EE559- Mathematical Pattern Recognition

Homework #6

Sourabh Tirodkar
3589406164

Results:

Problem 1:

Wine Data Set

a. I am using PRTools 5 and MATLAB for the implemenatation.

b. In real-life and industry, we only have training data which we can work with. So pre-processing techniques which includes standarization and recycling etc should be used for only for training data set. The testing data is not there beforehand.

   Also, if we do have the testing dataset into our account, we cannot check it and use it. Our goal is to create a classification algorithm that can work with any sets of testing data and this should be done once the training dataset is learned.

   Hence, we use normalization only for the training data set and apply the values calculated by it into the testing data.

   Standarization (Normalization of data)

```
Mean of Training Vector:
  Columns 1 through 12

   12.9654    2.2700    2.3763   19.6494   98.9101    2.2724    2.0294    0.3607↙
1.5762    5.0912    0.9532    2.5603

  Column 13

  729.7079

Standard Deviation of Training Vector:
  Columns 1 through 12

    0.8242    1.1093    0.2746    3.4848   11.6245    0.6180    0.9249    0.1209↙
0.5445    2.4180    0.2312    0.7276

  Column 13

  308.9619

After Standarization:
Column Mean:
   1.0e-14 *

  Columns 1 through 12

   -0.3739   -0.0035    0.2258    0.0357   -0.0433    0.0042    0.0531    0.0629↙
-0.1123    0.0213    0.1241   -0.1724

  Column 13

   -0.0207

Column SD
  Columns 1 through 12

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000↙
1.0000    1.0000    1.0000    1.0000

  Column 13

    1.0000
```

c.

1. Initial weight vector for perlc in PRTools 5 is Random Initialization. This can be seen at 'perlc.m'. The documentation is as below:

```
W = PERLC(A,MAXITER,ETA,W_INI,TYPE)
%   A         Training dataset
%   MAXITER   Maximum number of iterations (default 1000)
%   ETA       Learning rate (default 0.1)
%   W_INI     Initial weights, as affine mapping, e.g W_INI = NMC(A)
%             (default: random initialisation)
%   TYPE      'batch': update by batch processing (default)
%             'seq'  : update sequentially
```

2. There are two types of halting conditions.
   The first one is ‘tol’ argument in the Perceptron model. The default value is set to none, but if it is given a value, then iterations stop when difference between weight vectors from two successive iterations is less than tol value.
   The second halting condition: ‘max_iter’ argument which gives the max number of Iterations and after that no iterations are computed.

d. <u>Considering 2 features:</u>

```
errorTest1 =

    0.1910


Training_Accuracy_perceptron =

   80.8989


Testing_Accuracy_perceptron =

   80.8989

Final Weights

ans =

   -1.6580   -1.1169   -0.1099
    2.7638   -3.4761   -0.5020
   -1.2689   -1.5784    0.8123

fx >>
```

Considering 13 features:

```
    0.0899


Training_Accuracy_perceptron =

   100


Testing_Accuracy_perceptron =

   91.0112

Final Weights

ans =

 -268.8337  -57.2999 -179.6252
  355.7805 -144.2095   23.9509
  -71.5754  -84.3681   58.2089
  164.3512  -48.9568   17.2083
 -195.8130   42.6302    9.4834
  225.1471  -63.7986   51.2364
   33.7762   -8.0741  -25.5727
  185.4966   -6.6106 -100.0025
  -83.9096   10.2808  -38.0480
  -17.2602  -14.9700  -54.3956
  -14.1700 -127.1567  123.4165
  119.9479   44.1732 -122.9134
  216.4516   28.3962  -94.5342
  377.3435 -118.5952  -47.7680
```

e. 100 epoch

1. Considering 2 features

```
errorTest1 =

    0.2360


Training_Accuracy_perceptron =

   83.1461


Testing_Accuracy_perceptron =

   76.4045

Final Weights

ans =

   -1.6771   -0.7313   -0.6936
    2.7238   -4.3038    0.0263
   -1.2825   -0.9391    1.7199

fx >>
```

