Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 1
по курсу «Нейроинформатика»

Студент: Аксенов А. Е.
Группа: М80-408Б-20
Преподаватель: Горохов М. А.
Оценка:

Москва, 2023

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 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.

Оборудование

Процессор : Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
ОЗУ : 16 ГБ

Программное обеспечение

Python 3.8 + Jupyter Notebook

Сценарий выполнения 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.

1.1 Обучающее множество выглядит следующим образом. Слева признаки, справа метки.

$$\begin{bmatrix} 1.1 & -1.5 & 0.8 & 4.1 & 2.5 & -1.2 \\ -0.3 & 3.3 & 0.4 & -2.2 & 2.5 & 0.6 \end{bmatrix} \quad \Bigg| \quad \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

1.2 Сконфигурируем нейронную сеть. Веса задаются небольшими случайными числами. Сеть имеет один нейрон, 2 веса для признаков и 1 вес для константы, функция активации возвращает 1, когда вход больше 0, 0 когда вход меньше 0.

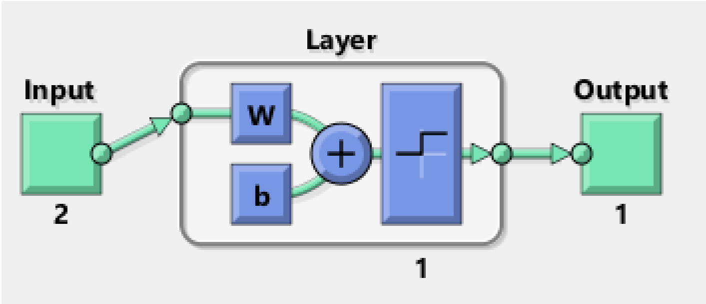

Рис. 1 Структура сети

1.3 Создадим алгоритм обучения по правилу Розенблата. Веса изменяется по следующему правилу

$$\mathbf{W}^* = \mathbf{W} + \mathbf{e}\mathbf{p}^T$$
$$\mathbf{b}^* = \mathbf{b} + \mathbf{e}$$

Рис. 2 Правило обновления весов

Здесь W* - обновленный вес, W- нынешний вес, e-ошибка, p – вектор признака объекта, b* - обновленная константа, b – нынешняя константа.

1.4 Обучим сеть по правилу Розенблата. Изобразим обучающее множество и разделяющие прямые.

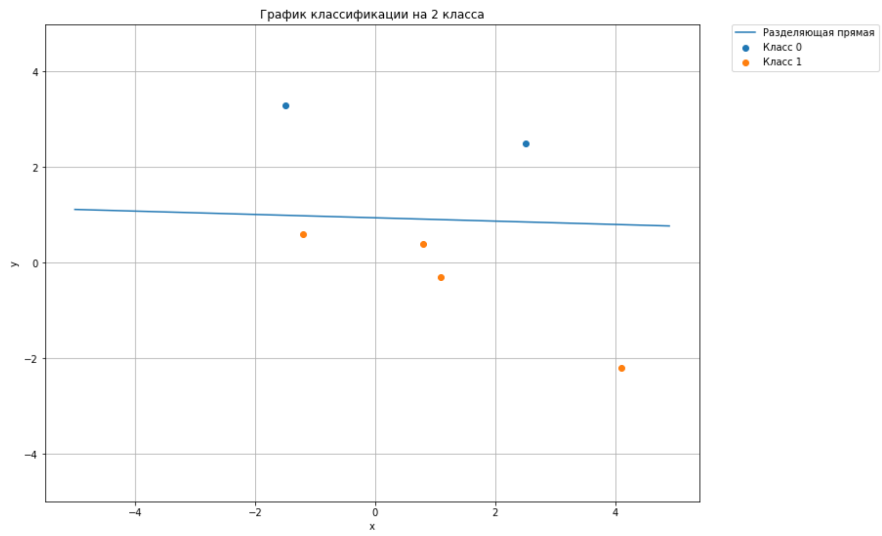

Рис. 4 График с разделяющей прямой и самими объектами

1.5 Добавим новые точки.

