Midterm Exam (ENME808E)

Author Name
Koyal Bhartia (116350990)

Machine Learning Midterm Exam

Date : 27/2/2019

Contents

1 Problem 1.5 from LFD **3**

1.1 Part a: $\eta = 100$ 4

1.2 Part b: $\eta = 1$ 5

1.3 Part c: $\eta = 0.01$ 7

1.4 Part d: $\eta = 0.0001$ 9

1.5 Part e: Comparision of result and Observations 11

2 Problem 2.5 from LFD **11**

3 Show that the VC dimension of convex d-gons (poygon with d sides) is 2d+1. **13**

4 Problem 3.1 from LFD **16**

4.1 Part a 17

4.2 Part b 18

5 Problem 3.2 from LFD **20**

6 Codes 1(Problem 1.5 from LFD) **24**

7 Codes 2(Problem 3.1 Part(a) from LFD) **28**

8 Codes 3(Problem 3.1 Part(b) from LFD) **32**

9 Codes 3(Problem 3.2from LFD) **36**

List of Figures

1 $\eta = 100$ Final hypothesis and target function on training data 4

2 $\eta = 100$ Output terminal showing Iterations and Error on test data 4

3 $\eta = 1$ Final hypothesis and target function on training data 5

4 $\eta = 1$ Output terminal showing Iterations and Error on test data 6

5 $\eta = 1$ PLA best fit line plotted on the test data 6

6 $\eta = 1$ Final hypothesis and target function on training data 7

7 $\eta = 1$ Output terminal showing Iterations and Error on test data 8

8 $\eta = 1$ PLA best fit line plotted on the test data 8

9 $\eta = 0.0001$ Final hypothesis and target function on training data 9

10 $\eta = 0.0001$ Output terminal showing Iterations and Error on test dat 10

11 $\eta = 0.0001$ PLA best fit line plotted on the test data 10

12 Convex-3-gon shattering 7 points 14

13 Convex-3-gon unable to shattering 8 points 15

14 Convex-4-gon shattering 9 points 15

15 Convex-4-gon unable to shattering 10 points 16

16 Convex-5-gon shattering 11 points 16

17 Semi-Circle Training data along with final hypotheses 17

18 Semi-Circle: Terminal Output - No. of misclassified points and iterations 18

19 PLA best fit line of Linear Regression on Semi-Circle 19

20 Terminal Output of iterations of Linear Regression on Semi-Circle 19

21 Graph of No. of Iterations vs sep of semi circle 20

22 Output terminal of iterations of PLA on semi circle 21

23 Graph of Theoretical Bound vs sep of semi circle 22

24 Output terminal of theoretical bound using PLA semi circle 22

25 Graph of Iteration vs sep 23

1 Problem 1.5 from LFD

The code for this problem can be found in the Codes section. It is scripted in python.

The learning model to be implemented here is an extension of the Perceptron Learning Algorithm. In the case of the general PLA the condition for weight update was: $y(t).s(t) \leq 0$, where the update happens by:

$$w(t+1) \leftarrow w(t) + y(t).x(t) \tag{1}$$

While the update of the general PLA algorithm uses the above check, here we use $y(t).s(t) \leq 1$ as the condition to update the weights by:

$$w(t+1) = w(t) + \eta.(y(t) - s(t)).x(t) \tag{2}$$

In the above equations: $s(t) = w^T(t)x(t)$ and η is a constant or the learning rate.

In the problem here, we consider different learning rates; and analyse it's effect on the convergence rate of the PLA algorithm. It is usually okay to neglect the learning rate as PLA guarantees to find a solution (if one exists) in an upperbounded number of steps. This however depends on the application at hand.

In this problem:

- A training data set is created using: **random.uniform(-1,1)**. The data set created is 2D.
- This data set is classified using the target function: **Slope=0.8 Intercept=0.2**. Points above this line has been classified as output y=1 and below the line as y=-1.
- The initial set of weights is taken as **w=[0 0 0]**.
- The best fit line has been found by continuously updating the weights and finally finding the optimum set of weights.
- The PLA algorithm is ran until no more points are misclassified or the max no of updates reaches 1000.
- The final hypothesis is then plotted using the optimum set of weights obtained.

Below is the implementation of the the variant of the Adaline algorithm for PLA for the values of η as 100, 1, 0.01 and 0.0001.

