[:,0]) - eps, np.min(X4[:,0]) * -1 + eps, 200),
                     np.linspace(np.min(X4[:,1]) - eps, np.max(X4[:,1]) + eps, 200))
Z = p4.predict(np.c_[xx.ravel(), yy.ravel()])
ZZ = []
for z in Z:
  if z[0] == 0:
    if z[1] == 0:
      ZZ.append(0)
    else:
      ZZ.append(1)
  elif z[0] == 1:
    if z[1] == 0:
      ZZ.append(2)
    else:
      ZZ.append(3)
ZZ = np.array(ZZ)
ZZ = ZZ.reshape(xx.shape).transpose()
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#2715ea'])
plt.pcolormesh(xx, yy, ZZ, cmap=cmap_light)

plt.scatter([-4.4, -3.4], [-1.1, -4], c = 'white')
plt.scatter([0.2, -2.1, -1.3, -0.2], [-0.9, -0.7, -3.1, -1.7], c = 'red')
plt.scatter([1.5, -4.9], [-1.2, 4.8], c = 'green')
plt.plot([0, 0],[-5, 5], c = "black")
```

```
plt.plot([-5, 5],[0, 0], c = "black")
plt.show()
```



Выводы

Согласно теореме сходимости персептрона, алгоритм сходится за конечное число шагов, если существует решение задачи.

Может случиться так, что обучение и вовсе не понадобится, если удачно подбираются классы и начальные значения весов, как у меня в первом пункте задачи: loss с первой же итерации был равен 0, и можно было не учить дальше.

Пресептрон Розенблатта подходит для классификации только линейно разделимых классов, из-за чего во 2 части задачи обучение было бесполезно, loss не сходился к нулю.

Для многоклассовой классификации можно использовать буквально бинарную классификацию, если номер класса представить в двоичном виде, и каждому разряду назначить свой выход нейронной сети.