МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Кафедра вычислительной математики и программирования

Лабораторная работа №1
по спецкурсу «Нейроинформатика»

Персептроны. Процедура обучения Розенблатта

Выполнил: Днепров И.С.
Группа: М8О-407Б, вариант 10
Преподаватели: Тюменцев Ю.В.

Москва, 2020

Цель работы

Исследовать свойства 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 кривую 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.

Оборудование

Процессор: 2,4 GHz Intel Core 2 Duo

Оперативная память: 8 ГБ 1067 MHz DDR3

Программное обеспечение

Matlab R2020b, 64-bit.

Сценарий выполнения работы

1. С помощью персептрона Розенблатта решить задачу классификации точек плоскости. Точки располагаются по осям в диапазоне $[-5;5]$. Для этого построить и обучить сеть, которая будет правильно классифицировать точки из заданного набора примеров. В сети должны быть нейроны, имеющие ненулевое смещение.

   Для первой обучающей выборки построить и обучить сеть, которая будет правильно относить точки к двум классам.

   1.1. Обучающее множество занести в отчёт.

```
x = [3  -3.8 -1.8 -1.1 -3.2 -4.8; ...
     2.4 0.2 0.4 -0.9 -2.5 4.2];
y = [0 1 1 1 1 0];
```

1.2. Создать сеть. Сконфигурировать сеть под обучающее множество. Отобразить структуру сети с помощью функции *display* и результаты занести в отчёт.

```
network =

    Neural Network

              name: 'Custom Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 1
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 3
        sampleTime: 1

    connections:

       biasConnect: true
      inputConnect: true
      layerConnect: false
     outputConnect: true
```

```
subobjects:

         input: Equivalent to inputs{1}
        output: Equivalent to outputs{1}

        inputs: {1x1 cell array of 1 input}
        layers: {1x1 cell array of 1 layer}
       outputs: {1x1 cell array of 1 output}
        biases: {1x1 cell array of 1 bias}
  inputWeights: {1x1 cell array of 1 weight}
  layerWeights: {1x1 cell array of 0 weights}

functions:

      adaptFcn: 'adaptwb'
    adaptParam: (none)
      derivFcn: 'defaultderiv'
     divideFcn: (none)
   divideParam: (none)
    divideMode: 'sample'
       initFcn: 'initlay'
    performFcn: 'mae'
  performParam: .regularization, .normalization
      plotFcns: {'plotperform', plottrainstate}
    plotParams: {1x2 cell array of 2 params}
      trainFcn: 'trainc'
```

```
    trainParam: .showWindow, .showCommandLine, .show, .epochs,
                         .time, .goal, .max_fail

        weight and bias values:

                      IW: {1x1 cell} containing 1 input weight
    matrix
                      LW: {1x1 cell} containing 0 layer weight
    matrices
                       b: {1x1 cell} containing 1 bias vector

        methods:

                   adapt: Learn while in continuous use
                configure: Configure inputs & outputs
                   gensim: Generate Simulink model
                     init: Initialize weights & biases
                  perform: Calculate performance
                      sim: Evaluate network outputs given inputs
                    train: Train network with examples
                     view: View diagram
              unconfigure: Unconfigure inputs & outputs
```

1.3. Реализовать алгоритм обучения по правилу Розенблатта. Код алгоритма занести в отчёт.

```
epoches = 2;
learningCoeff = 0.1;

for epoch = 1:epoches
    for i = 1:length(x) % цикл по всем образцам
```

```
        out = network(x(:, i)); % выход сети для i-го образца
        error = y(:, i) - out;
        network.IW{1} = network.IW{1} + learningCoeff * error *
x(:, i)';
        network.b{1} = network.b{1} + learningCoeff * error;
        fprintf('epochs: %d; iterations: %d; error: %.6e\n',
epoch, i, mae(y - network(x)));
    end
    fprintf('W =\n%s\nb =\n%s\n', mat2str(network.IW{1}),
mat2str(network.b{1}));
end
```