1.1 Part a: $\eta = 100$

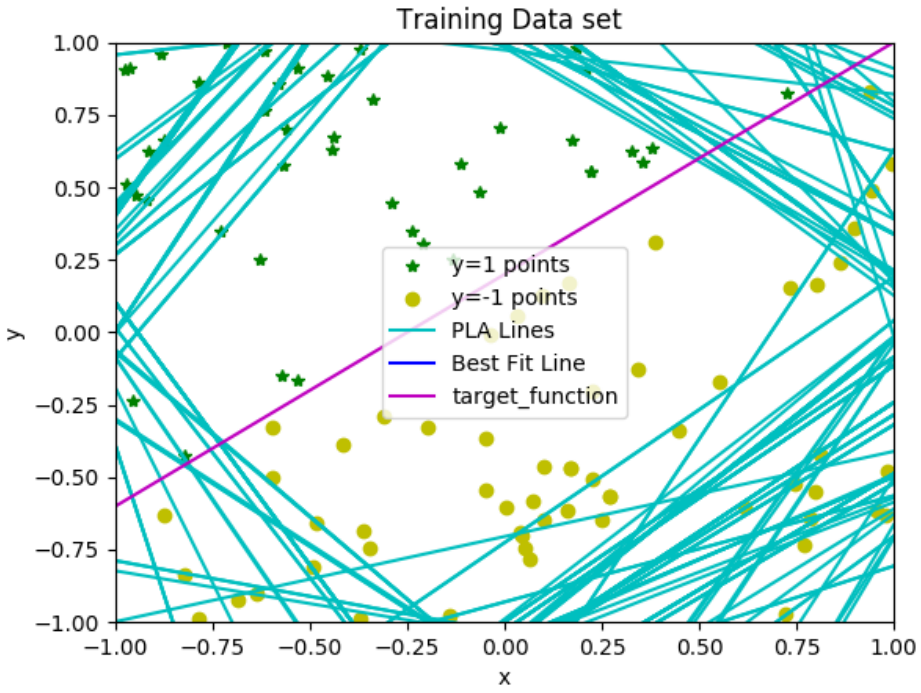

Figure 1: $\eta = 100$ Final hypothesis and target function on training data

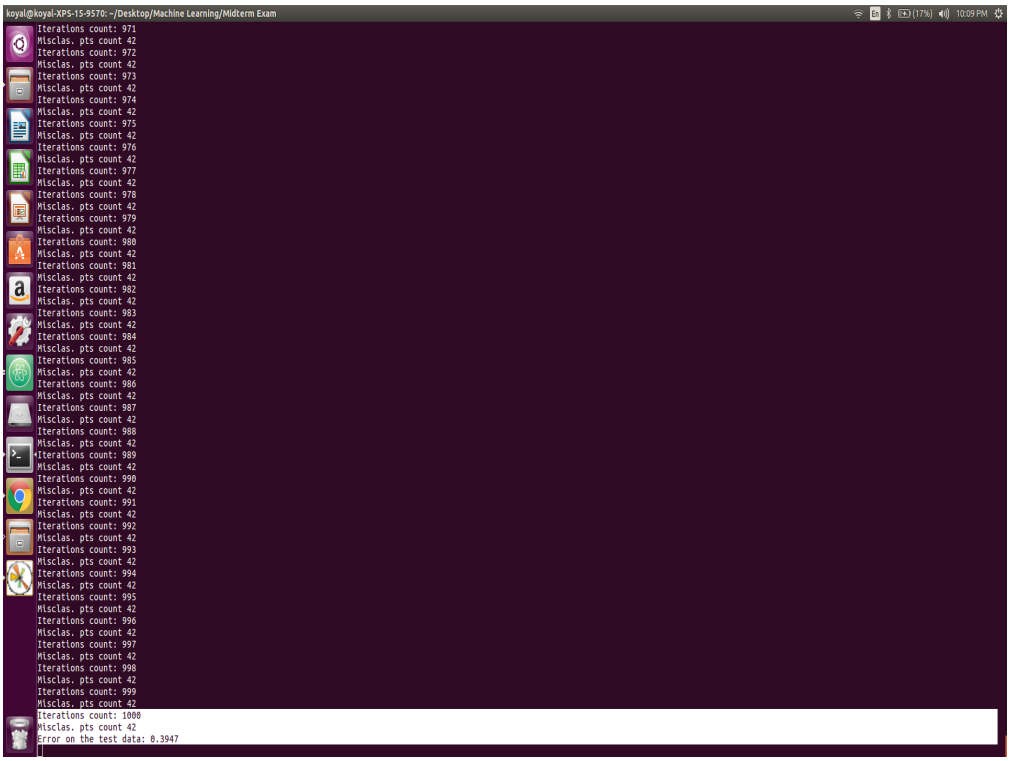

Figure 2: $\eta = 100$ Output terminal showing Iterations and Error on test data

Here the learning rate is too large, and as we see the weight vector immediately becomes the training case, and the algorithm fails to converge to a solution. There happens to be an overshoots and we get non-optimal point whose error is quite higher. From the terminal output it can be seen that even after 1000 updates, the algorithm is unable to unconverge. The no of misclassified points is still around 42, and we do not get any PLA best fit line or the final hypothesis. The error is also veryy high in the range 0.3..!!