2. Considering 13 features

```
Training_Accuracy_perceptron =

   100


Testing_Accuracy_perceptron =

   92.1348

Final Weights

ans =

 -149.4106  -18.6794  -41.8036
   81.6257  -32.1575    3.7154
  -31.3660  -22.3807    6.6887
   63.4393  -16.3619    3.9312
  -26.9162    8.5818   -0.0499
   41.6934  -16.5910    2.4824
   32.9665    1.8323  -11.1073
   72.4471    0.7448  -17.9195
  -18.7361   10.6768   -7.8331
   23.4465    1.8850   -6.3954
   -5.7662  -30.4216   24.5202
   56.2730   14.6560  -19.5196
   69.3481    3.9891  -15.7466
   93.1246  -27.6508   -3.0173

fx >>
```

f. Part d comparison.
   Training accuracy has increased from 80.8989 to 100. This is obvious as the more the data, better it is learned.
   For testing too, the same follows. The testing accuracy has been increased as we go from 2 features to 13 features.

   Part e comparison.
   This is for 100 iterations. Training accuracy has increased from 83% to 100%. This can be followed from the part d as well.
   For testing too, the same follows. The testing accuracy has been increased as we go from 2 features to 13 features.

Comparison of 2 features
In 1 epoch, we get training accuarcy as 80.8989
In 100 epoch, we get training accuracy as 83.1461

In 1 epoch, we get testing accuarcy as 80.8989
In 100 epoch, we get testing accuracy as 76.4045

As see, the training accuracy of trainind data increases.
But testing accuracy is seen to be reduced.

Comparison of 13 features
In 1 epoch, we get training accuarcy as 100
In 100 epoch, we get training accuracy as 91

In 1 epoch, we get testing accuarcy as 100
In 100 epoch, we get testing accuracy as 92

As see, the training accuracy of training data is 100% for both 1 epoch and 100 epoch.
Also, as we increased the epoch, we do get better accuracy in terms of training accuracy.

g. Considering 2 features (Non Standarized data)

```
errorTest2 =

    0.2247


Training_Accuracy_MSE =

   83.1461


Testing_Accuracy_MSE =

   77.5281

Training weights

ans =

  -42.3536   64.1456  -11.1332
    3.3491   -4.8480    0.4399
   -1.1209   -1.0995    1.6393
```

Considering 13 features (Non Standarized Data)

```
Training_Accuracy_MSE =

   100


Testing_Accuracy_MSE =

   96.6292

Training weights

ans =

 -270.6938  117.5612  -42.4085
    9.6759   -6.0842    4.3647
    2.4839   -3.3175    5.3339
   22.8514  -15.7121   13.5315
   -2.1319    1.0174   -0.1861
    0.2903   -0.1685    0.0971
  -11.3742    2.5238    5.9771
   12.1864    3.0395  -20.1883
   -0.3912   20.9808  -49.9399
   -1.0487    1.7663   -3.1295
   -0.3247   -2.3088    5.8847
    7.0994    3.4826  -15.8695
   14.6984   -0.3025  -14.8254
    0.0607   -0.0196   -0.0170

fx >>
```

h. Conisdering 2 features (Using Standarized Data)

```
errorTest2 =

    0.2247


Training_Accuracy_MSE =

   83.1461


Testing_Accuracy_MSE =

   77.5281

Training weights

ans =

   -1.4761   -1.2064   -1.7085
    2.7603   -3.9957    0.3626
   -1.2434   -1.2197    1.8185

>>
```

Conisdering 13 features (Using Standarized Data)

```
Training_Accuracy_MSE =

   100


Testing_Accuracy_MSE =

   96.6292

Training weights

ans =

  -14.3715   -4.1509  -21.4974
    7.9749   -5.0146    3.5974
    2.7555   -3.6801    5.9170
    6.2739   -4.3138    3.7151
   -7.4293    3.5454   -0.6484
    3.3751   -1.9582    1.1287
   -7.0296    1.5598    3.6941
   11.2716    2.8113  -18.6727
   -0.0473    2.5371   -6.0389
   -0.5710    0.9618   -1.7041
   -0.7850   -5.5827   14.2291
    1.6415    0.8052   -3.6692
   10.6940   -0.2201  -10.7863
   18.7418   -6.0692   -5.2633

fx >>
```

i. The test accuracy results of part g and h are identical. The weight vectors do change but using either standarized or non standarized data doesn't cause any differences in either training or testing data set.
For both 2 feature and 13 feature selection, the testing accuracy is identical.

