Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Нейроинформатика»

Студент: М. А. Бронников
Преподаватель: Н. П. Аносова
Группа: М8О-407Б

Дата:
Оценка:
Подпись:

Москва, 2020

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 четырем классам. Отобразить дискриминантную линию и проверить качество обучения.

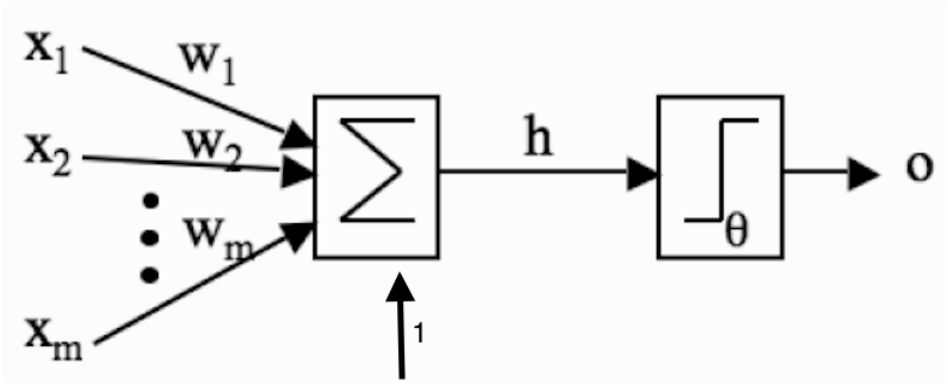
Вариант №4:
Обучающие данные:

1. $[(4, 3.6), (3.4, 1.2), (0.7, -4.5), (4.3, 2.2), (2.3, -4.4), (3.6, 4.3)]$
   $[0, 1, 0, 0, 0, 1]$

2. $[(4.3, -3.1), (-2.5, 3, 9), (0.9, 0), (1.1, 3.1), (0.3, -3), (-0.5, -0.8), (4.6, 1.2), (1.9, 2.2)]$
   $[(0, 0), (1, 1), (0, 1), (0, 1), (1, 0), (1, 1), (0, 0), (0, 1)]$

1 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:

```
1  class RosenblattLayer:
2      def __init__(self, steps = 50, early_stop = False):
3          self.steps = steps
4          self.w = None
5          self.negative_is_zero = False
6          self.early_stop = early_stop
7
8      def fit(self, X, y):
9          # add column for bias and transpose data for comphort operations:
10         X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
11         y_t = np.array(y.T)
12
13         if (y_t == 0).sum():
14             self.negative_is_zero = True
15             y_t[y_t == 0] = -1
16
17         #init weights
```

