МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра 806: «Вычислительная математика и программирование»

Лабораторная работа № 1
по курсу «Нейроинформатика»

Студент: А. О. Тояков
Преподаватель: Н. П. Аносова
Группа: М8о-407б-18
Дата:
Оценка:
Подпись:

Москва, 2021

ПЕРСЕПТРОНЫ. ПРОЦЕДУРА ОБУЧЕНИЯ РОЗЕНБЛАТТА

Цель работы: 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 четырём классам. Отобразить дискриминантную линию и проверить качество обучения.

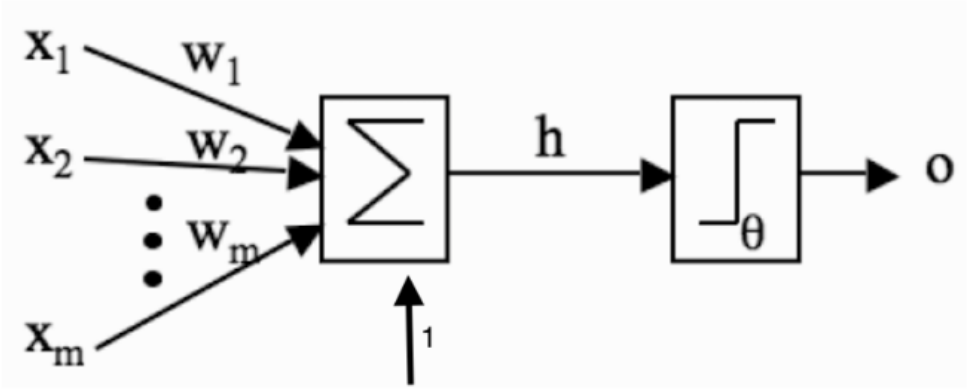
Вариант № 26

Обучающие данные:

1. [(-3.8, -4.8), (-2.1, -1,9), (1.5, 4.5), (4.3, -0.5), (-2.6, 2.6), (2.5, 2.4)]
   [1, 1, 1, 0, 1, 0]
2. [(-4, -3.6), (-3.4, 1.2), (0.7, -4.5), (4.3, 2.2), (2.3, -4.4), (3.6, 4.3), (4.8, 3.5), (2.8, 0.1)]
   [(0, 1), (0, 1), (0, 0), (1, 0), (0, 0), (1, 1), (1, 0), (0, 0)]

ХОД РАБОТЫ

Для решения этой задачи необходимо воспользоваться Перцептроном Розенблата, который имеет следующую структуру:



Чтобы реализовать слой таких перцептронов можно воспользоваться представлением весов и смещений перцептронов как марицу $(n+1) \times m$, где n - число входов, а m - число выходов. При этом в качестве выходов я использую функцию $net = \sum_{i=0}^{n} w_i x_i + b$, а ошибку измеряюс помощи метрики $MAE = \frac{\sum_{i=1}^{N} |t_i - a_i|}{N}$. Для ужобства классификации и обучения, я заеняю метки негативных классов с 0 на -1.

Реализация слоя из перцептронов Розенблата:

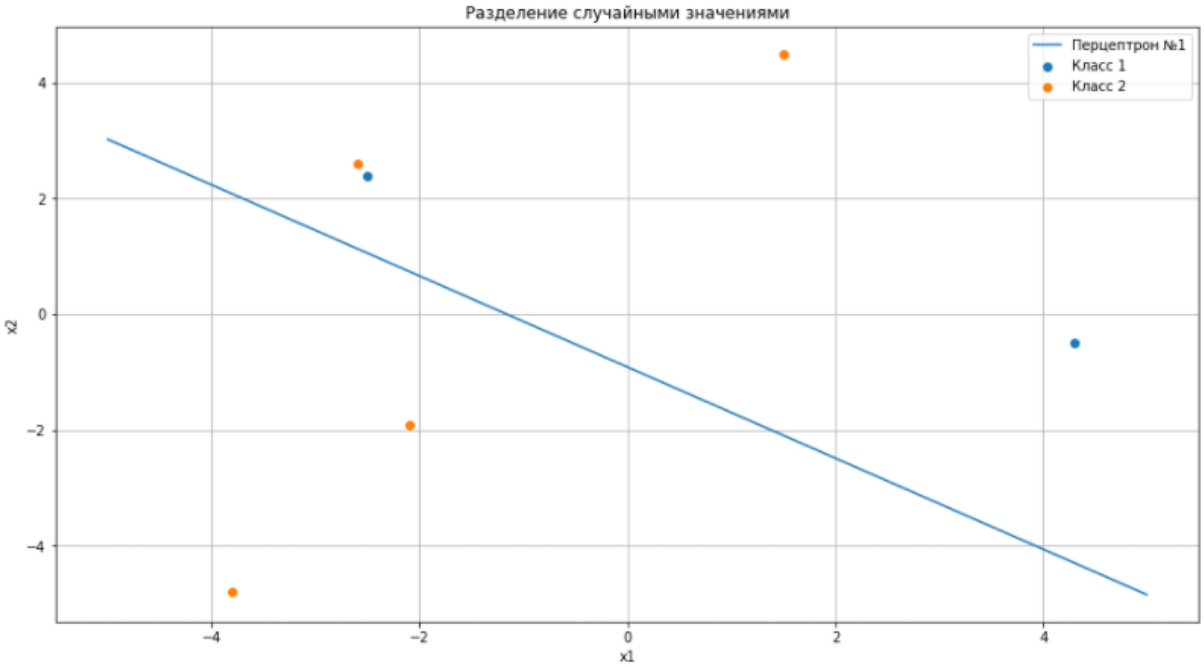
```
class RosenblattLayer:
    def __init__(self, steps = 50, early_stop = False):
        self.steps = steps
        self.w = None
        self.negative_is_zero = False
        self.early_stop = early_stop

    def fit(self, X, y):
        # add column for bias and transpose data for comphort operations:
        X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
        y_t = np.array(y.T)

        if (y_t == 0).sum():
            self.negative_is_zero = True
            y_t[y_t == 0] = -1

        #init weights
        if self.w is None:
            self.w = np.random.random((X_t.shape[1], y_t.shape[1]))

        # main loop

        for step in tqdm(range(self.steps)):
            stop = True
            for i in range(X_t.shape[0]):
                # compute error of all perceptrons
                predict = X_t[i].dot(self.w)
                if np.sum(predict*y_t[i] < 0):
                    stop = False
                    e = y_t[i] - X_t[i].dot(self.w)
                    e[predict*y_t[i] >= 0] = 0.0
                    self.w += X_t[i].reshape(X_t.shape[1], 1).dot(e.reshape(1, y_t.shape[1]))
            if stop and self.early_stop:
```

```
                break

        return self

    def set_steps(self, steps):
        self.steps = steps

    def set_early_stop(self, stop):
        self.early_stop = stop

    # Predict answers
    def predict(self, X):
        X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
        return X_t.dot(self.w).T

    def predict_classes(self, X):
        X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
        ans = X_t.dot(self.w)
        a_t =  ans < 0
        if self.negative_is_zero:
            ans[a_t] = 0
        else:
            ans[a_t] = -1
        ans[np.logical_not(a_t)] = 1
        return ans.T

    def display(self):
        ans = " Input(n," + str(self.w.shape[0] - 1) + ")  -->  "
        ans += "Rosenblat Perceptrons(" + str(self.w.shape[1]) + ")  -->  "
        ans += "Output(n, " + str(self.w.shape[1]) + ")"
        return ans

    def weights(self):
        return self.w[:-1]

    def bias(self):
        return self.w[-1]

    def score(self, X, y):
        X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
        y_t = np.array(y.T)
        y_t[y_t == 0] = -1
        return np.abs(y_t - X_t.dot(self.w)).mean()
```