1.2 Part b: $\eta = 1$

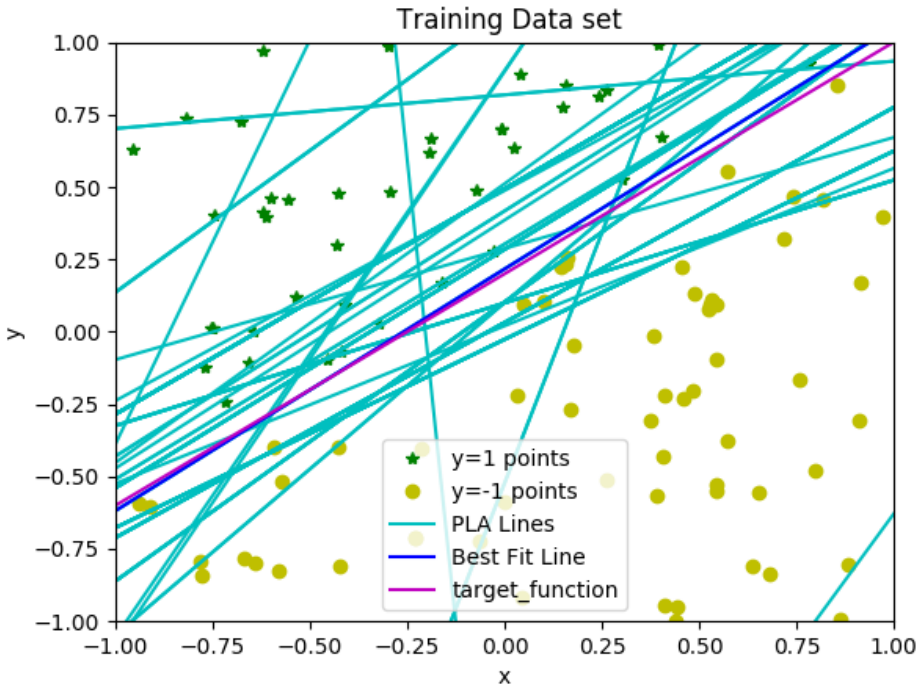

Figure 3: $\eta = 1$ Final hypothesis and target function on training data

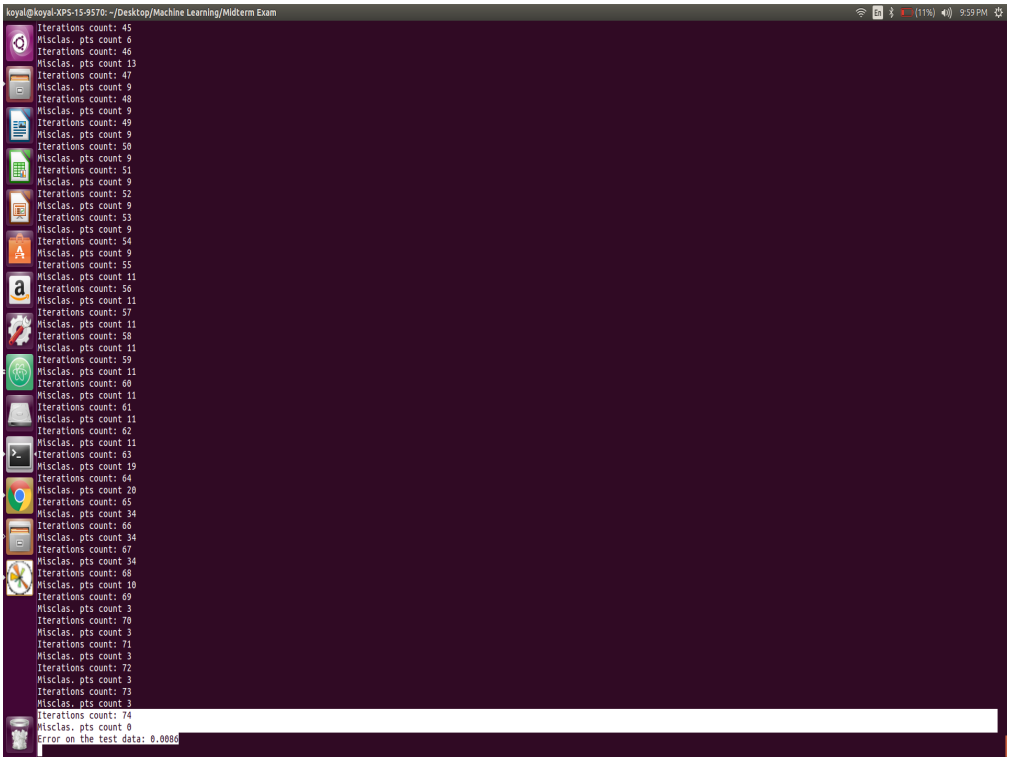

Figure 4: $\eta = 1$ Output terminal showing Iterations and Error on test data

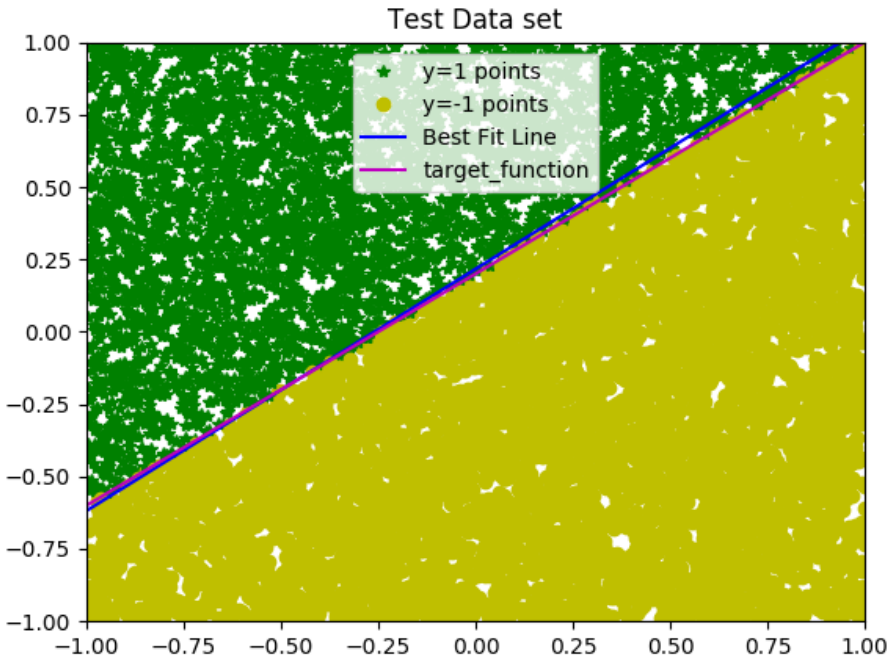

Figure 5: $\eta = 1$ PLA best fit line plotted on the test data

Here for a good learning rate of $\eta = 1$ we see that the PLA successfully converges after around 74 iterations. The error on the randomly generated testing dataset is also very low, i.e 0.8 percent.

The final plot shows the final hypothesis on the testing dataset, for easier illustration of the error encountered.