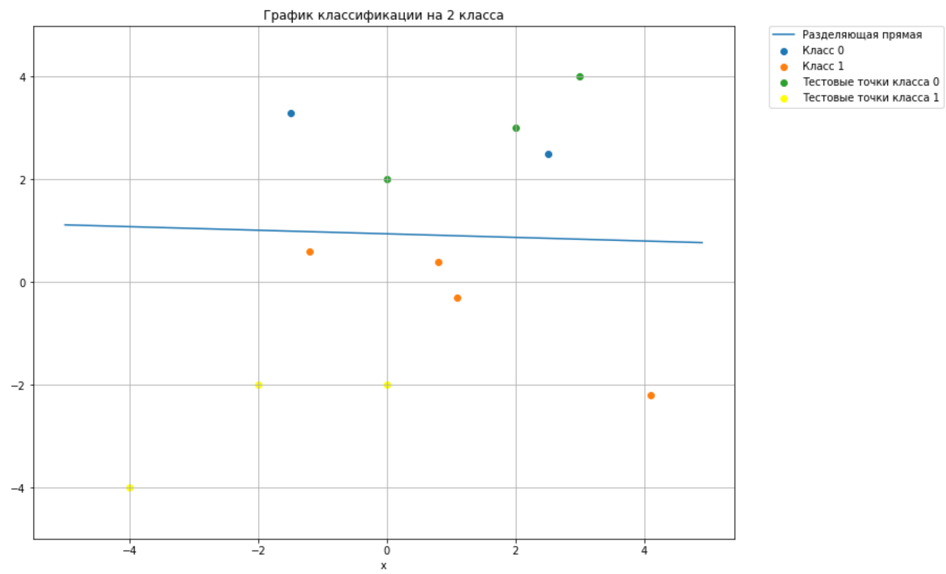

Рис. 5 График с разделяющей прямой и новыми добавленными точками

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

2.1 Давайте разберем более подробно работу персептрона на примере рассматриваемой модели, если классы являются линейно неразделимыми. По сути когда мы умножаем веса на вход и складываем результат, мы строем некоторую прямою в системе координат P1XP2, где P1 и P2 –входные признаки. Изобразить это можно так

Рис. 6 Построение разделяющей прямой

То есть результатом работы нашей модели является некоторая прямая, разделяющая классы. Но может ли прямая разделить линейно неразделимые классы? Ответ – Нет. Потому что в самом понятии линейно неразделимости множеств вкладывается свойство, что нельзя провести прямую, которая разделяет эти классы с точностью 100%. Приведем пример линейно неразделимого классов.

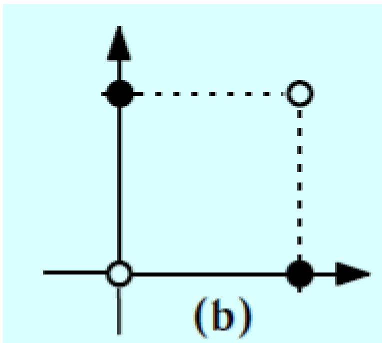

Рис. 7 Пример линейно неразделимого класса

Из Рис. 7 следует, что не существует способа прямой разделить эти классы с точностью 100%. Так как персептрон строит разделяющую прямую, то не сможет решить такую задачу с точностью 100%. Продемонстрируем данный факт на другом множестве в следующем пункте.

2.2 Добавим 1 точку к изначальному обучающему множеству, чтоб классы были линейно неразделимы. Когда персептрон классифицирует множество, то проводятся разделяющие прямые между классами. Так вот мы не можем провести прямую между 2-мя линейно неразделимыми множествами, что и будет продемонстрировано дальше.

2.3 Попробуем дообучить модель на новом множестве. Построим график с новой разделяющей прямой.