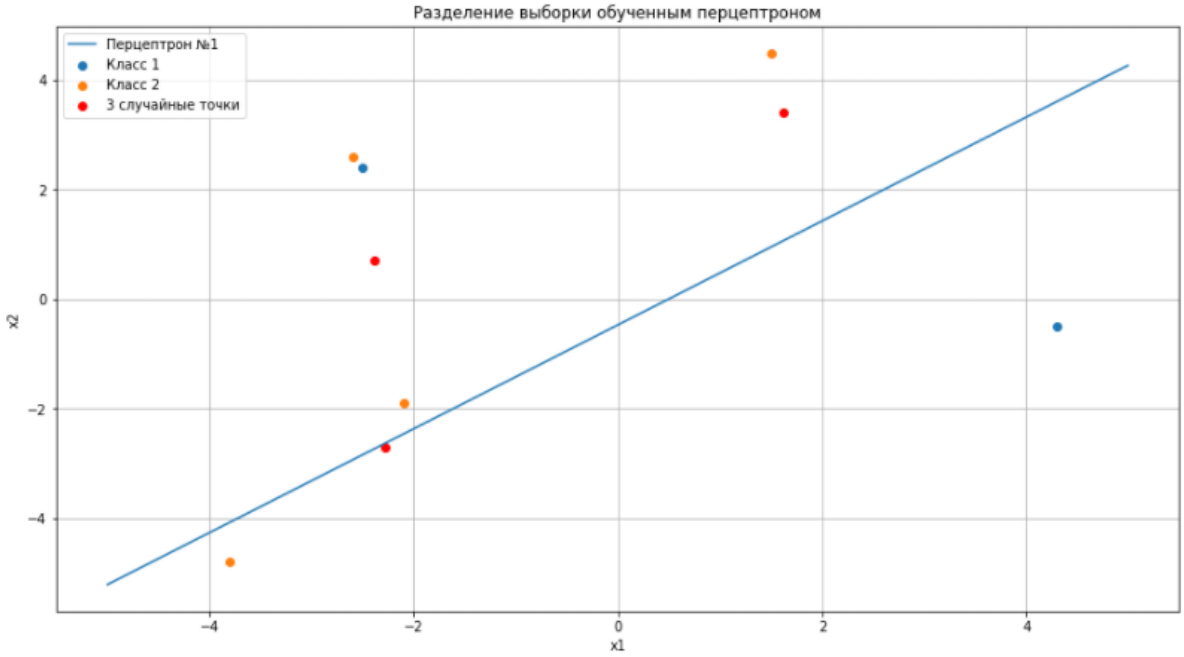
После этого я создал модель со случайными коэффициентами для первых обучающих данных и вывел её на экран.

```
P = np.array([
    [-3.8, -2.1, 1.5, 4.3, -2.6, -2.5],
    [-4.8, -1.9, 4.5,-0.5, 2.6, 2.4]
])

T = np.array([[1, 1, 1, 0, 1, 0]])
```

Вот так модель со случайными коэффициентами разделяет выборку:



После чего я обучил модель и построил разделяющую прямую, добавив 3 случайные точки и классифицировал их:



Полученная модель научилась классифицировать данные:

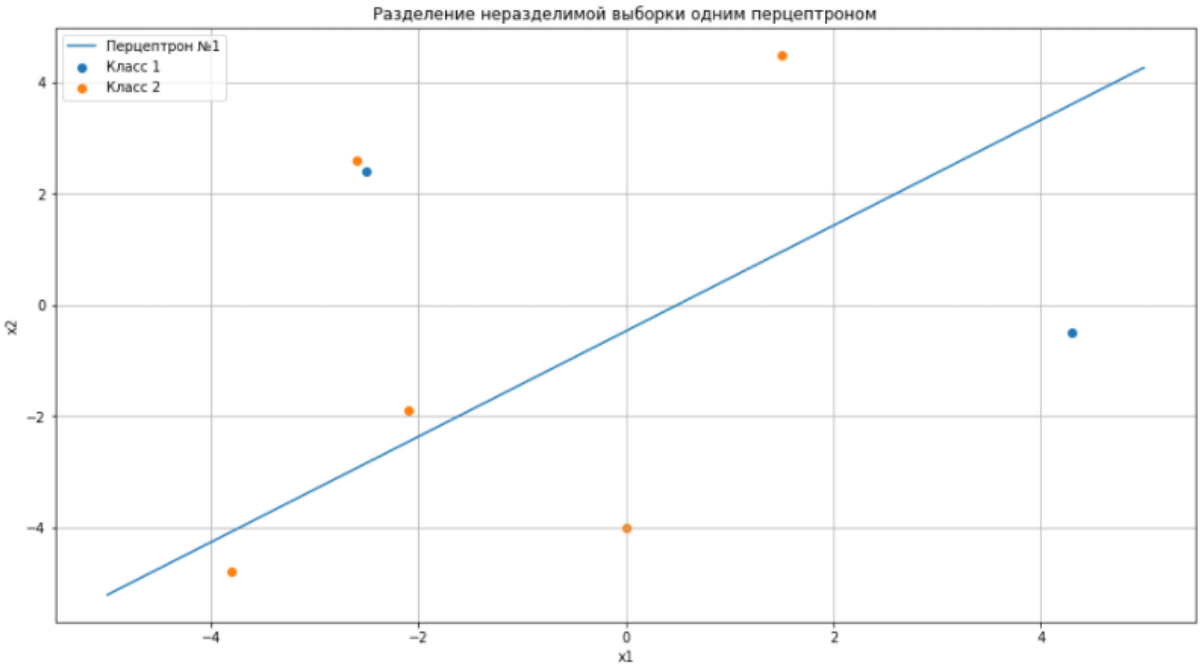
```
model.predict_classes(P)
array([[1., 0., 0., 1., 0., 0.]])
```