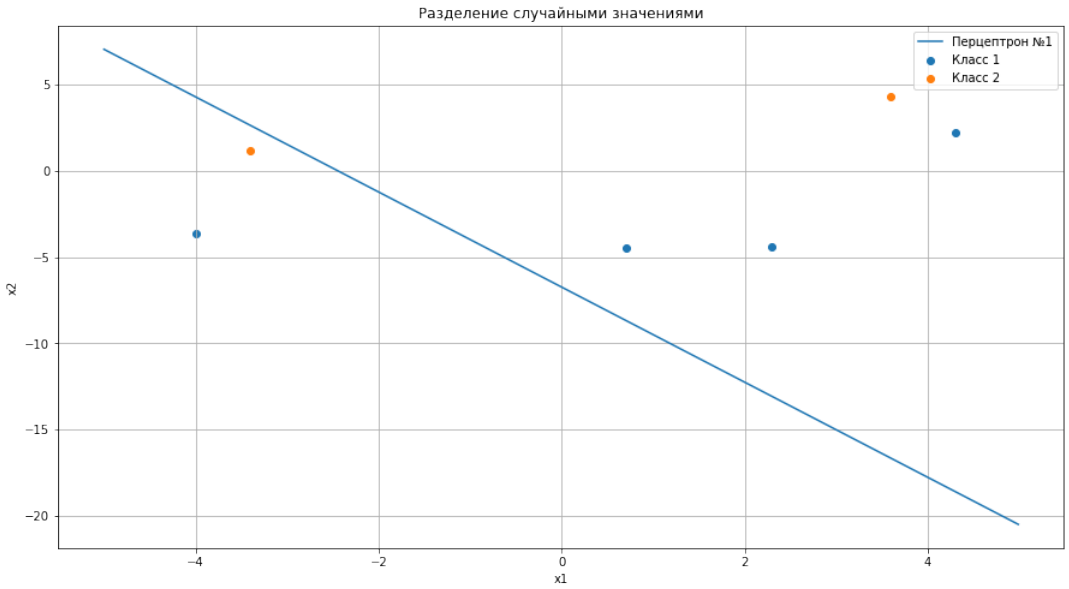
```
18          if self.w is None:
19              self.w = np.random.random((X_t.shape[1], y_t.shape[1]))
20
21          # main loop
22
23          for step in tqdm(range(self.steps)):
24              stop = True
25              for i in range(X_t.shape[0]):
26                  # compute error of all perceptrons
27                  predict = X_t[i].dot(self.w)
28                  if np.sum(predict*y_t[i] < 0):
29                      stop = False
30                      e = y_t[i] - X_t[i].dot(self.w)
31                      e[predict*y_t[i] >= 0] = 0.0
32                      self.w += X_t[i].reshape(X_t.shape[1], 1).dot(e.reshape(1, y_t.shape
                            [1]))
33              if stop and self.early_stop:
34                  break
35
36          return self
37
38      def set_steps(self, steps):
39          self.steps = steps
40
41      def set_early_stop(self, stop):
42          self.early_stop = stop
43
44      # Predict answers
45      def predict(self, X):
46          X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
47          return X_t.dot(self.w).T
48
49      def predict_classes(self, X):
50          X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
51          ans = X_t.dot(self.w)
52          a_t = ans < 0
53          if self.negative_is_zero:
54              ans[a_t] = 0
55          else:
56              ans[a_t] = -1
57          ans[np.logical_not(a_t)] = 1
58          return ans.T
59
60      def display(self):
61          ans = " Input(n," + str(self.w.shape[0] - 1) + ") --> "
62          ans += "Rosenblat Perceptrons(" + str(self.w.shape[1]) + ") --> "
63          ans += "Output(n, " + str(self.w.shape[1]) + ")"
64          return ans
65
```

```
66      def weights(self):
67          return self.w[:-1]
68
69      def bias(self):
70          return self.w[-1]
71
72      # MAE
73      def score(self, X, y):
74          X_t = np.append(X, np.ones((1, X.shape[1])), axis = 0).T
75          y_t = np.array(y.T)
76          y_t[y_t == 0] = -1
77          return np.abs(y_t - X_t.dot(self.w)).mean()
```

Далее я создал модель со случайными коэффицентами для первых обучающих данных и вывел её схему на экран:

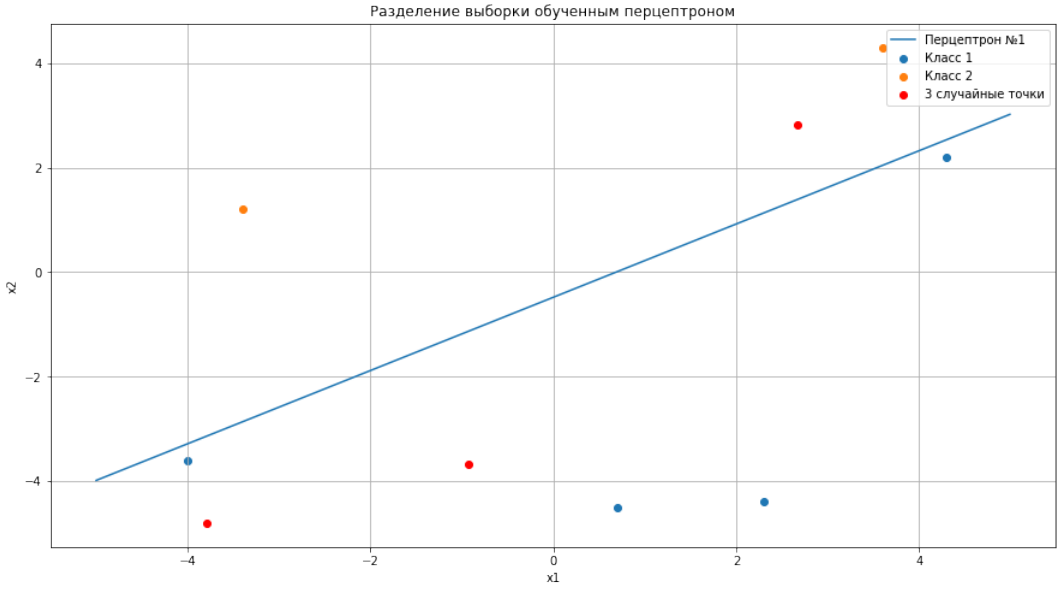
```
1  P = np.array([
2      [-4, -3.4, 0.7, 4.3, 2.3, 3.6],
3      [-3.6, 1.2, -4.5, 2.2, -4.4, 4.3]
4  ])
5  T = np.array([[0, 1, 0, 0, 0, 1]])
6
7  model = RosenblattLayer(0).fit(P, T)
8  print(model.display())
9  >>> Input(n,2) --> Rosenblat Perceptrons(1) --> Output(n, 1)
```