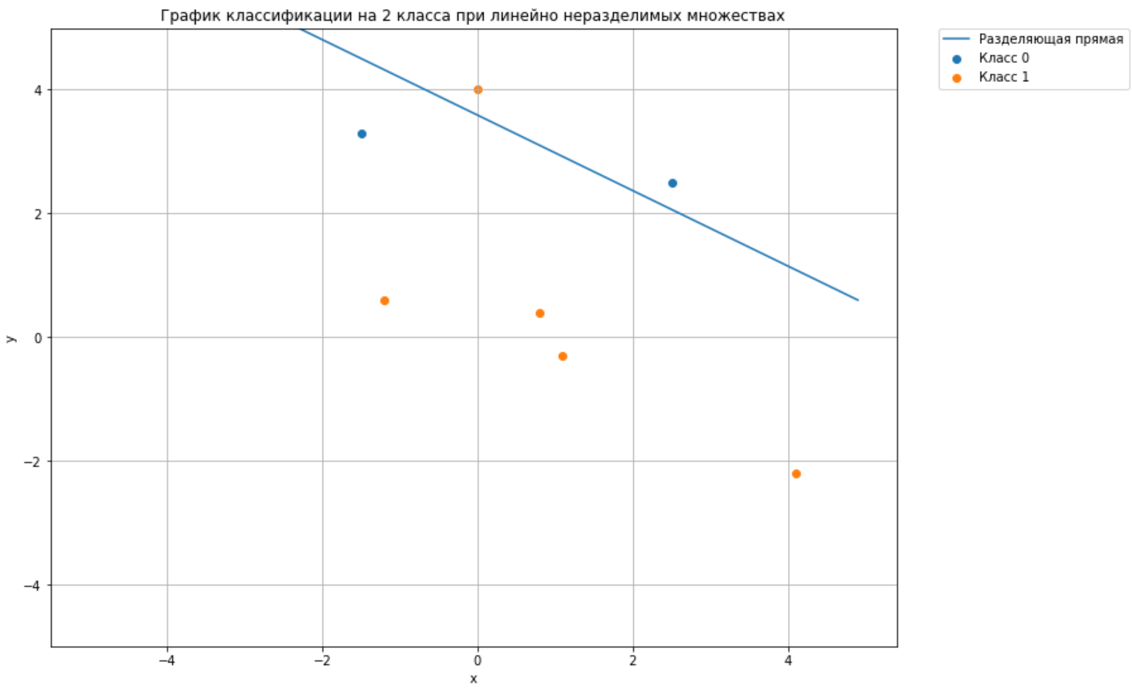

Рис. 9 График с разделяющей прямой и самими объектами после дообучения

Модель не смогла справиться эффективно с линейно неразделим множеством.

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

3.1 Обучающее множество выглядит следующим образом. Слева признаки, справа метки.

$$\begin{bmatrix} 3.6 & -1.5 & -2.8 & 1 & -3.6 & -0.8 & 2.2 & 3.4 \\ 1.3 & 4.9 & 1.5 & -1.2 & -4.8 & -3.2 & -1.3 & 2.3 \end{bmatrix} \Bigg| \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3.2 Сконфигурируем нейронную сеть. Веса задаются небольшими случайными числами. Сеть имеет два нейрон, 4 веса для признаков и 2 веса для констант, функция активации возвращает 1, когда вход больше 0, 0 когда вход меньше 0.

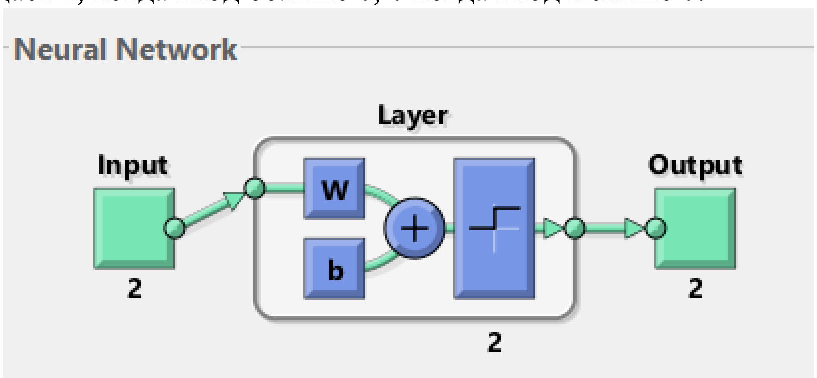

Рис. 10 Структура сети

3.3 Обучим сеть.. Нарисуем множество объектов с разделяющими прямыми.