j. Comparison of results of Perceptron and MSE

2 features:
For Perceptron: In 100 epoch, we get testing accuracy as 76.4045
For MSE: we get 77.5281
As seen, we have testing accuracy which is quite similar and close to each other. As MSE is an approximation for the perceptron, the results does shows similarity.

13 features:
For Perceptron: Test accuracy is 92.148
For MSE: Test accuracy is 96.6292
From above, we can conclude that MSE test accuracy is slightly better than the perceptron classifier.

Sourabh Tirodkar 3589406164

EE 559

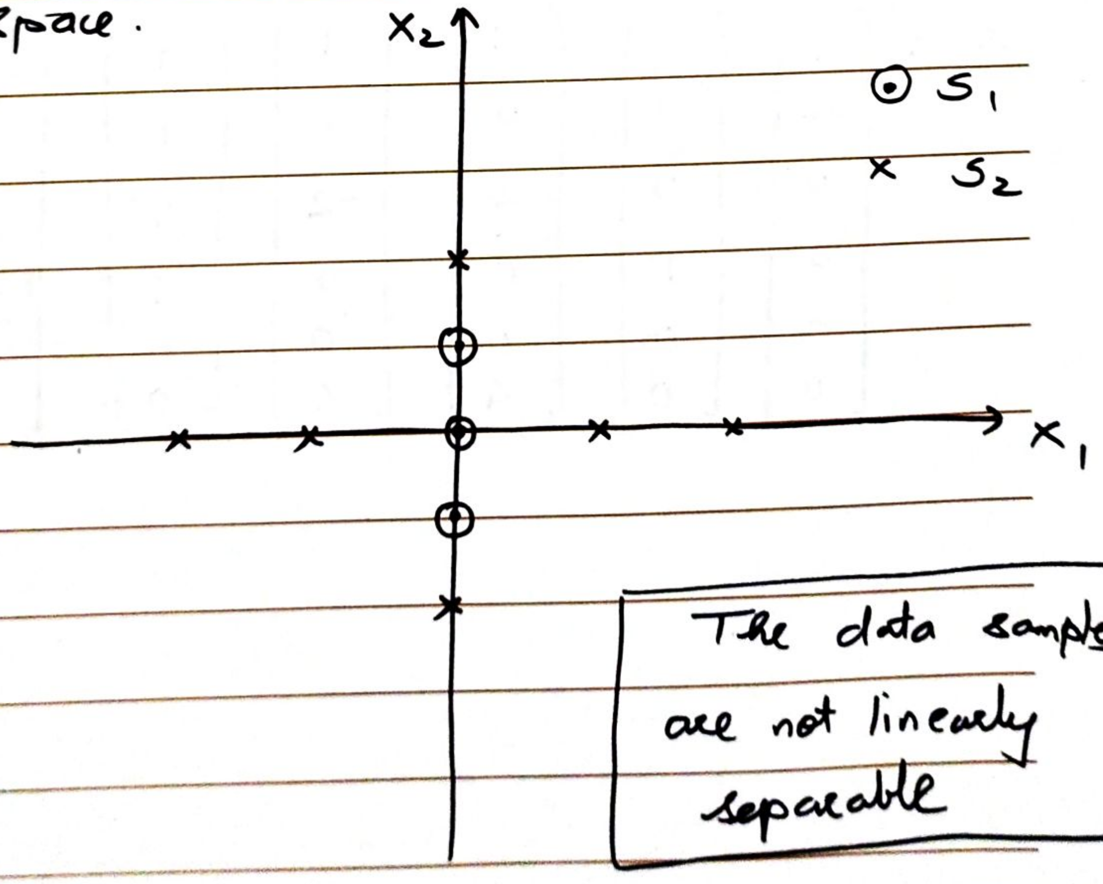
HW # 6

Q) Problem 2

$S_1 : (0,0)^T, (0,1)^T, (0,-1)^T$

$S_2 : (-2,0)^T, (-1,0)^T, (0,2)^T,$
$(0,-2)^T, (1,0)^T, (2,0)^T$

a) Plot in 2D (non-augmented) feature space.