Модель со случайными коэффицентами разделяет выборку следующим образом:



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

```
model.set_steps(50)
model.set_early_stop(True)
model.fit(P, T)
x = np.random.randint(-5, 4, 3) + np.random.random(3)
y = np.random.randint(-5, 4, 3) + np.random.random(3)
test = np.append(x.reshape(1, 3), y.reshape(1, 3), axis=0)
print("Test Data:")
print(test)
print("Predict:")
print(model.predict_classes(test))
>>> Test Data:
[[-0.92833974 -3.79022874 2.66960153]
 [-3.67138285 -4.80837051 2.81960205]]
Predict:
[[0. 0. 1.]]
```



Полученная модель обучилась классифицировать данные:

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

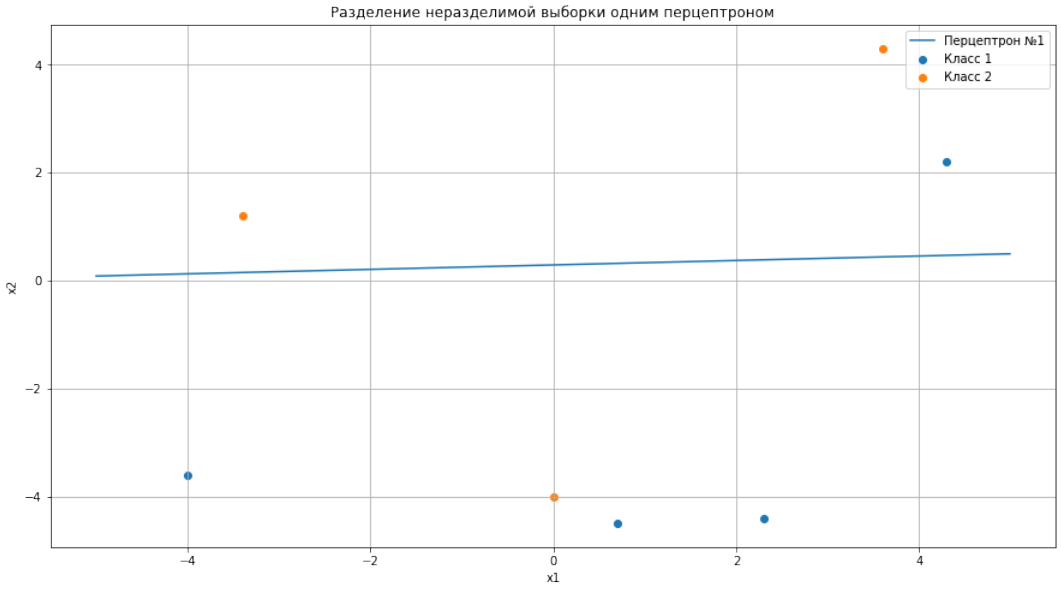
Веса модели:

```
print("Weights:")
print(model.weights())
print("Biases:")
print(model.bias())
>>> Weights:
```

```
[[-5.61599277e+17]
 [ 8.00796233e+17]]
Biases:
[3.857279e+17]
```

Добавив в датасет точку так, что выборка стала линейно неразделима, я увидел что даже после нескольких циклов обучения модель все еще не научилась разделять данные, выдавая ужасный результат классификации:

```
P = np.append(P, np.array([[0], [-4]]), axis=1)
T = np.append(T, np.array([[1]]), axis=1)
model = RosenblattLayer(50, True)
model.fit(P, T)
model.predict_classes(P) == T
>>> array([[False, False, False, True, False, False, True]])
```