Рис. 11 Доля верный ответов для задачи с 4 классами

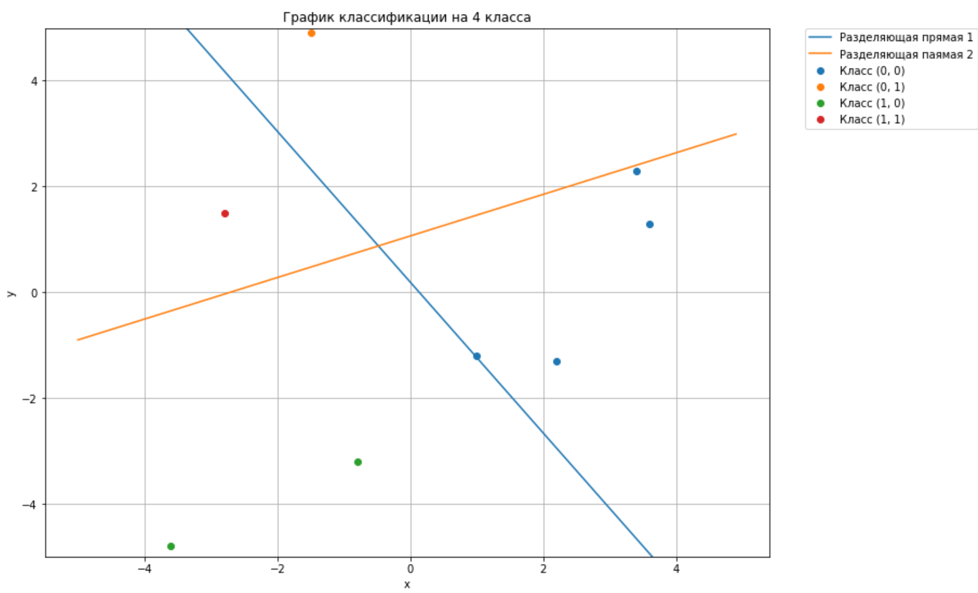

Рис. 12 График с разделяющими прямыми и самими объектами для задачи с 4 классами

Код программы

```
#!/usr/bin/env python
# coding: utf-8

# # Лабораторная работа 1
# ## Персептрон

# In[1]:


import numpy as np
import matplotlib.pyplot as plt


# ## Класс персептрона для классификации на 2 класса

# In[2]:


class Perceptron:
    # Инициализация
    def __init__(self, in_size, learning_rate):
        self.w = np.random.randn(in_size + 1) / np.sqrt(in_size)
        self.learning_rate = learning_rate

    # Функция ошибки
    def loss(self, x):
        return 1 if x > 0 else 0
    # Обучение
    def fit(self, X, y, epochs=10):
        X = np.c_[X, np.ones((X.shape[0]))]
        for epoch in np.arange(0, epochs):
            for (x, target) in zip(X, y):
                p = self.loss(np.dot(x, self.w))
                if p != target:
                    error = (p - target)
                    self.w -= self.learning_rate * error * x
            print(f'Epoch: {epoch+1}, error: {error}, weights: {self.w}')
    # Предсказание
    def predict(self, X):
        X = np.atleast_2d(X)
        X = np.c_[X, np.ones((X.shape[0]))]
        return self.loss(np.dot(X, self.w))
    # Получить веса
    def get_weights(self):
        return self.w


# ## Обучающая выборка 1

# In[3]:


points1 = np.array([
    [1.1, -0.3],
    [-1.5, 3.3],
    [0.8, 0.4],
    [4.1, -2.2],
    [2.5, 2.5],
    [-1.2, 0.6],
], dtype=np.float64)
```

```
labels1 = np.array([1, 0, 1, 1, 0, 1], dtype=np.int32)


# In[4]:


perceptron = Perceptron(2, 0.1)
perceptron.fit(points1, labels1)
# Достатоно быстро веса перестают изменяться
# Это вызвано скорее всего тем, что быстро достигается приемлемое качество
weights = perceptron.get_weights()


# In[5]:


# Коорд для прямой
x = np.arange(-5, 5, 0.1)
y = np.apply_along_axis(lambda t: (-weights[2] - t * weights[0]) /
weights[1], 0, x)

# Коорд для точек
x_c1 = points1[labels1 == 0, 0]
y_c1 = points1[labels1 == 0, 1]

x_c2 = points1[labels1 == 1, 0]
y_c2 = points1[labels1 == 1, 1]


# Строим график
plt.figure(figsize=(12, 9))
plt.ylim(-5, 5)
plt.title('График классификации на 2 класса')
line = plt.plot(x, y, label='Разделяющая прямая')
c1 = plt.scatter(x_c1, y_c1, label='Класс 0')
c2 = plt.scatter(x_c2, y_c2, label='Класс 1')

plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.show()


# In[6]:


# Коорд для прямой
x = np.arange(-5, 5, 0.1)
y = np.apply_along_axis(lambda t: (-weights[2] - t * weights[0]) /
weights[1], 0, x)

# Коорд для точек
x_c1 = points1[labels1 == 0, 0]
y_c1 = points1[labels1 == 0, 1]

x_c2 = points1[labels1 == 1, 0]
y_c2 = points1[labels1 == 1, 1]