The data samples are not linearly separable

Solving Non-linear classifier
Phi - Machine Approach

b) Expanded feature space : Data points.

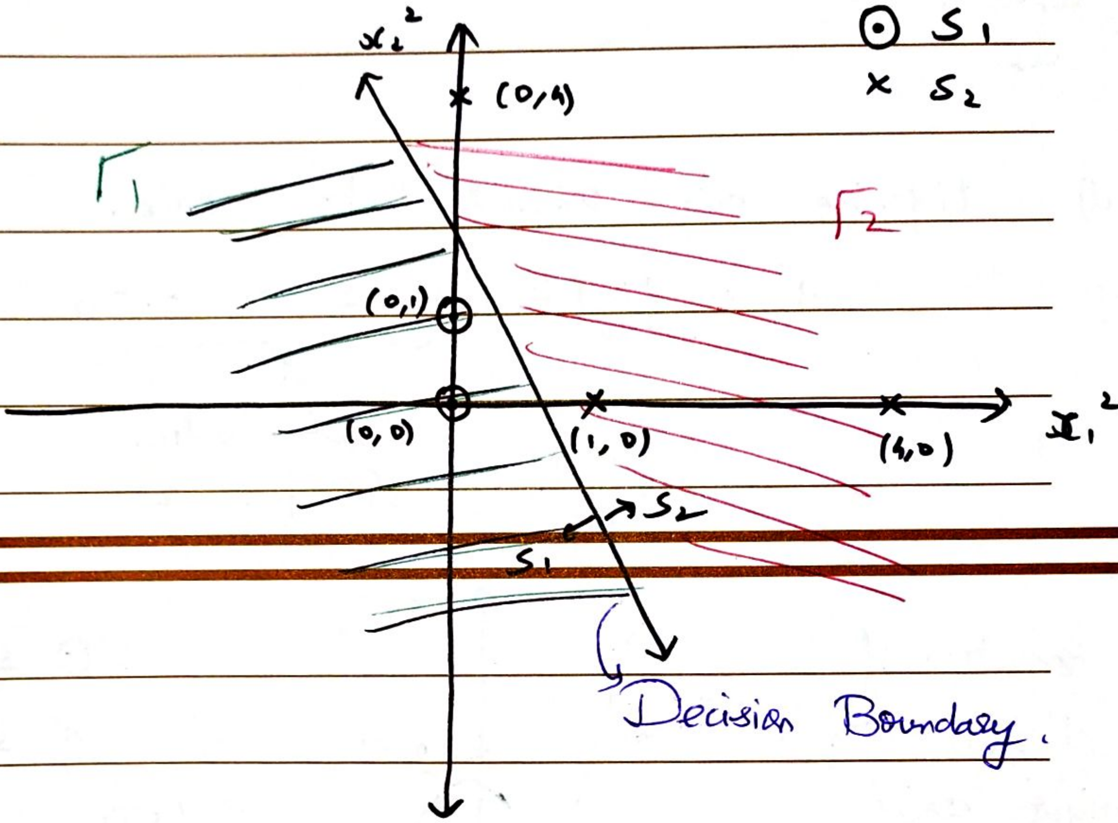
$$\begin{array}{c} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{array} \quad S_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$S_2 : \begin{bmatrix} 1 \\ -2 \\ 0 \\ 4 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \\ 0 \\ 0 \\ 4 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -2 \\ 0 \\ 0 \\ 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \\ 4 \\ 0 \\ 0 \end{bmatrix}$$

c) Plot, Axis : x_1^2 & x_2^2

$S_1 : (0,0)^T, (0,1)^T, (0,1)^T$

$S_2 : (4,0)^T, (1,0)^T, (0,4)^T, (0,4)^T,$
$(1,0)^T, (4,0)^T$



Decision eqn

$$H = g(u) = 1 - 2u_1 - 0.5u_2 = 0$$

Here, $u_1 = x_1^2$
$u_2 = x_2^2$

Decision Rule

$g(u_1) > 0 \Rightarrow u_1 \in S_1$
$g(u_2) < 0 \Rightarrow u_2 \in S_2$

$$W^T = \begin{bmatrix} 1 \\ -2 \\ -0.5 \end{bmatrix} \qquad u = \begin{bmatrix} 1 \\ u_1 \\ u_2 \end{bmatrix}$$

