\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pattern Recognition

Homework assignment 3

Rahul Krishna



To check whether the classes are separable, we can plot the mean values and test if we can visually draw a hyperplane that can separate the data. The plot looks like Fig 1.

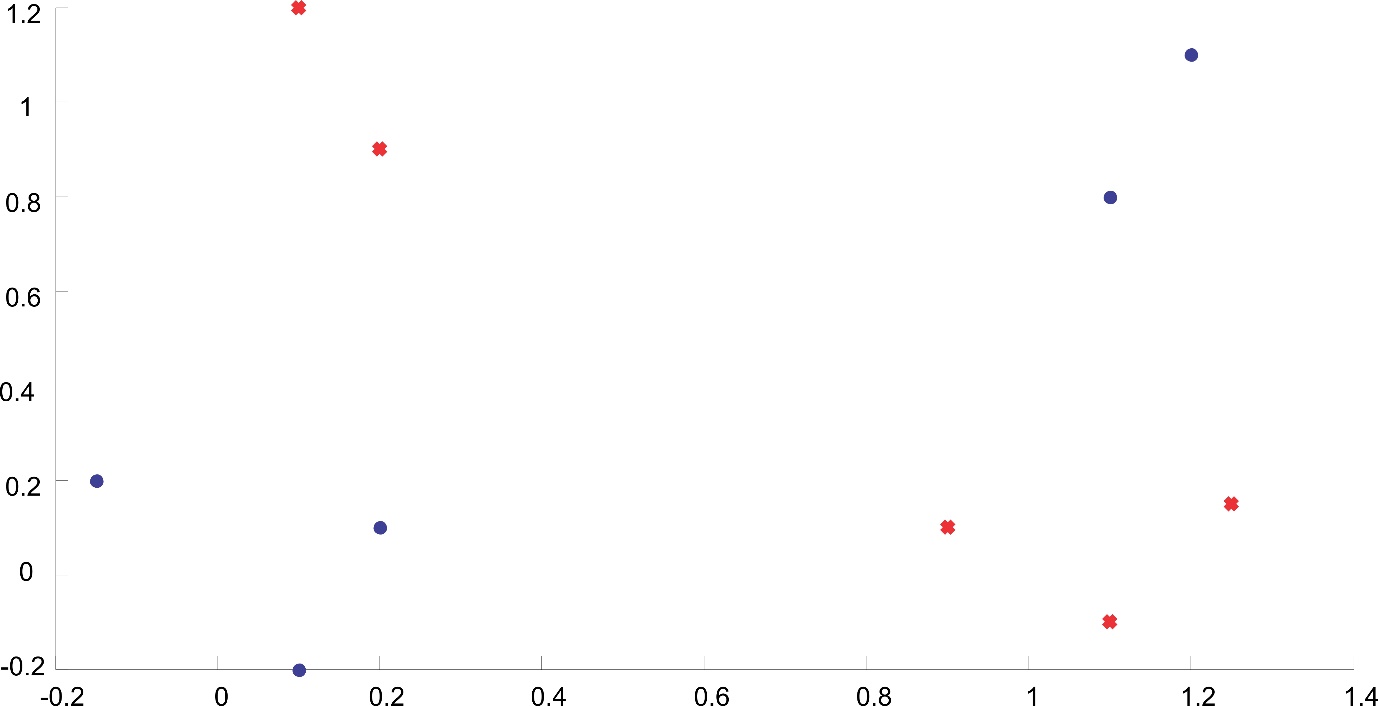


Figure 1: Feature Vectors. Blue - Class 1; Red - Class 2.

One can immediately notice that the data is not linearly separable. Now, in order to classify the data, we can make use of a 3 layer perceptron algorithm. The functions of each layer is described below.

**Layer 1** In this layer we have 5 neurons with 2 dimensions. The primary function of this layer is to map the input vectors onto one of the vertices of a H5 hypercube.

**Layer 2** The aim of the nodes in this layer is to create a hyperplane in the 5-dimensional space. The synaptic weights for each of these second-layer neurons are chosen such that the realized hyperplane leaves only one of the H5 vertices on one side and all the rest on the other. Let us denote the side where the class 1 lies by +1 and the other side by -1. For all the nodes, we get similar hyperplanes.