# Строим график
plt.figure(figsize=(12, 9))
plt.ylim(-5, 5)
plt.title('График классификации на 2 класса')
```

```
line = plt.plot(x, y, label='Разделяющая прямая')
c1 = plt.scatter(x_c1, y_c1, label='Класс 0')
c2 = plt.scatter(x_c2, y_c2, label='Класс 1')
plt.scatter([0,2,3],[2,3,4],label='Тестовые точки класса 0')
plt.scatter([-4,-2,0],[-4,-2,-2],label='Тестовые точки класса
1',color='yellow')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.show()


# ##### Задача решена хорошо, потому что множества являются линейно
разделимыми
#

# ### Добавление точки для получения двух линейно неразделимых множеств

# In[7]:


# Добавляем точку, чтоб классы не был линейно разделимы
points1 = np.concatenate([points1, np.array([[0, 4]])])
labels1 = np.concatenate([labels1, np.array([1])])


# In[8]:


# Создаем модель
perceptron = Perceptron(2, 0.1)
# Обучаем
perceptron.fit(points1, labels1)

weights = perceptron.get_weights()
# Коорд для прямой
x = np.arange(-5, 5, 0.1)
y = np.apply_along_axis(lambda t: (-weights[2] - t * weights[0]) /
weights[1], 0, x)

# Коорд для точек
x_c1 = points1[labels1 == 0, 0]
y_c1 = points1[labels1 == 0, 1]

x_c2 = points1[labels1 == 1, 0]
y_c2 = points1[labels1 == 1, 1]

# Строим график
plt.figure(figsize=(12, 9))
plt.ylim(-5, 5)
plt.title('График классификации на 2 класса при линейно неразделимых
множествах')
plt.plot(x, y, label='Разделяющая прямая')
plt.scatter(x_c1, y_c1, label='Класс 0')
plt.scatter(x_c2, y_c2, label='Класс 1')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.show()


# Как видно из графика, после добавления дополнительной точки, множества
```

```
стали линейно неразделимы и поэтому персептрон, не смог разделить точки на 2
класса

# ## Класс персептрона для классификации на 4 класса

# In[9]:


class Perceptron:
    # Инициализация
    def __init__(self, in_size, out_size, learning_rate):
        self.w = np.random.randn(out_size, in_size + 1) / np.sqrt(in_size)
        self.learning_rate = learning_rate
    # Функция потерь
    def loss(self, x):
        return np.vectorize(lambda t: 1 if t > 0 else 0)(x)
    # Обучение
    def fit(self, X, y, epochs=10):
        X = np.c_[X, np.ones((X.shape[0]))]
        for epoch in np.arange(0, epochs):
            for (x, target) in zip(X, y):
                p = self.loss(np.dot(x, self.w.T))
                if not np.array_equal(p, target):
                    error = np.reshape(p - target, (2, 1))
                    self.w -= self.learning_rate * error * x
            print(f'Epoch: {epoch+1}, error: {error.T}, weights: {self.w}')
    # Предсказание
    def predict(self, X):
        X = np.atleast_2d(X)
        X = np.c_[X, np.ones((X.shape[0]))]
        return self.loss(np.dot(X, self.w.T))