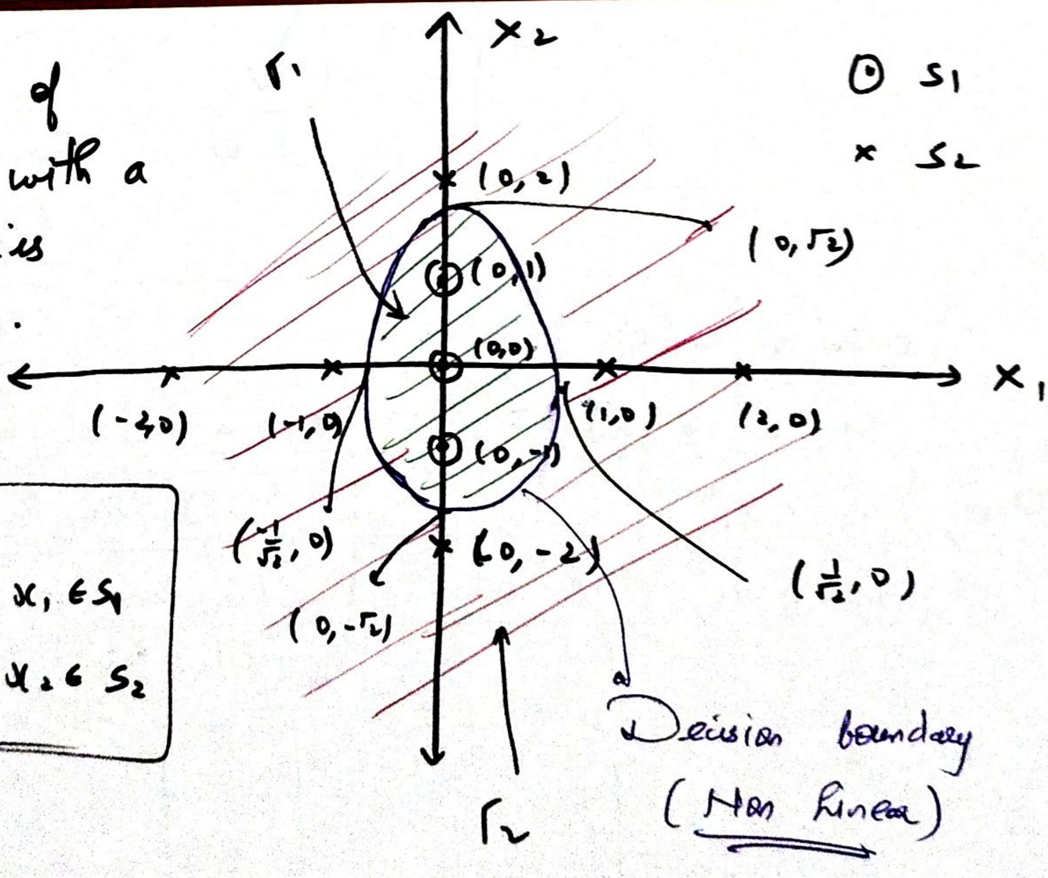
so, $w' = \begin{array}{c} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{array} \begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ -0.5 \end{bmatrix}$

d) Mapping onto original feature space.

$$H = f(x) = \underbrace{1 - 2x_1^2 - 0.5}\, x_2^2 = 0$$

Quadratic.
(Poly. in 2).

Equation of ellipse with a major axis along x_2.



Decision Rule
$g(x_1) > 0 \Rightarrow x_1 \in S_1$
$g(x_2) > 0 \Rightarrow x_2 \in S_2$