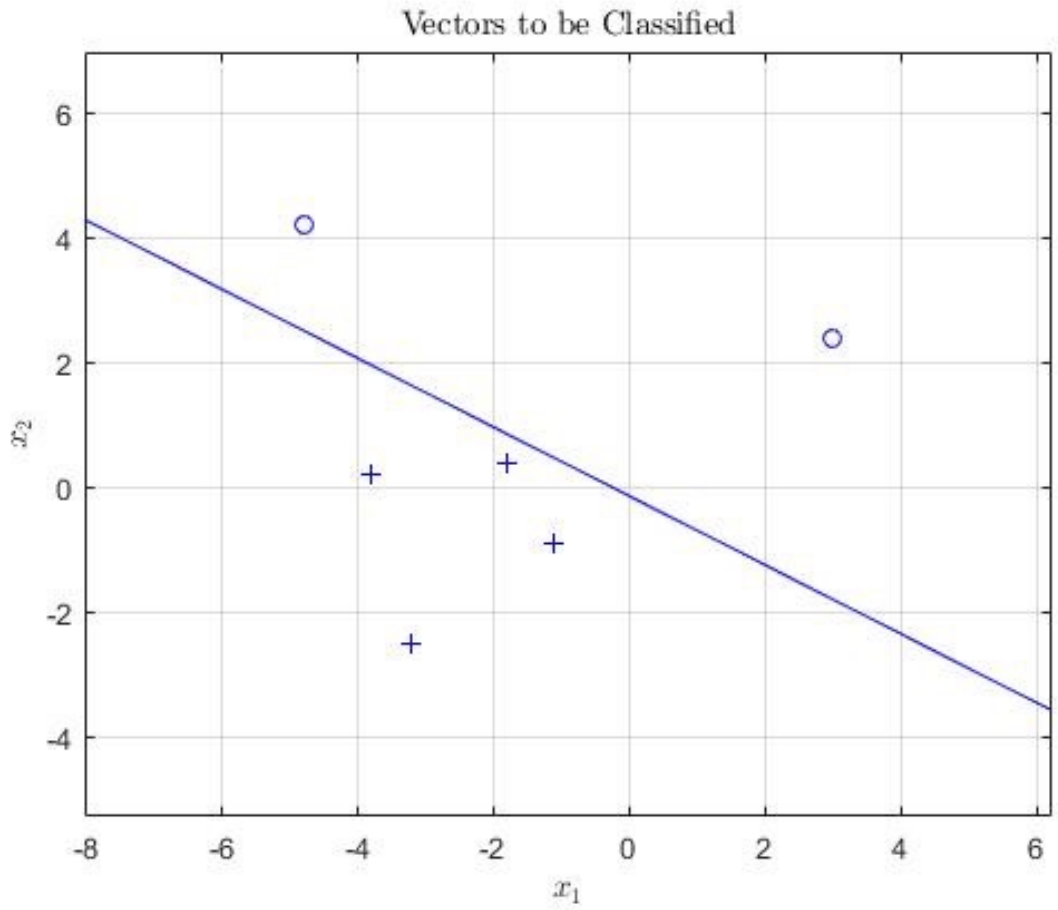
1.3.1. Инициализировать сеть случайными значениями. Для инициализации весов и смещений использовать функцию *rands*. Занести в отчёт весовые коэффициенты и смещения.

```
W = [-0.1225   -0.2369]
b = -0.9311
```

1.3.2. Рассчитать два цикла обучения по правилу. Для расчёта выходов сети использовать функцию *net*. В качестве показателя качества обучения использовать функцию *mae*. Занести в отчёт весовые коэффициенты и смещения после расчёта каждой эпохи (итерации). Также занести в отчёт ошибку обучения сети по всей обучающей выборке $\left(mae(T - net(P))\right)$.

```
|epochs: 1 | iterations: 1 | error: 3.333333e-01|
|epochs: 1 | iterations: 2 | error: 3.333333e-01|
|epochs: 1 | iterations: 3 | error: 3.333333e-01|
|epochs: 1 | iterations: 4 | error: 3.333333e-01|
|epochs: 1 | iterations: 5 | error: 3.333333e-01|
|epochs: 1 | iterations: 6 | error: 5.000000e-01|
W = [-0.0730035622659032 -0.416236961590032]
b = -0.804919190001181
|epochs: 2 | iterations: 1 | error: 5.000000e-01|
|epochs: 2 | iterations: 2 | error: 1.666667e-01|
|epochs: 2 | iterations: 3 | error: 1.666667e-01|
|epochs: 2 | iterations: 4 | error: 1.666667e-01|
|epochs: 2 | iterations: 5 | error: 1.666667e-01|
|epochs: 2 | iterations: 6 | error: 3.333333e-01|
W = [-0.153003562265903 -0.776236961590032]
b = -0.704919190001181
```

