**Московский авиационный институт**
**(Национальный исследовательский университет)**
Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

# Лабораторная работа № 1

по курсу «Нейроинформатика».

Тема: «Персептроны. Процедура обучения Розенблатта».

Студент: Вельтман Л.Я.
Группа: М8О-407Б
Преподаватели: Тюменцев Ю.В.
Козлов Д.С.
Вариант: 7
Оценка:

Москва, 2020

# Цель работы.

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 правиль- но относить точки к двум классам. Отобразить дискриминантную линию и проверить качество обучения.
2. Изменить обучающее множество так, чтобы классы стали линейно неразделимы- ми. 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.

# Оборудование.

Операционная система: macOS Catalina version 10.15.5

Процессор: 2,3 GHz 2-ядерный процессор Intel Core i5

Оперативная память: 8 ГБ 2133 MHz LPDDR3

# Программное обеспечение.

Работа выполнена на Python3 с применением библиотек numpy (для вычислений), pandas (показать выборки в удобном формате в виде табличек) и matplotlib (для графиков) при помощи командной оболочки Jupyter Notebook.

# Сценарий выполнения работы.

Сначала происходит инициализация весов и смещения рандомными начальными значениями. Далее для первого значения выборки вычисляется вектор ошибки. Ощибка вычисляется по следующей формуле $error[i] = labels[i] - predict[i]$, где $labels[i]$ – истинный результат для i-ого примера, а $predict[i] = heaviside\big(x[i]W + b\big)$- предсказанное значение сети для i-ого примера (x[i] – i-ый вектор входов, W – текущая матрица весов, b – текущий вектор смещений). Если вектор ошибки ненулевой, происходит корректировка весов пропорционально компонентам вектора ошибки и все начинается сначала, но уже с новыми весами ($W = W + \text{lr} * x * error$). Если же вектор ошибки нулевой, то программа переходит к следующему значению в выборке. Программа завершает свою работу, когда дошла до последнего значения в выборке и вычислила на нем нулевую ошибку. Предсказанные значения - это значения, полученные при помощи функции Хевисайда от матричного произведения матрицы весов и значений входов.

**Первая обучающая выборка**

In [7]:
```
sequences = np.array([[-3.8, -0.2, 2.9, -4.5, -4.2, 4.4], [0.4, 3.9, 2.3
labels = np.array([1, 1, 0, 0, 1, 0])
```

In [8]:
```
ds = np.insert(sequences, 2, labels, axis=0)
df = pd.DataFrame(ds.T, columns=['x1', 'x2', 'y'])
```

In [9]:
```
df
```

Out[9]:

| | x1 | x2 | y |
|---|---|---|---|
| **0** | -3.8 | 0.4 | 1.0 |
| **1** | -0.2 | 3.9 | 1.0 |
| **2** | 2.9 | 2.3 | 0.0 |
| **3** | -4.5 | -4.3 | 0.0 |
| **4** | -4.2 | 2.9 | 1.0 |
| **5** | 4.4 | 1.8 | 0.0 |

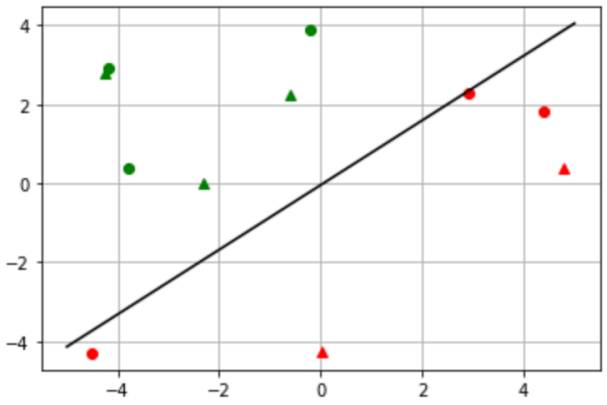
Обучим нейросеть на первой обучающей выборке

In [17]:
```
model = Perceptron(epochs, lr)
weights, bias = model.Train(sequences.T, labels, False)
print('weights: {}, bias: {}'.format(weights, bias))
```

```
iter: 0 weights: [0.9782229  0.45558491] bias: 0.32383276483316237

iter: 1 weights: [0.5982229  0.49558492] bias: 0.42383276483316235

iter: 2 weights: [0.21822291 0.53558492] bias: 0.5238327648331623

iter: 5 weights: [-0.16177709  0.57558492] bias: 0.6238327648331623

iter: 8 weights: [-0.45177711  0.34558492] bias: 0.5238327648331623

iter: 12 weights: [-0.74177713  0.11558491] bias: 0.42383276483316235

iter: 15 weights: [-0.29177711  0.54558495] bias: 0.32383276483316237

iter: 19 weights: [-0.58177714  0.31558495] bias: 0.22383276483316236

iter: 22 weights: [-0.13177712  0.74558498] bias: 0.12383276483316236

weights: [-0.42177714  0.51558498], bias: 0.02383276483316235
```

Отобразим первую обучающую выборку, случайно заданную выборку (треугольнички) и дискриминантную линию.