**Layer 3** From the previous layer we see that each time an input vector from class  enters the network, one of the 5 neurons of the second layer results in a 1 and the remaining 4 give -1. In contrast, for class  vectors all neurons in the second layer output a -1. Classification is now a straightforward task. Choose the output layer neuron to realize an OR gate. Its output will be 1 for class  and -1 for class  vectors.

The decision hyperplanes corresponding to each of the input vectors from class are shown below (Fig 2).

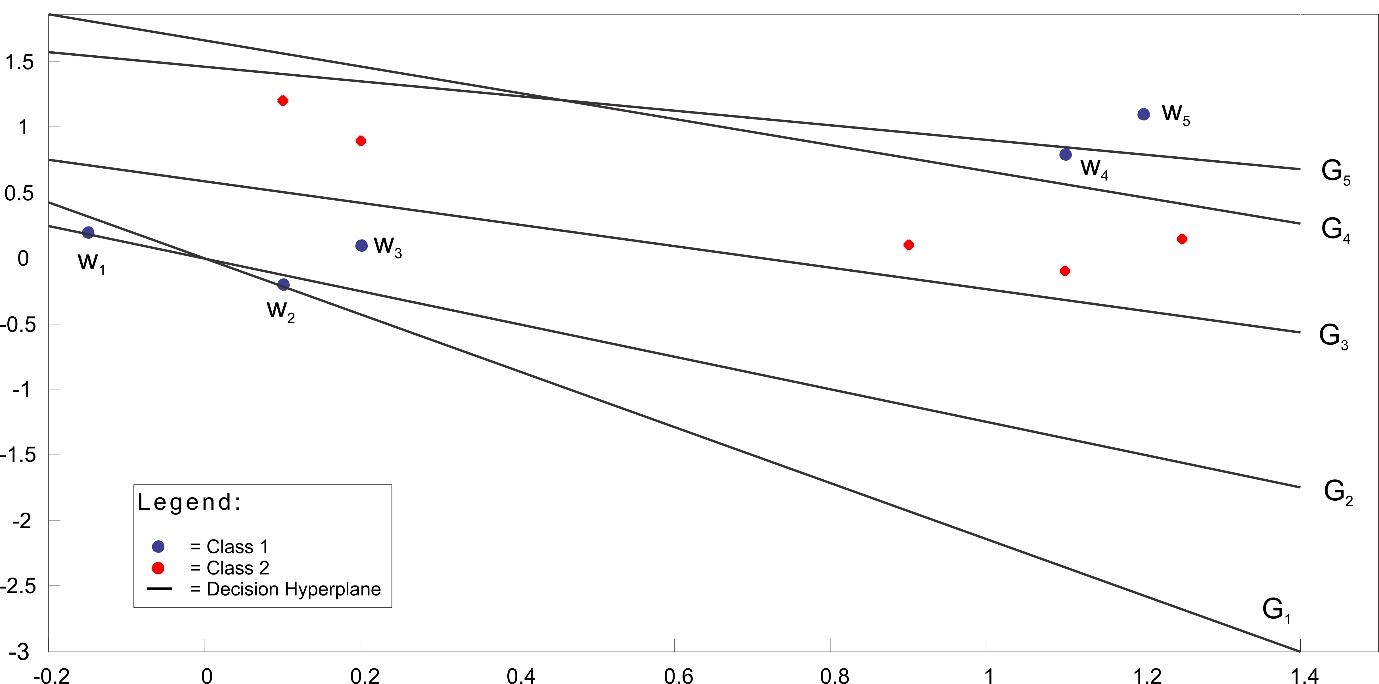


Figure 2: Decision hyperplanes corresponding each of the input vector in class w1

The plot above was generated using matlab. The code is given below:

% Multilayer Perceptron

w1 = [[0.1; -0.2], [0.2; 0.1], [-0.15; 0.2], [1.1; 0.8], [1.2; 1.1]];

w1(3,:) = 1;

w2 = [[1.1; -0.1], [1.25; 0.15], [0.9; 0.1], [0.1; 1.2], [0.2; 0.9]];

w2(3,:) = 1;

