EE-559 Mathematical Pattern Recognition
HW #6

Q 1)

Aim: To design a two-class Perceptron classifier [Here, Two-class Two-feature]. For the three datasets: Synthetic 1, Synthetic 2 and Synthetic 3,
(I) Find the final weight vector, classification error rate for training and test data sets.
(II) For each data set plot in 2D showing the data points and the decision boundary.
(III) For Synthetic 1 and Synthetic 2 data sets, compare the classification error rates / results between using perceptron algorithm and MDTCM algorithm.

Program Code: Attached in CODE_EE559_HW6_HarikrishnaPrabhu-Program q1.py

Observation:

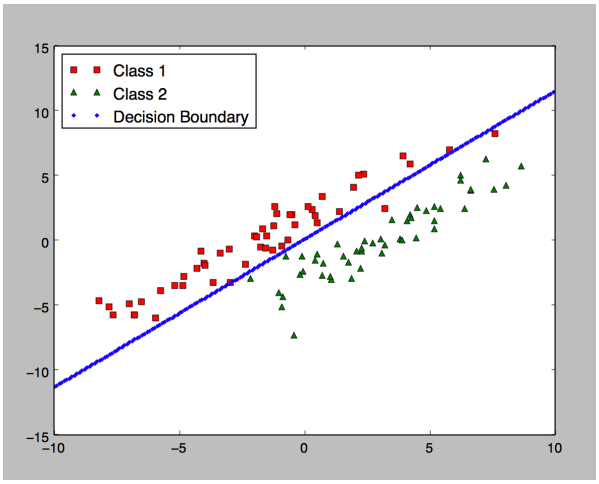

Figure 1: Synthetic_1 train data 2D plot

```
HariKrishnas-MacBook-Pro:trial rickerish_nah$ python q1.py
Enter the data points with which you want to run the Perceptron Algorithm:
(1)Synthetic 1
(2)Synthetic 2
(3)Synthetic 3
1


For Synthetic: 1

Min_w/Final_w: [-23.76442  24.76147  -2.9    ]
Error Rate for Training data: 3
Error Rate for Test data: 0
```

Output 1: Final Weight vector and classification error rates for synthetic 1 train and test data

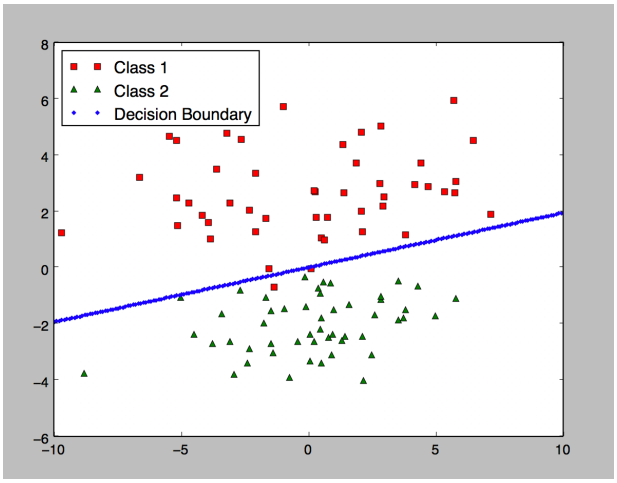

Figure 2: Synthetic_2 train data 2D plot

```
HariKrishnas-MacBook-Pro:trial rickerish_nah$ python q1.py
Enter the data points with which you want to run the Perceptron Algorithm:
(1)Synthetic 1
(2)Synthetic 2
(3)Synthetic 3
2



For Synthetic: 2

Min_w/Final_w: [ -3.2624    12.264332   2.1      ]
Error Rate for Training data: 2
Error Rate for Test data: 1
```

Output2: Final Weight vector and classification error rates for synthetic 2 train and test data

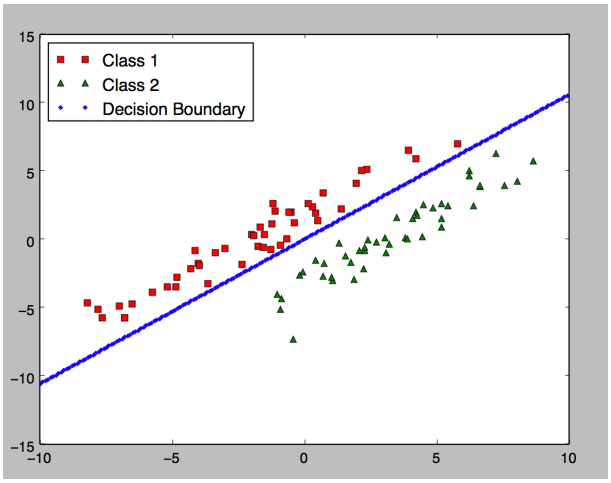

Figure 3: Synthetic_3 (Feature_train) train data 2D plot

```
HariKrishnas-MacBook-Pro:trial rickerish_nah$ python q1.py
Enter the data points with which you want to run the Perceptron Algorithm:
(1)Synthetic 1
(2)Synthetic 2
(3)Synthetic 3
3



For Synthetic: 3

Final: [-11.39314  10.75886   0.1     ]
Error Rate for Training data: 0
Error Rate for Test data: 0
```

Output3: Final Weight vector and classification error rates for synthetic 3 train and test data

```
HariKrishnas-MacBook-Pro:MPR rickerish_nah$ python trial.py
Mean Values for Train 1:
[[-1.8731152  -0.1166418 ]
 [ 2.98095798  0.03548129]]


Mean Values for Train 2:
[[-0.2032685   2.75522592]
 [ 0.13275594 -2.0526066 ]]


Error_Rate of Train 1: 21
Error_Rate of Test 1: 24
Error_Rate of Train 2: 3
Error_Rate of TEST 2: 4
Error_Rate Ratio of Train 1: 21 / 100
Error_Rate Ratio of Test 1: 24 / 100
Error_Rate Ratio of Train 2: 3 / 100
Error_Rate Ratio of Test 2: 4 / 100
HariKrishnas-MacBook-Pro:MPR rickerish_nah$
```

Output4: Error rates of Synthetic 1 and 2 Data sets using MDTCM method.

Inference:

(I) For each of the synthetic data set, final weight vector and the classification error for both train and test data were calculated.

Error rate:

i Synthetic 1 Train: 3% and Synthetic 1 Test: 0%

ii Synthetic 2 Train: 2% and Synthetic 2 test: 1%

iii Synthetic 3 Train: 0% and Synthetic 3 Test: 0%

As shown in output figures 1,2 and 3.

(II) For each of the data sets, the data points and the decision boundary is plotted in a 2D graph as shown in the figures 1, 2 and 3.

(III) For Synthetic 1 Data set, we can see that using MDTCM we observed an error rate of 20% in average and using the Perceptron Algorithm we observed an error rate of 2-3 % in average. This is because the decision boundary drawn in case 1 was with respect to the data mean only and here it's with respect to the data set itself. When we compare the distance between the data points and its projection on the decision boundary, we see that in MDTCM the distance value

vary widely. Whereas in the Perceptron case, the distance between data point and the decision boundary fairly is the same.

In Synthetic 2 data set, we see that the classification error rates are more or less the same, it is just a case of the orientation of the data points. Here it happens for the mean to lie in a such a way that the decision boundary lies aptly classifying the data points. But this method, MDTCM is not as efficient as the Perceptron Algorithm.

MDTCM takes in to account just the spatial distribution of the data points, but the perceptron algorithm takes into account the weight of each data point. Hence is its efficiency.