1.3.3.После обучения отобразить обучающую выборку и дискриминантную линию. Для отображения использовать функции *plotpv* и *plotpc* соотвественно. Также отобразить сетку с помощью функции *grid*.



1.4. Провести обучение сети с помощью встроенной функции *train* и проверить качество обучения. Занести в отчёт окно *Neural Network Training*.

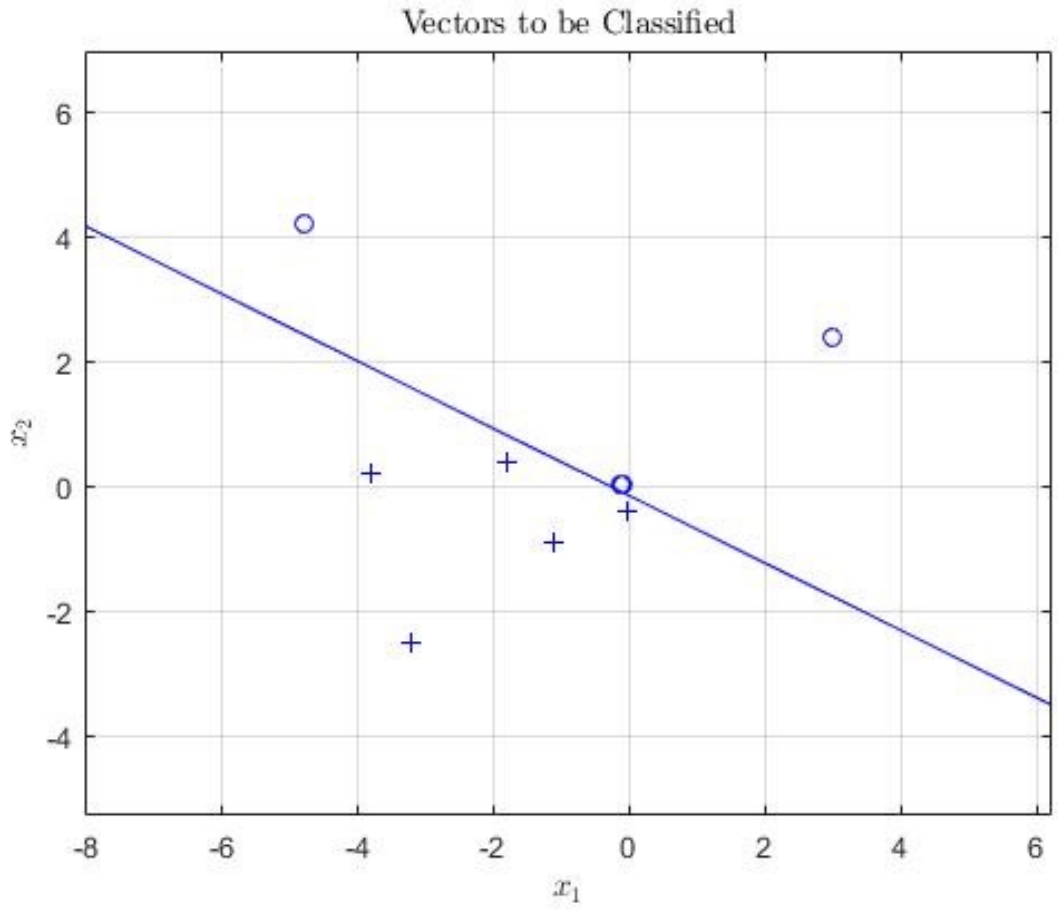
1.4.1. Инициализировать сеть случайными значениями.