In [20]:
```
PlotFirst(df)
area = np.linspace(-5, 5, 10)
line = DividingLine(area, weights[0], weights[1], bias)
plt.plot(area, line, color='black')

for x, y, color in zip(randSeqs[:, 0], randSeqs[:, 1], pred):
    plt.plot(x, y, "^", c=colors[color])
```




Изменим первую обучающую выборку так, чтобы она стала линейно неразделимой.

In [21]:
```
sequences_ = np.array([[-3.8, -0.2, 2.9, -4.5, -4.2, 4.4], [0.4, 3.9, 2.
labels_ = np.array([1, 1, 0, 0, 0, 0])

ds_ = np.insert(sequences_, 2, labels_, axis=0)
df_ = pd.DataFrame(ds_.T, columns=['x1', 'x2', 'y'])
```

In [22]:
```
df_
```

Out[22]:

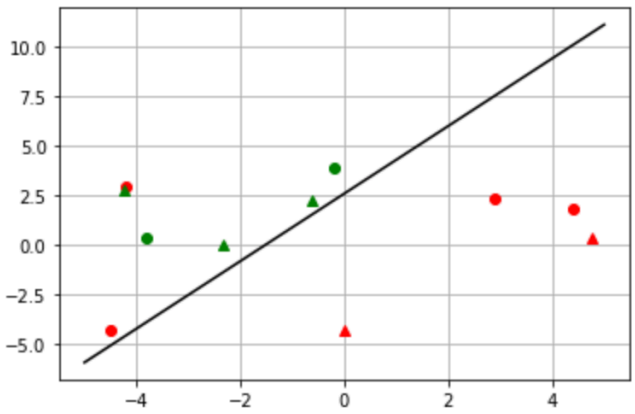
| | x1 | x2 | y |
|---|---|---|---|
| **0** | -3.8 | 0.4 | 1.0 |
| **1** | -0.2 | 3.9 | 1.0 |
| **2** | 2.9 | 2.3 | 0.0 |
| **3** | -4.5 | -4.3 | 0.0 |
| **4** | -4.2 | 2.9 | 0.0 |
| **5** | 4.4 | 1.8 | 0.0 |

In [23]:
```
model_ = Perceptron(epochs, lr)
weights_, bias_ = model_.Train(sequences_.T, labels_, False)
print('weights: {}, bias: {}'.format(weights_, bias_))
```

```
iter: 0 weights: [0.30801277 0.26387084] bias: 0.32383276483316237

iter: 3 weights: [-0.07198723  0.30387084] bias: 0.42383276483316235

iter: 7 weights: [-0.36198725  0.07387084] bias: 0.32383276483316237

iter: 10 weights: [0.08801277 0.50387087] bias: 0.22383276483316236

iter: 13 weights: [-0.20198726  0.27387087] bias: 0.12383276483316236

iter: 17 weights: [-0.49198728  0.04387087] bias: 0.02383276483316235

iter: 20 weights: [-0.04198726  0.4738709 ] bias: -0.07616723516683765

iter: 24 weights: [-0.33198728  0.2438709 ] bias: -0.17616723516683766

iter: 25 weights: [0.11801274 0.67387094] bias: -0.27616723516683767

iter: 28 weights: [-0.26198726  0.71387094] bias: -0.17616723516683766

iter: 32 weights: [-0.55198728  0.48387093] bias: -0.27616723516683767

iter: 35 weights: [-0.10198726  0.91387097] bias: -0.3761672351668377

iter: 40 weights: [-0.39198728  0.68387097] bias: -0.4761672351668377

iter: 41 weights: [0.0280127  0.39387095] bias: -0.5761672351668377

iter: 46 weights: [-0.35198729  0.43387095] bias: -0.4761672351668377

iter: 47 weights: [0.0680127  0.14387093] bias: -0.5761672351668377

weights: [-0.3119873   0.18387093], bias: -0.4761672351668377
```

Отобразим измененную обучающую выборку, случайно заданную выборку (треугольнички) и дискриминантную линию.

In [26]:
```
PlotFirst(df_)
area = np.linspace(-5, 5, 10)
line = DividingLine(area, weights_[0], weights_[1], bias_)
plt.plot(area, line, color='black')

for x, y, color in zip(randSeqs[:, 0], randSeqs[:, 1], pred_):
    plt.plot(x, y, "^", c=colors[color])
```



### Вторая обучающая выборка

In [11]:
```
sequences2 = np.array([[-0.6, -4.7, 2.1, -1.7, -1.8, 0.4, 0.5, -2.6], [4
labels2 = np.array([[1, 1, 0, 0, 0, 0, 0, 0], [1, 1, 0, 0, 1, 0, 0, 0]])

ds2 = np.insert(sequences2, 2, labels2, axis=0)
df2 = pd.DataFrame(ds2.T, columns=['x1', 'x2', 'y1', 'y2'])
```

In [12]:
```
df2
```

Out[12]:

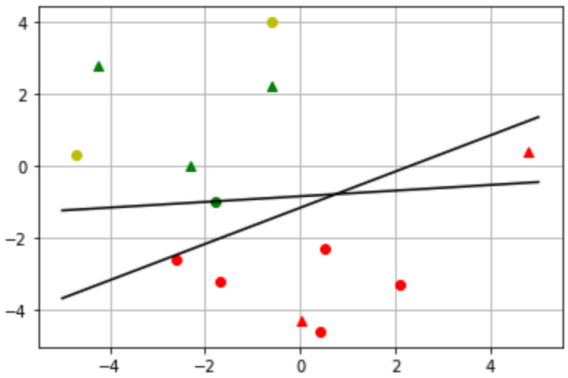
| | x1 | x2 | y1 | y2 |
|---|---|---|---|---|
| **0** | -0.6 | 4.0 | 1.0 | 1.0 |
| **1** | -4.7 | 0.3 | 1.0 | 1.0 |
| **2** | 2.1 | -3.3 | 0.0 | 0.0 |
| **3** | -1.7 | -3.2 | 0.0 | 0.0 |
| **4** | -1.8 | -1.0 | 0.0 | 1.0 |
| **5** | 0.4 | -4.6 | 0.0 | 0.0 |
| **6** | 0.5 | -2.3 | 0.0 | 0.0 |
| **7** | -2.6 | -2.6 | 0.0 | 0.0 |

In [28]:
```
model2 = Perceptron(epochs, lr)
weights2, bias2 = model2.Train(sequences2.T, labels2.T, True)
```

```
iter: 1 weights: [[0.08674343 0.41937221]
 [0.01591036 0.52776479]] bias: [0.86880146 0.33083925]

iter: 4 weights: [[ 0.08674343 -0.05062777]
 [ 0.01591036  0.5577648 ]] bias: [0.86880146 0.43083925]

iter: 9 weights: [[-0.12325656 -0.05062777]
 [ 0.34591036  0.5577648 ]] bias: [0.76880146 0.43083925]

iter: 14 weights: [[ 0.05674343 -0.23062777]
 [ 0.44591036  0.4577648 ]] bias: [0.66880146 0.53083925]

iter: 16 weights: [[ 0.23674343 -0.23062777]
 [ 0.54591036  0.4577648 ]] bias: [0.56880146 0.53083925]

iter: 21 weights: [[-0.23325655 -0.23062777]
 [ 0.57591036  0.4577648 ]] bias: [0.66880146 0.53083925]
```

Отобразим вторую обучающую выборку, случайно заданную выборку (треугольнички) и дискриминантную линию.

In [32]:
```
PlotSecond(df2)
area = np.linspace(-5, 5, 10)
line1 = DividingLine(area, weights2[0][0], weights2[1][0], bias2[0])
line2 = DividingLine(area, weights2[0][1], weights2[1][1], bias2[1])
plt.plot(area, line1, color='black')
plt.plot(area, line2, color='black')


for x, y, color in zip(randSeqs[:, 0], randSeqs[:, 1], pred):
    plt.plot(x, y, "^", c=colors[color])
```




Как видно на графике, моя вторая обучающая выборка сама по себе является линейно неразделимой.

# Код программы.

Данная программа может решать задачи, разрешимые однослойным перцептроном Розенблата. На линейно-неразделимых задачах программа не зацикливается благодаря счетчику итераций, если его значение превышает заданное количество эпох, то программа завершает работу.

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


class Perceptron:
    def __init__(self, epochs=25, lr=0.1, flag=True):
        self.epochs = epochs
        self.lr = lr
        self.weights = []
        self.bias = 0
        self.errors = []

    def Train(self, sequences, labels, flag=True):
        random.seed(7)
        if flag:
            self.weights = np.random.sample((2, 2))
            self.bias = np.random.sample(2)
        else:
            self.bias = random.random()
            self.weights = np.random.sample(2)

        iteration = 0
        i = 0
        while i < sequences.shape[0]:
            if iteration == self.epochs:
                break
            predictedLabel = self.Predict(sequences[i], self.weights, self.bias, flag)
            error = labels[i] - predictedLabel
            self.errors.append(abs(error))
            if (np.linalg.norm(error) != 0):
                if flag:
                    self.weights += self.lr * sequences[i].reshape(-1,
1).dot(error.reshape(1, -1))
                else:
                    self.weights += self.lr *  sequences[i] * error
                self.bias += self.lr * error
                i = -1
            i += 1
            iteration += 1
        return self.weights, self.bias

    def MAE(self):
        return sum(self.errors) / len(self.errors)

    def F(self, x, w, b):
        return np.dot(x, w) + b


    def Predict(self, seq, weights, bias, flag=True):
        if (flag):
            return np.heaviside(self.F(seq, weights, bias), 1)
        return 1 if self.F(seq, weights, bias) >= 0 else 0
```

# Выводы:

Выполнив лабораторную работу, я реализовала одну из первых моделей нейросетей - однослойный перцептрон Розенблатта. Обучила эту модель на заданных выборках и с помощью этой обученной модели предсказывала классы объектов. Также я выяснила, что однослойный перцептрон Розенблатта не подходит для решения линейно неразделимых задач.