Веса модели:

```
print("Веса:")
print(model.weights())
print("Смещения:")
```

```
print(model.bias())
Веса:
[[ 4.91741280e+158]
 [-5.18252304e+158]]
Смещения:
[-2.40652016e+158]
```

Добавив в датасет точку так, что выборка стала линейно неразделима, обнаружилось, что даже после нескольких эпох обучения модель всё не научилась точно классифицировать данные:

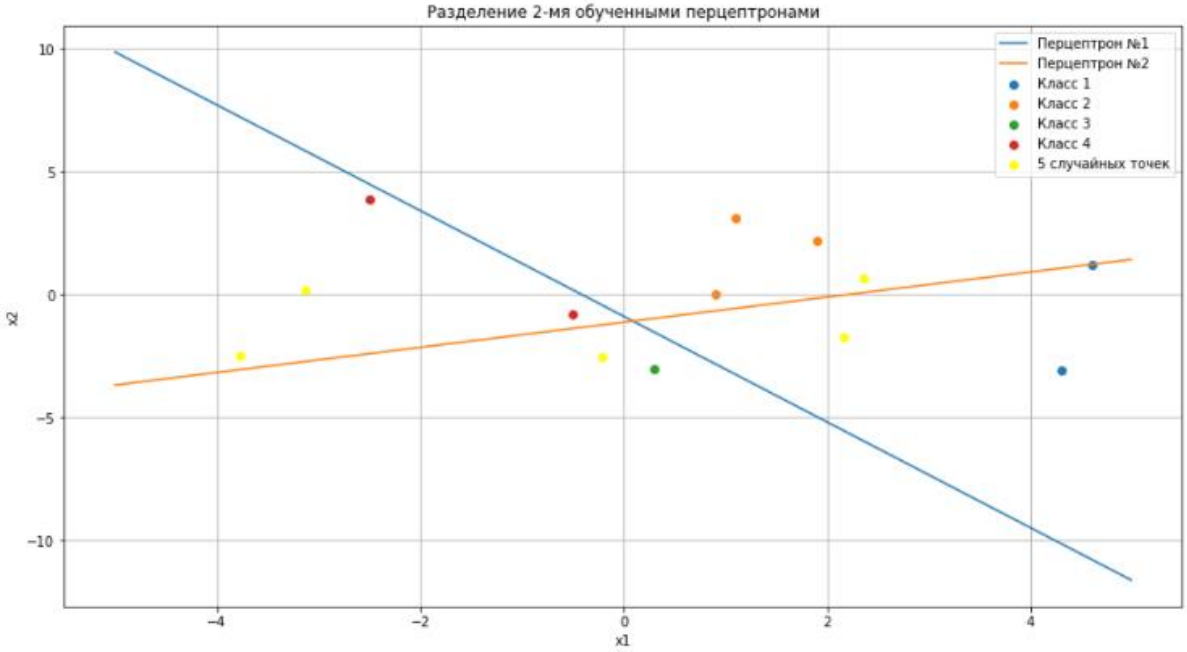


Для того, чтобы классифицировать данные на несколько классов, недостаточно одного перцептрона, а нужен целый слой.

Создание и обучение сети с классификацией 5 случайных точек:

```
P = np.array([
    [-4.0, -3.4, 0.7, 4.3, 2.3, 3.6, 4.8, 2.8],
    [-3.6, 1.2, -4.5, 2.2, -4.4, 4.3, 3.5, 0.1]
])

T = np.array([
    [0, 0, 0, 1, 0, 1, 1, 0],
    [1, 1, 0, 0, 0, 1, 0, 0]
])
```

```
model = RosenblattLayer(0).fit(P, T)
print(model.display())
model.set_steps(50)
model.set_early_stop(True)
model.fit(P, T)
```



Веса модели:

```
print("Веса:")
print(model.weights())
print("Смещения:")
print(model.bias())
Веса:
[[-8.77308336e+091 -6.67181680e+221]
 [ 1.02610626e+091 -4.78576181e+221]]
Смещения:
[-4.07416642e+091 -1.37963142e+221]
```

ВЫВОД

Выполнив первую лабораторную работу, я познакомился перцептроном Розенблатта, который не имеет прикладного значения, но имеет историческое. Хоть реализация его проста, у него есть несколько недостатков. Во-первых, плохая сходимость, а во-вторых, невозможность классификации на неразделяемых данных. От них можно избавиться, используя более передовые архитектуры нейронных сетей, но это будет уже в следующих лабораторных.