% Get the 5 decision Hyper Planes

for i=1:5

g{i} = sPercep(w1(:,i), w2, 10);

end

% Decision Node

scatter(w1(1,:),w1(2,:))

hold all, scatter(w2(1,:), w2(2,:), 'rx')

for i=1:5

refline(-g{i}(1)/g{i}(2),-g{i}(3)/(g{i}(2)))

end

% % The perceptron function

function w =sPercep(x,Y,nIter)

% x - mean of data from Class 1

% Y - a [2xN] matrix with data from class 2, where each row is a mean value

% nIter - number of iterations the perceptron must run before culminating

% w is the output mean value

w = zeros(3,1);

[~,N]=size(Y);

tempMat = zeros(3,N+1);

tempMat(:,1)=x;

tempMat(:,2:N+1)=Y(:,1:N);

for iter=1:nIter

for n=1:N+1

if n==1

res = w'\*tempMat(:,n);

if res<=0

w=w+tempMat(:,n);

end

else

res = w'\*tempMat(:,n);

if res>=0

w=w-tempMat(:,n);

end

end

end

end

return



1. Generating random sequences:

%% PART 1 - To train a neural network

errout=0;

for i=1:10

s=0.01;

m=[[0;0],[1;1],[0;1],[1;0]];

N=100;

k=2;

mu=i;

[X, y] = data\_generator(m,s,N,0);

par\_vec=[mu, 0, 0, 0];

net = NN\_training(X,y,k,1,100,par\_vec)

% At this point a new GUI window opens, the mean dquared error can be

% computed using the GUI.

1. Computing the Probability of error.

[new\_X, new\_y]=data\_generator(m,s,N/2,0);

y1=sim(net,new\_X);

err=sum(new\_y.\*y1<0)/length(new\_y);

errout=[errout,err];

end

bar(errout(2:11));

% Data generator function

function [X, y] = data\_generator(m,s,N,seed)

randn('seed',seed);

S = s\*eye(2);

[l c]=size(m);

X=[];

for i=1:c

X = [X mvnrnd(m(:,i)',S,N)'];

end

y=[ones(1,N) ones(1,N) -ones(1,N) -ones(1,N)];