1.3 Part c: $\eta = 0.01$

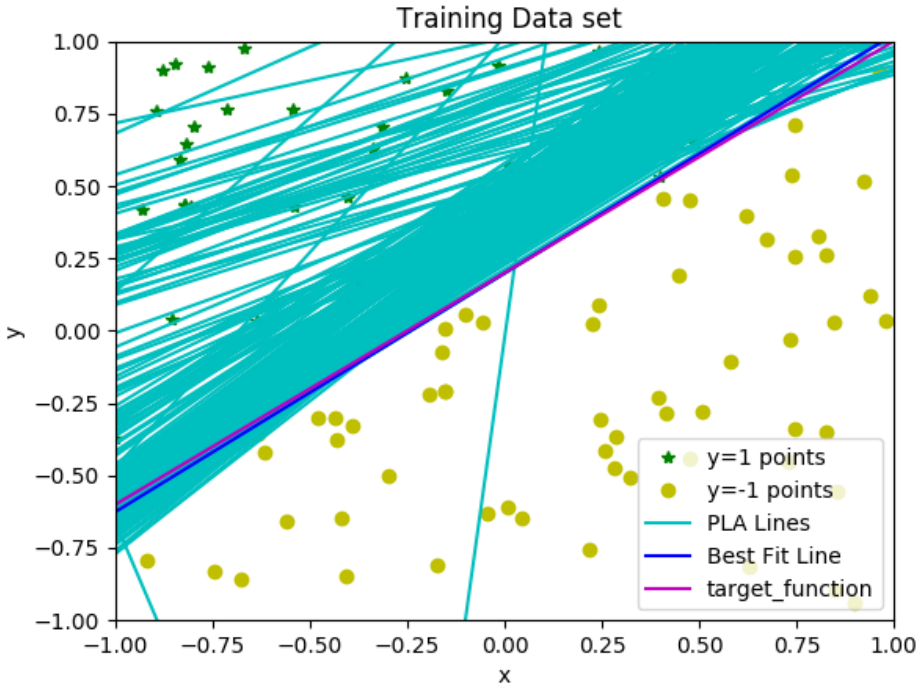

Figure 6: $\eta = 1$ Final hypothesis and target function on training data

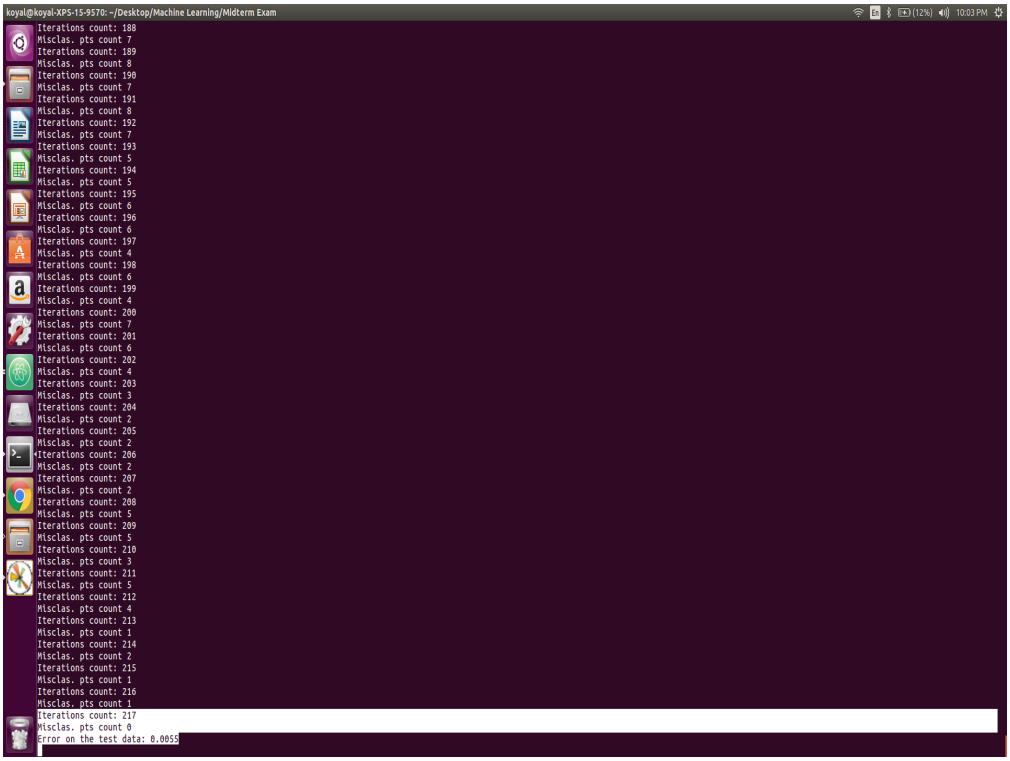

Figure 7: $\eta = 1$ Output terminal showing Iterations and Error on test data

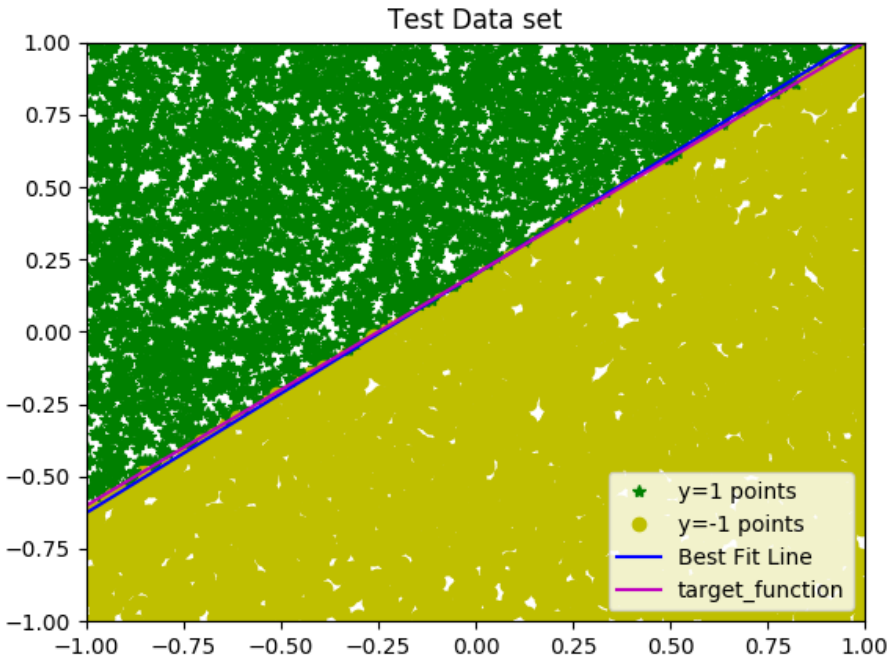

Figure 8: $\eta = 1$ PLA best fit line plotted on the test data

The above plot is for the learning rate of $\eta = 0.01$. We can see that as the rate decreases from 1 to 0.01 the number of iterations has increased from 74 to 217. This is intuitively correct as now we are taking smaller steps to reach the target. The error on the test data has also considerably decreased to about 0.5 percent..!! This is also evident from the plot of the final hypothesis on the test data.