1.4.2. Провести обучение сети с помощью функции train с числом эпох равным 50. Если необходимо, то произвести обучение несколько раз. Занести в отчёт весовые коэффициенты и смещения.

```
W = [-2.1088   -6.1074]

b = -1.0205
```

1.4.3. Проверить качество обучения: случайным образом задать 3 точки и классифицировать их. Для генерации случайных чисел использовать функцию *rands*. Отобразить сетку, дополнительные точки, обучающую выборку и дискриминантную линию. Результаты занести в отчёт.

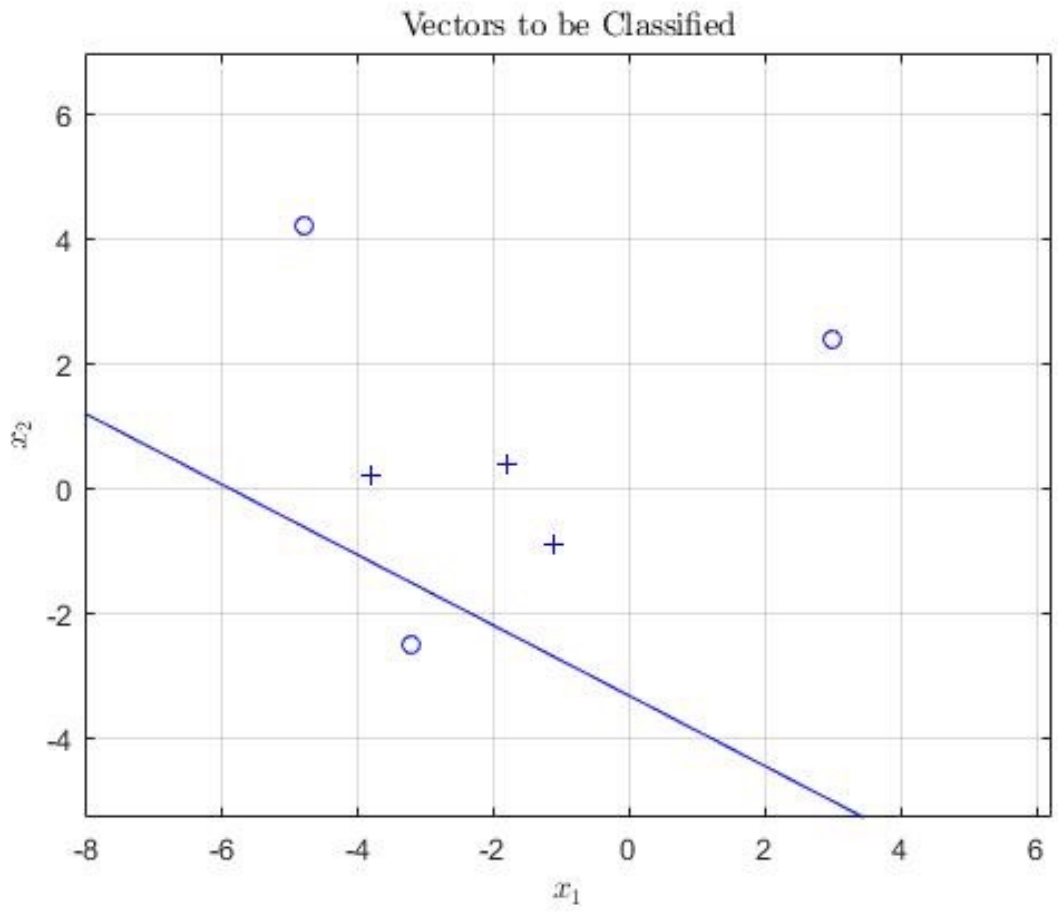


2. Изменить обучающее множество так, чтобы классы стали линейно неразделимыми. Проверить возможности обучения по правилу Розенблатта.

   2.1. Изменить обучающее множество.

   2.2. Инициализировать сеть случайными значениями.

   2.3. Провести обучение с помощью функции train с числом эпох равным 50. Отобразить обучающую выборку и полученную дискриминантную линию. Результаты занести в отчёт.


Vectors to be Classified
6
4
2
0
-2
-4
-8
-6
-4
-2
0
2
4
6
x_2
x_1

3. Для второй обучающей выборки построить и обучить сеть, которая будет правильно относить точки к четырём классам.