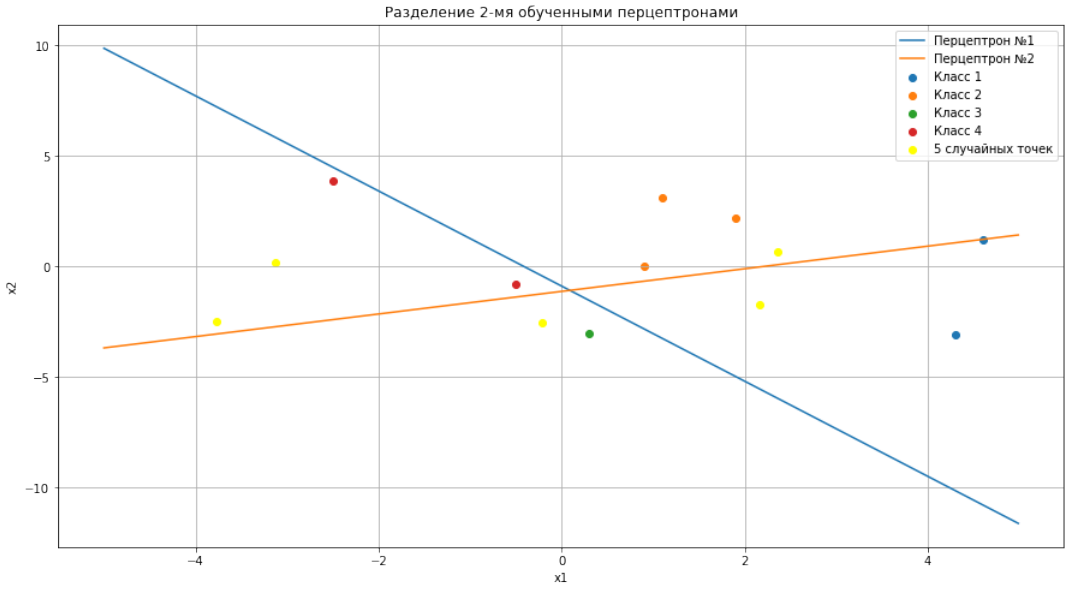
Для того, чтобы классифицировать данные на несколько классов, недостаточно одного перцептрона, а необходим целый слой. Так как мой класс реализует целый слой, интерфейс работы никак не поменялся.

Создание и обучение сети с последующей классификацией 5 случайных точек:

```
P = np.array([
    [4.3, -2.5, 0.9, 1.1, 0.3, -0.5, 4.6, 1.9],
    [-3.1, 3.9, 0.0, 3.1, -3, -0.8, 1.2, 2.2]
])

T = np.array([
```

```
    [0, 1, 0, 0, 1, 1, 0, 0],
    [0, 1, 1, 1, 0, 1, 0, 1]
])
model = RosenblattLayer(0).fit(P, T)
model.set_steps(50)
model.set_early_stop(True)
model.fit(P, T)
print(model.display())
>>> Input(n,2) --> Rosenblat Perceptrons(2) --> Output(n, 2)
```



Веса модели:

```
print("Weights:")
print(model.weights())
print("Biases:")
print(model.bias())
>>> Weights:
[[-1.70146535e+37 -8.52978957e+04]
 [-7.90898753e+36 1.66812729e+05]]
Biases:
[-6.97992809e+36 1.87414081e+05]
```

2 Выводы

Выполнив первую лабораторную работу по курсу «Нейроинформатика», я узнал о перцептроне Розенблата, который имеет историческое значение для всей науки.

Перцептрон Розенблата несмотря на свою простоту и умение классифицировать объекты, имеет ряд очевидных недостатков:

- Неозможность классификации на неразделяемых данных. (Показано в отчете)
- Плохая сходимость и неустойчивость. (Один раз метод не сошелся при выполнении работы на второй обучающей выборки)

Эти недостатки решают более современные архитектуры, с которыми мне предстоит познакомиться.

Эта работа была одной из самых простых для выполнения, однако это только вводная работа, и в будущем я ожидаю более трудных и интересных задач.