1.4 Part d: $\eta = 0.0001$

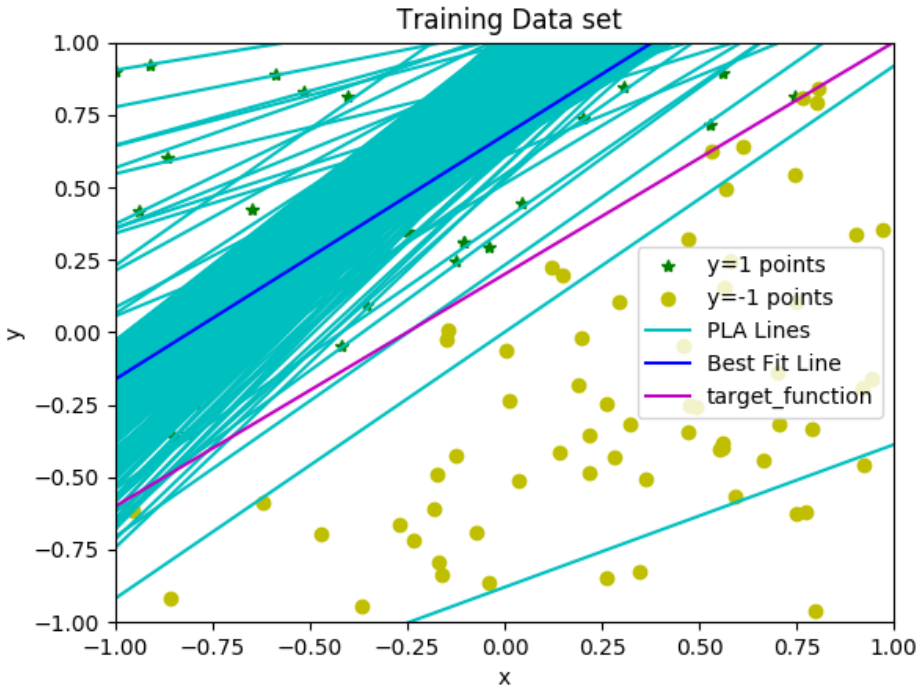

Figure 9: $\eta = 0.0001$ Final hypothesis and target function on training data

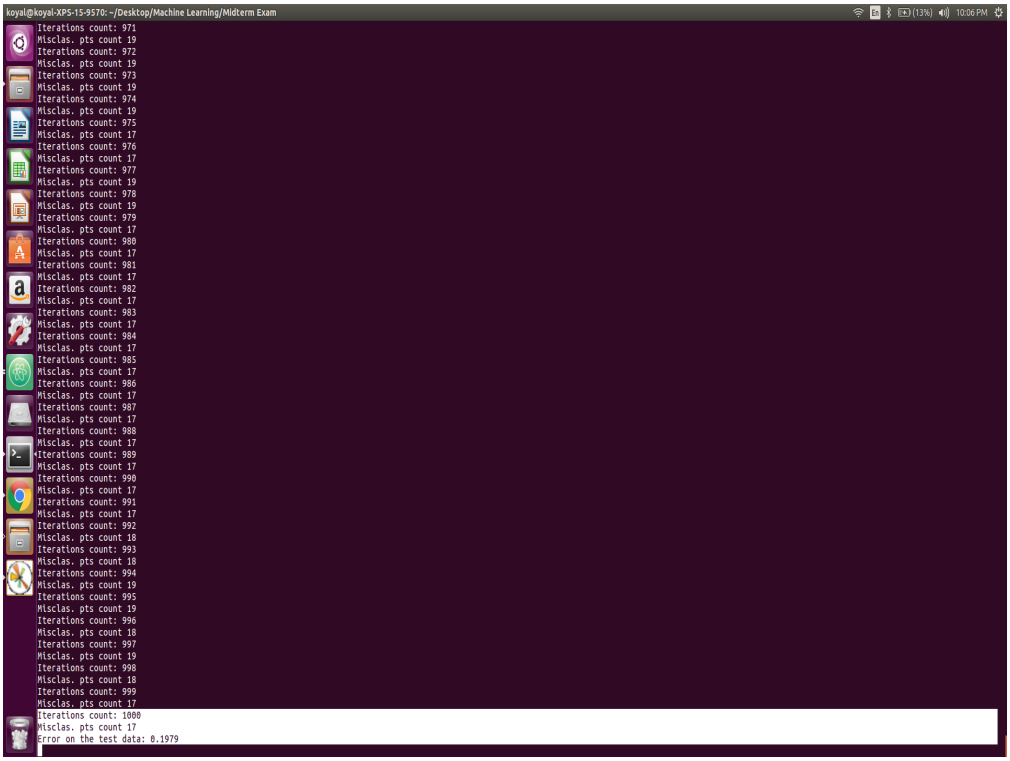

Figure 10: $\eta = 0.0001$ Output terminal showing Iterations and Error on test dat

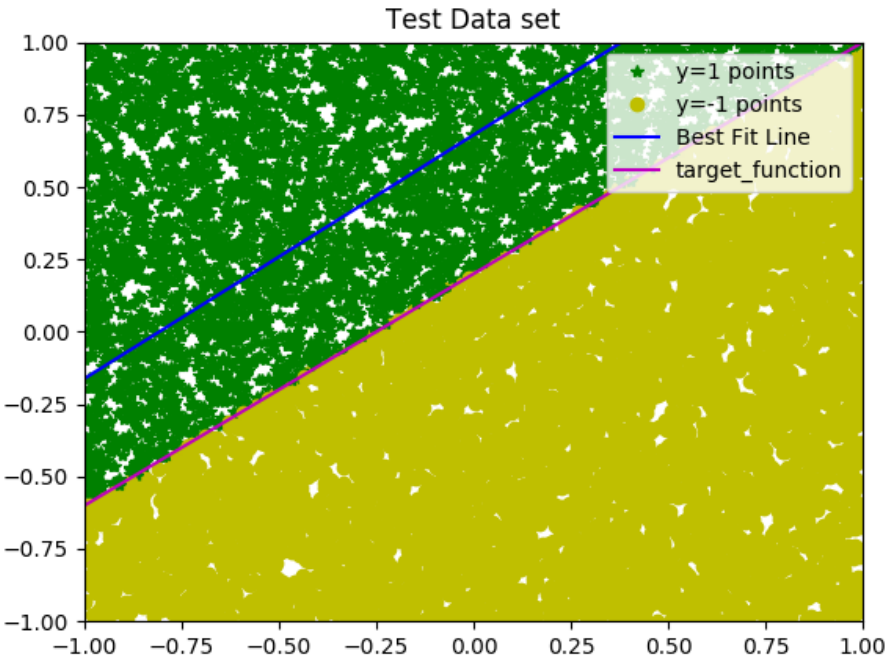

Figure 11: $\eta = 0.0001$ PLA best fit line plotted on the test data

Taking $\eta = 0.0001$ is extremely small. It is the other extrema of taking a very large learning rate, and happens to behave in a similar manner. The algorithm doesnt completely converge even after 1000 updates. As can be seen from the terminal output, 17 misclassified points are still remaining to be classified correctly. The error is also as high as 19 percent..!!

1.5 Part e: Comparision of result and Observations

Learning rate is basically the length of the steps that the algorithm makes down the gradient on the error curve. While training of of the PLA model, we try to determine the minima. The choice of the learning rate helps us determine **how fast** we can reach that minima.
In case of a **very high learning rate**, the algorithm overshoots the optimal point or the minima.
Also in case of a very **lower learning rate**, there is an overshoot though the magnitude is lesser than in the case of a high learning rate and takes a longer time to converge. The values obtained above also show that in case of an overshoot, we get a non-optimal point whose error is be higher.