   3.1. Обучающее множество

```
x = [2    2.3 0.4 -1.9 -3.2 -0.4 4.1 -5;...
     -1.3 4.5 0.4 -4.3 -4.1 -5   1.4 -4.7];
y = [0 0 0 1 1 1 0 1;...
     1 0 0 0 0 1 1 0];
```

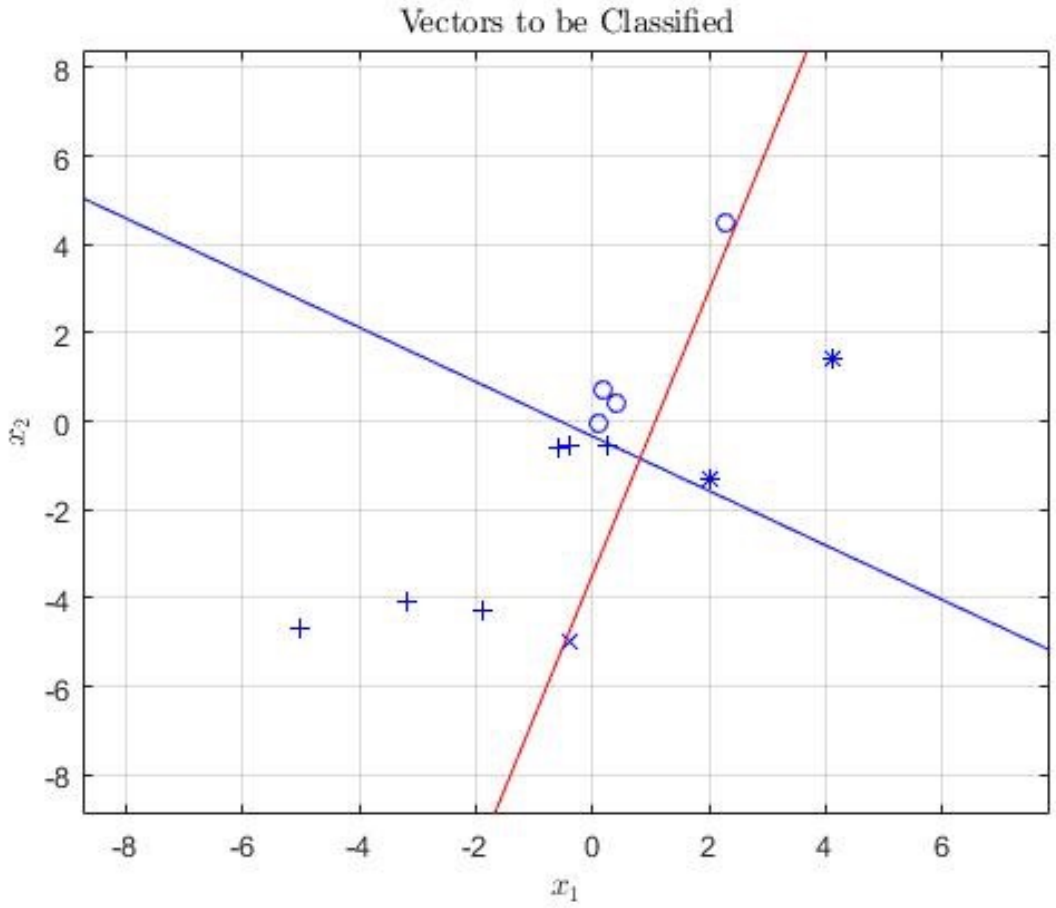
   3.2. Создать сеть.

   3.3. Инициализировать сеть случайными значениями.