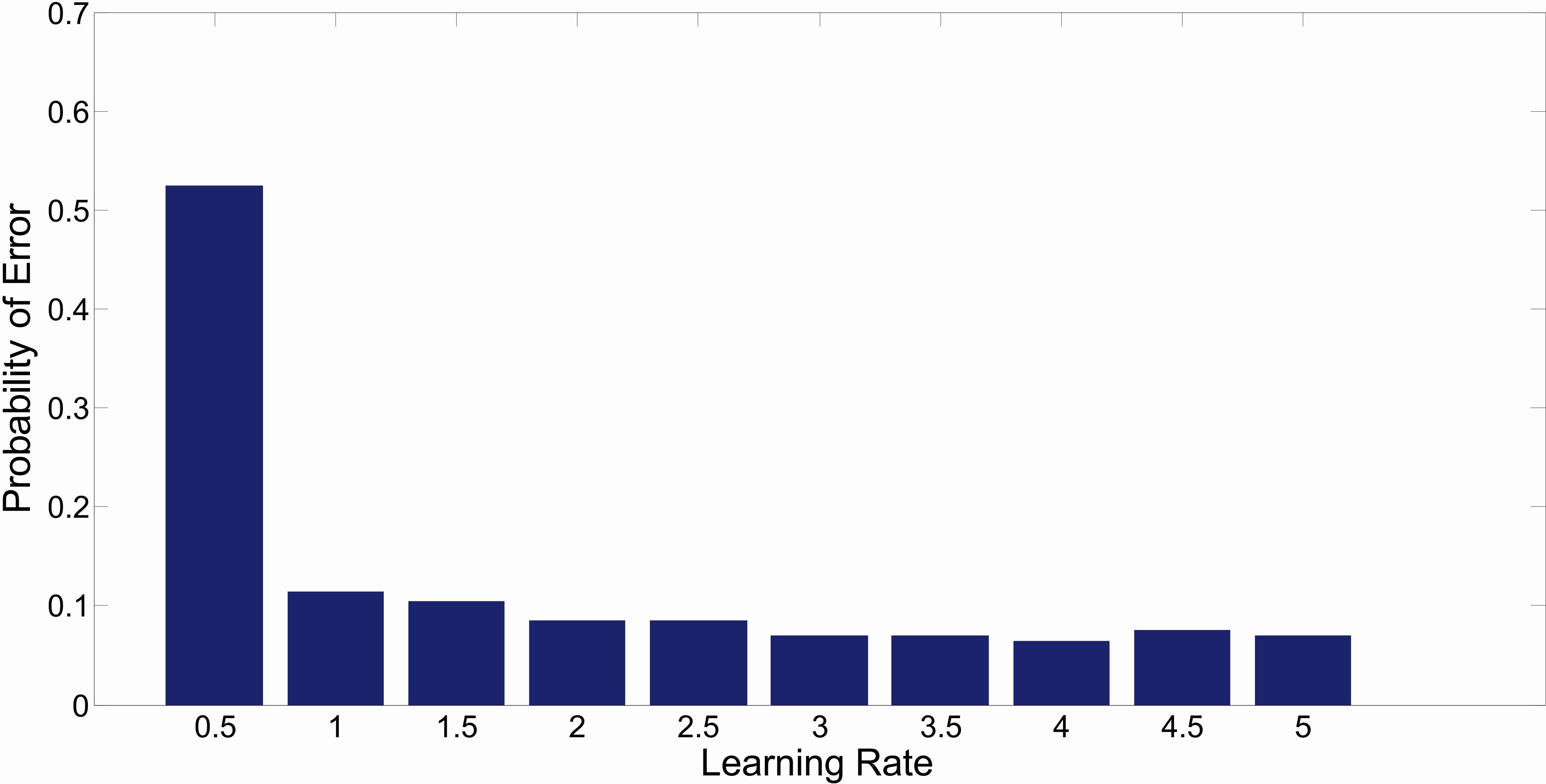
end

1. Mean squared error as a function of iterations.

MSE=0.86916



1. Variation of Probability of error with learning rate 



We can see that as the learning rate increases the probability of error decreases.

COMPUTER EXPERIMENT

(a)

%% Data Set [X1 y1], [X2 y2]

s=2;

m=[[-5;5],[5;-5],[5;5],[-5;-5]];

N=100;

seed1=0;

seed2=10;

[X1, y1]=data\_generator(m,s,N,seed1);

[X2, y2]=data\_generator(m,s,N,seed2);

lr=0.01;

par\_vec=[lr, 0, 0, 0];

Nodes=[2, 4, 15];

Net1=NN\_training(X1, y1, Nodes(1), 1, 1000, par\_vec);

Net1=NN\_training(X1, y1, Nodes(2), 1, 1000, par\_vec);

Net1=NN\_training(X1, y1, Nodes(3), 1, 1000, par\_vec);

% Data generator function

function [X, y] = data\_generator(m,s,N,seed)

rand('seed',seed);

S = s\*eye(2);

[l c]=size(m);

X=[];

for i=1:c

X = [X mvnrnd(m(:,i)',S,N)'];

end

y=[ones(1,N) ones(1,N) -ones(1,N) -ones(1,N)];

end

(b) The previous code generated 2 vectors X1 and X2 according to the question. Now, we shall use Probability of error to measure how the pervious code works when X2 is classified with a neural network trained with data form X1.

Code:

%% Probablity of error

y2=sim(net,X2); % Replace X2 by X1 for class 1

err=sum(y2.\*y1<0)/length(y2);

err

|  |  |  |
| --- | --- | --- |
| Nodes | Data Classified = X1 | Data Classified = X2 |
|  |  |  |
| 2 | 0.0025 | 0.0075 |
| 4 | 0 | 0.0075 |
| 15 | 0 | 0.0075 |

(c) We can notice from the above table that as the number of nodes in the perceptron increases, the Probability of Error reduces. But, when data from class X2 is classified using the neural network obtained from X1, the probability of error is slightly greater, and it doesn’t depend on the number of nodes.