The following observations is a summary of the analysis above as we change the learning rate:

- Learning rate choice scales w.
- Though the learning rate effects the rate of convergence assuming it's between 0 and 1, it does not affect eventual convergence. Hence in the case of $\eta = 100$ the weight vector immediately became the training case and failed to converge to a solution.
- However, the learning rate affects the speed at which we reach convergence
- A smaller η results in a smaller step size and a better approximation of the true derivative, which in turn improves the ability to locate the optimal point.
- However, as the algorithm approaches the true optimum the error of fit decreases, consequently decreasing the step size and improving the approximation of the derivative. But a large learning rate may hamper this by over-estimating the gradient resulting in the algorithm "oscillating" around the true optimum.

2 Problem 2.5 from LFD

Prove by induction that $\sum_{i=0}^{D} \binom{N}{i} \leq N^D + 1$, hence

$m_H(N) \leq N^{d_{vc}} + 1$

Proof of $\sum_{i=0}^{D} \binom{N}{i} \leq N^D + 1$:

We first test this condition for different values of D:

D=0: $\sum_{i=0}^{0} \binom{N}{i} = 1 \leq 2$; This proves $\sum_{i=0}^{0} \binom{N}{i} \leq N^0 + 1$

D=1: $\sum_{i=0}^{1} \binom{N}{i} = N + 1 \leq N + 1$; This proves $\sum_{i=0}^{1} \binom{N}{i} \leq N^1 + 1$

D=2: $\sum_{i=0}^{2} \binom{N}{i} = \frac{N^2}{2} + \frac{N}{2} + 1 \leq \frac{N^2}{2} + \frac{N^2}{2} + 1 \leq N^2 + 1$

Hence this proves $\sum_{i=0}^{2} \binom{N}{i} \leq N^2 + 1$

Now to prove the further steps of $D > 2$ we first prove a result that will be required i.e:

$$\frac{N!}{(N-D)!} \leq N^D \tag{3}$$

To prove the above by induction: we consider this as known for D=0.

$\frac{N!}{(N-0)!} \leq N^0$ as $1 \leq 1$

Now for $D > 0$ eqn 3 can be written as:

$\frac{N!}{(N-D)!} = \frac{N!}{(N-(D-1))!}(N-(D-1)) \leq N^{D-1}(N-(D-1))$

$N^{D-1}(N-(D-1)) = N^D - (D-1)N^{D-1}$

$(D-1) \geq 0, N^{D-1} \geq 1 \Rightarrow (D-1)N^{D-1} \geq 0 \Leftrightarrow -(D-1)N^{D-1} \leq 0$

This proves:

$N^D - (D-1)N^{D-1} \leq N^D$

Thus: $\frac{N!}{(N-D)!} \leq N^D$**.**

Now using this and going back to the earlier proof to prove the induction step for D¿2:

$$\sum_{i=0}^{D} \binom{N}{i} = \sum_{i=0}^{D-1} \binom{N}{i} + \binom{N}{D} \leq N^{D-1} + 1 + \binom{N}{D} \tag{4}$$

Using eqn 3 proved above in eqn 4 we get:

$$N^{D-1} + 1 + \binom{N}{D} \leq N^{D-1} + 1 + \frac{N^D}{D!} \tag{5}$$

Now it's needed to get rid of the D! in the denominator, the following manipulation is used for the same: For $D > 2 \rightarrow D! > 2 \Rightarrow \frac{1}{D!} < \frac{1}{2}$

Using this eqn 5 becomes:

$$N^{D-1} + 1 + \frac{N^D}{D!} \leq N^{D-1} + 1 + \frac{N^D}{2} \tag{6}$$

Now let's try to manipulate the N^{D+1} term:

Here for $D > 2$ and $N \geq D \rightarrow N > 2 \Leftrightarrow \frac{1}{N} < \frac{1}{2} \Leftrightarrow \frac{N^{D-1}}{N^D} < \frac{1}{2} \Leftrightarrow N^{D-1} < \frac{N^D}{2}$

Usinf this eqn 6 becomes:

$$N^{D-1} + 1 + \frac{N^D}{2} \leq \frac{N^D}{2} + 1 + \frac{N^D}{2} \leq N^D + 1 \tag{7}$$

Thus as seen from eqn4 and eqn7 we have proved that:

$$\sum_{i=0}^{D} \binom{N}{i} \leq N^D + 1 \tag{8}$$

Now to prove the second part of the problem i.e $m_H(N) \leq N^{d_{vc}} + 1$

we use the known formula of a growth function i.e

$$m_H(N) \leq \sum_{i=0}^{k-1} \binom{N}{i} \tag{9}$$

Comparing eqn 9 with eqn 8 we easily get:

$$m_H(N) \leq N^{k-1} + 1 \tag{10}$$

beacuse k-1 in eqn 9 is D in eqn 8

Now using the defination of VC dimension which states that: If d_{VC} is the VC dimension of H, then $k = d_{VC} + 1$ is a break point for m_H since $m_H(N)$ cannot equal 2^N for any $N > d_{VC}$.

Thus using this definition in eqn 10 we get:

$$m_H(N) \leq N^{d_{vc}} + 1 \tag{11}$$

3 Show that the VC dimension of convex d-gons (poygon with d sides) is 2d+1.

Recalling the definition of VC dimension as already stated in the above problem and considering it's relation with the breaking point as $k = d_{VC} + 1$ we move forward.

The proof can be done by taking several cases and examples. An analysis of the examples taken can be generalised to prove that the VC dimension of convex d-gons (poygon with d sides) is 2d+1.

Before proceeding to the specific cases, lets consider an **M interval** problem to get an understanding about the problem at hand:

Case of M interval problem: 1-interval: Breaking point $\rightarrow$ 3. Eg in the points $+ - +$, there are 2 +ves, requiring another interval which is not possible to be shattered by just 1 interval.

2-interval model: Breaking point $\rightarrow$ 5. Eg: $+ - + - +$ we will need at least 3 intervals to shatter it. Hence the breaking point was 5. Also any combination of 4 points i.e $+ - +-$ will needs at most 2 intervals which can easily be shattered.

In the same way for the **3-interval** model, it cannot shatter if there are more than 3 alternately

placed +ve. i.e $+-+-+-+$. Hence the shattering point here was 7.

Hence we see that for a case of any no of intervals if the no of alternate +ve is greater that the interval count(M) it cannot shatter. Thus to generate this no of alternate +ve, we need the same no of -ves i.e M.