    # Получить веса
    def get_weights(self):
        return self.w


# In[10]:


# Выборка
points2 = np.array([
    [3.6, 1.3],
    [-1.5, 4.9],
    [-2.8, 1.5],
    [1, -1.2],
    [-3.6, -4.8],
    [-0.8, -3.2],
    [2.2, -1.3],
    [3.4, 2.3]
], dtype=np.float64)
labels2 = np.array([
    [0, 0],
    [0, 1],
    [1, 1],
    [0, 0],
    [1, 0],
    [1, 0],
    [0, 0],
    [0, 0]
], dtype=np.int32)


# In[11]:
```

```python
# Создаем модель
perceptron = Perceptron(2, 2, 0.1)
# Обучаем
perceptron.fit(points2, labels2,epochs=100)
# Берем веса
weights = perceptron.get_weights()


# In[12]:


# Функция для работы с точками
def partition_on_class(points, labels, mask):
    res = []
    for i in np.arange(points.shape[0]):
        if np.array_equal(labels[i], mask):
            res.append(points[i])
    return np.array(res)


# In[13]:


# Коорд прямых
x = np.arange(-5, 5, 0.1)

y1 = np.vectorize(lambda t: (-weights[0][2] - t * weights[0][0]) /
weights[0][1])(x)
y2 = np.vectorize(lambda t: (-weights[1][2] - t * weights[1][0]) /
weights[1][1])(x)

# Точки классов
points_c1 = partition_on_class(points2, labels2, [0, 0])
points_c2 = partition_on_class(points2, labels2, [0, 1])
points_c3 = partition_on_class(points2, labels2, [1, 0])
points_c4 = partition_on_class(points2, labels2, [1, 1])

# Строим график
plt.figure(figsize=(12, 9))
plt.ylim(-5, 5)
plt.title('График классификации на 4 класса')
plt.plot(x, y1, label='Разделяющая прямая 1')
plt.plot(x, y2, label='Разделяющая паямая 2')
plt.scatter(points_c1[:,0], points_c1[:,1], label='Класс (0, 0)')
plt.scatter(points_c2[:,0], points_c2[:,1], label='Класс (0, 1)')
plt.scatter(points_c3[:,0], points_c3[:,1], label='Класс (1, 0)')
plt.scatter(points_c4[:,0], points_c4[:,1], label='Класс (1, 1)')

plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.show()


# ##### Задача решена хорошо, потому что множества являются линейно
разделимыми. В задаче 4 класса, потому 2 прямые разделяют выборку.
#

# In[ ]:
```

Выводы

Выполнил лабораторную работу я познакомился с архитектурой полно связной нейронной сети – персептрон, реализовал алгоритм обновления весов Розенблатта. Из плюсов данной модели хочется выделить лёгкость обучения, возможность решать простые задачи с линейно разделимыми множествами. Из минусов сразу бросается в глаза проблемы с применением методов оптимизации на основе градиентов из-за ступенчатой функции, невозможность эффективно справляться с задачами классификации линейно неразделимых множеств. Что касается мультиклассовой классификации, с этой задачей персептрон справился благодаря увеличению количества выходов. Тут работает простое правило: S выходов, принимающих значения {0, 1}, может разделять входные векторы на 2^S классов.