   3.4. Провести обучение с помощью функции *train* с числом эпох равным 50. Занести в отчёт весовые коэффициенты и смещения. Занести в отчёт окно *Neural Network Training*.

```
W = [-2.9814 -3.7228; 3.8944 -1.6014]
b = [-0.3829;-13.4898]
```

3.5. Проверить качество обучения: случайным образом задать 5 точек и классифицировать их. Отобразить сетку, дополнительные точки, обучающую выборку и дискриминантные линии. Результаты занести в отчёт



Код программы

```
set(0, 'DefaultTextInterpreter', 'latex');
% Задание 1
% Входные данные
x = [3 -3.8 -1.8 -1.1 -3.2 -4.8; ...
      2.4 0.2 0.4 -0.9 -2.5 4.2];
y = [0 1 1 1 1 0];

% Создаем сеть и инициализируем ее случайными значениями
network = newp([-5 5; -5 5], [0 1]);  % x = [x1, x2], y = [y]
network.inputweights{1}.initFcn = 'rands';
network.biases{1}.initFcn = 'rands';
```

```
network = init(network);
display(network);

% Обучаем ее две эпохи по правилу Розенблатта
epoches = 2;
learningCoeff = 0.1;

for epoch = 1:epoches
    for i = 1:length(x) % цикл по всем образцам
        out = network(x(:, i)); % выход сети для i-го образца
        error = y(:, i) - out;
        network.IW{1} = network.IW{1} + learningCoeff * error * x(:, i)';
        network.b{1} = network.b{1} + learningCoeff * error;
        fprintf('epochs: %d; iterations: %d; error: %.6e\n', epoch, i, mae(y
- network(x)));
    end
    fprintf('W =\n%s\nb =\n%s\n', mat2str(network.IW{1}),
mat2str(network.b{1}));
end

% Результаты обучения
plotpv(x, y);
plotpc(network.IW{1}, network.b{1});
grid on;
xlabel('$x_1$');
ylabel('$x_2$');

%% Инициализируем сеть случайными значениями и обучаем ее встроенной функцией
network = newp([-5 5; -5 5], [0 1]);
network.inputweights{1}.initFcn = 'rands';
network.biases{1}.initFcn = 'rands';
network = init(network);
network.trainParam.epochs = 50;
network = train(network, x, y);

% Результаты обучения с классификацией случайных точек
randomDotsX = rands(3, [-5 5; -5 5])';
randomDotsY = network(randomDotsX);

plotpv([x randomDotsX], [y randomDotsY]);
plotpc(network.IW{1}, network.b{1});
grid on;
xlabel('$x_1$');
ylabel('$x_2$');

%% Задание 2
% Обучение на линейнонеразделимом множестве
```

```
x = [3 -3.8 -1.8 -1.1 -3.2 -4.8; ...
      2.4 0.2 0.4 -0.9 -2.5 4.2];
y = [0 1 1 1 0 0];

plotpv(x, y);

% Инициализируем сеть случайными весами
network = newp([-5 5; -5 5], [0 1]);
network.inputweights{1}.initFcn = 'rands';
network.biases{1}.initFcn = 'rands';
network = init(network);

% Обучаем ее
network.trainParam.epochs = 50;
network = train(network, x, y);

% Результаты обучения
plotpv(x, y);
plotpc(network.IW{1}, network.b{1});
grid on;
xlabel('$x_1$');
ylabel('$x_2$');

%% Задание 3
% Классификация по 4 классам
% Входные данные
x = [2 2.3 0.4 -1.9 -3.2 -0.4 4.1 -5;...
     -1.3 4.5 0.4 -4.3 -4.1 -5 1.4 -4.7];
y = [0 0 0 1 1 1 0 1;...
     1 0 0 0 0 1 1 0];

% Инициализируем сеть случайными весами
network = newp([-5 5; -5 5], [0 1; 0 1]);
network.inputweights{1}.initFcn = 'rands';
network.biases{1}.initFcn = 'rands';
network = init(network);

% Обучаем ее
network.trainParam.epochs = 50;
network = train(network, x, y);

% Результаты обучения с классификацией случайных точек
randomDotsX = rands(5, [-5 5; -5 5])';
randomDotsY = network(randomDotsX);

plotpv([x randomDotsX], [y randomDotsY]);
plotpc(network.IW{1}, network.b{1});
```

```
grid on;
xlabel('$x_1$');
ylabel('$x_2$');
```

Выводы

В MatLab оказалось довольно удобно строить простейшие нейросети, благодаря встроенной нейросетевой библиотеке. Персептрон Розенблатта довольно прост в реализации и неплохо работает при разделении линейно-разделимых множеств, для линейно-неразделимых множеств персептрон Розенблатта не применим, что ожидаемо.