Hence the breaking point for **M interval problem was M+M+1 i,e 2M+1.**

Now, let us extend a similar logic in the case of convex d-gons. For the sake of generality and to consider the worst possible case, we take the points around the circumference of a circle. This gives the worst combination possible in each cse of d.

The proof follows as here: we consider 2d+2 points. Let these 2d+2 points contain d+1 positives, and d+1 negatives. This means that there will be d+1 instances of the pattern $+-+$ in order. And obviously each such pattern will require a line to separate it. Also to maintain convexity no line can separate two such patterns. This shows that we need at least d+1 lines. Hence this proves that at least d+1 lines is required to shatter 2d+2 points; therefore at least d lines will be needed to shatter 2d+1 points.

Further analysis on 3 types of polygon is shown below:

Consider first the polygon of 3 sides i.e d=3

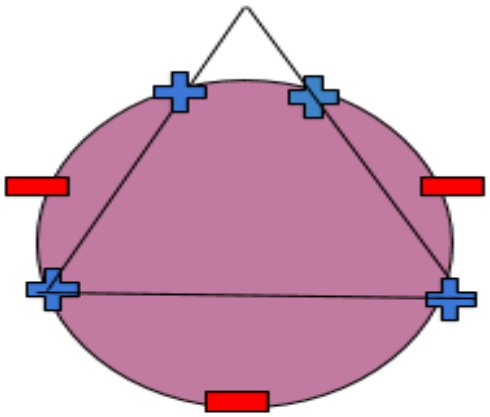

Figure 12: Convex-3-gon shattering 7 points

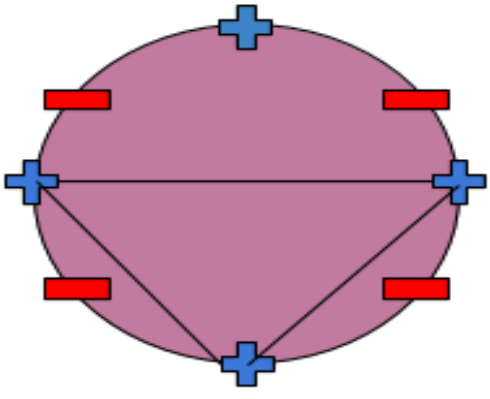

Figure 13: Convex-3-gon unable to shattering 8 points

In the above 2 cases where we consider a polygon of **side 3**, we consider 2 cases: 1 with no of points as 7 and another with the no of points as 8. We see that it is able to shatter 7 points, but failing to shatter 8 points. It is also intuitive to noitce that the triangle will be able to shatter all the combination of +ve and -ve on the 7 points at hand Hence the breaking point is 8; or the VC dimension is 7. **Obs: 2(3)+1=7**

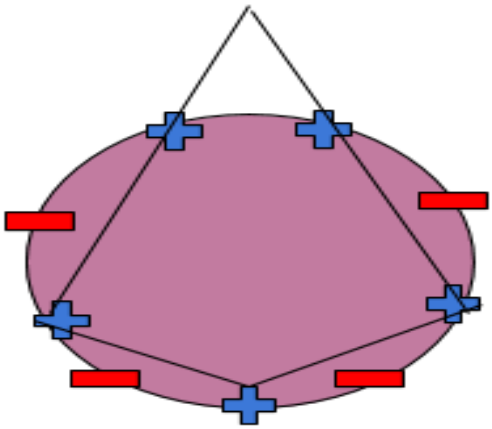

Figure 14: Convex-4-gon shattering 9 points

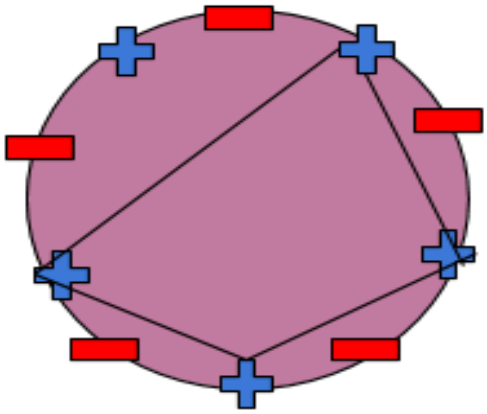

Figure 15: Convex-4-gon unable to shattering 10 points

Similarly when we consider a polygon of **side 4**, we consider 2 cases: 1 with no of points as 9 and another with the no of points as 10. We see that it is able to shatter 9 points, and all of its combinations of patterns; but fails to shatter 10 points. Hence the breaking point is 10; or the VC dimension is 9. **Obs: 2(4)+1=9**

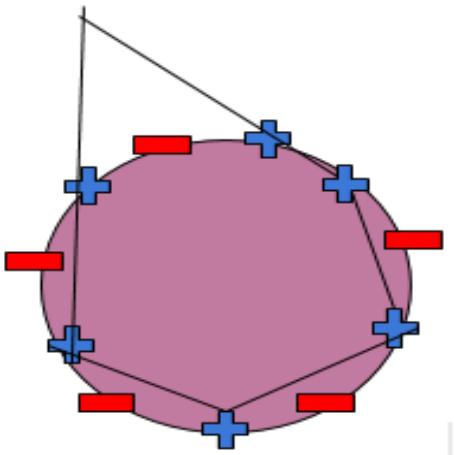

Figure 16: Convex-5-gon shattering 11 points

On similar lines for a polygon of **side 5**, we see that it is able to shatter 11 points, but failing to shatter 11 points; Hence the VC dimension is 11. **Obs: 2(5)+1=11**

On the basis of the analysis shown above, it can be safely generalised that the VC dimension of a convex polygon of d sides is 2d+1.

4 Problem 3.1 from LFD

In this problem 2 semicircles have been considered as shown: of the same radius and thickness, separated by a distance and having different center points. In the circles shown red is -1 and blue is +1.

If the separation between the circles is lesser than 0, then their would be a high degree of overlap

between the +ve and -ve points, and hence the task would not be linearly separable. Hence here we take the separatio sep between the circles as $\geq= 0$. The values taken for the purpose of the problem are: rad=10, thk=5 and sep=5. A total of 2000 training data set is considered, with around 1000 points in each of the 2 semi-circles.

4.1 Part a

Here the PLA is run starting from w=0. The PLA is repeatedly iterated, until it converges. As shown in the figure below, the light blue lines indicate the PLA lines drawn in each iteration which gets followed by an update of the weights. The weight update follows the following condition: $y(t).s(t) \leq 0$, where the update happens by:

$$w(t+1) \leftarrow w(t) + y(t).x(t) \tag{12}$$

where $s(t) = w^T(t)x(t)$ The final hypotheses obtained is shown in dark blue shade. The final hypothesis is obtained when the number of misclassified points reduces to 0. The total number of iterations required for the convergence is recorded. Also as seen from the terminal output, the no if iterations it takes to converge is aprroximately 16. This is a random wrt to the random points generated. The no of iterations it takes to converge at times even goes as low as 8 or 6. Here we see that the PLA runs compartaively much faster in the case of a training data set as taken than the general case of a random dataset taken in the earlier problems.

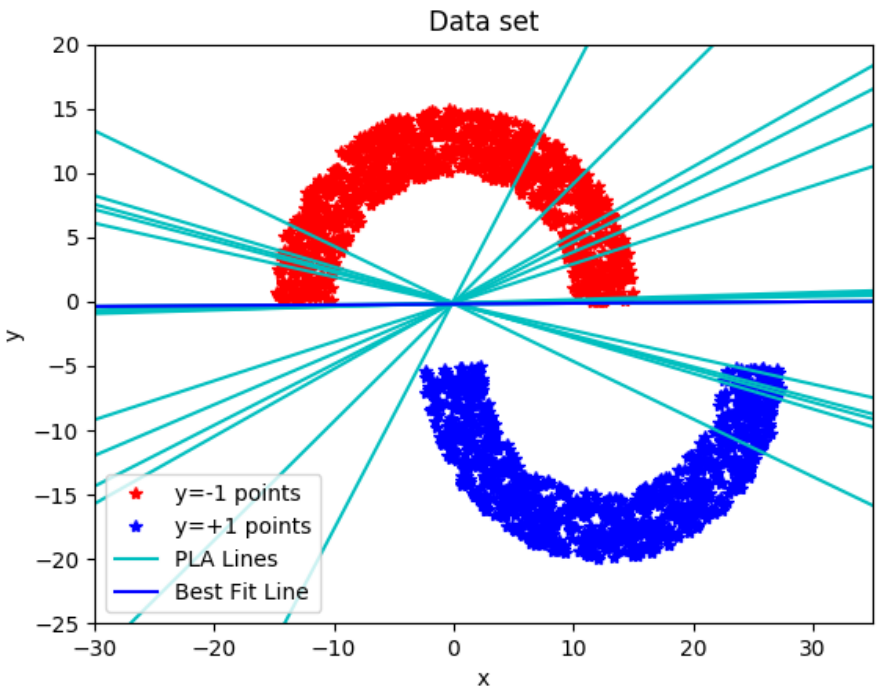

Figure 17: Semi-Circle Training data along with final hypotheses

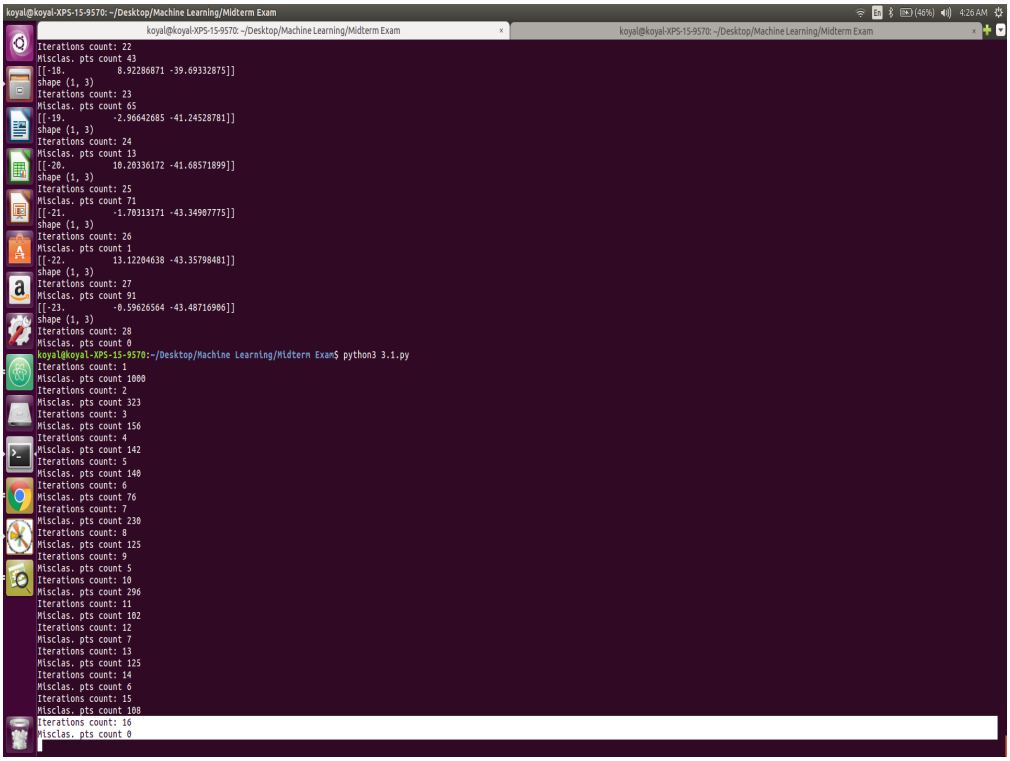

Figure 18: Semi-Circle: Terminal Output - No. of misclassified points and iterations

4.2 Part b

Here we use linear regression for the classification process. It is implemented by taking a random target function and accordingly classifying the the randomly generated data points.

The initial weights is got by taking pseudo inverse of the data set as shown below:

$$W = (X^T X)^{-1} X^T Y \tag{13}$$

where X=(1,x1,x2). Hence we get w(weights) as a (3x1) matrix.

The goal of linear regression is to find a target function that can minimize the error, which can be measured as the sum of absolute or squared error.

$$AbsoluteError = \sum_i |y_i - f(x_i| \tag{14}$$

$$SquaredError = \sum_i (y_i - f(x_i|)^2 \tag{15}$$

As seen from the figure below, it is clearly niticible that when Linear Regression is used in this task, it tries to find the final hypothesis in such a way that the total squared error is a minimum. And in fact it is doing the job pretty well, and the classification also appears to be better than the earlier case when the initial weights were considered as 0.

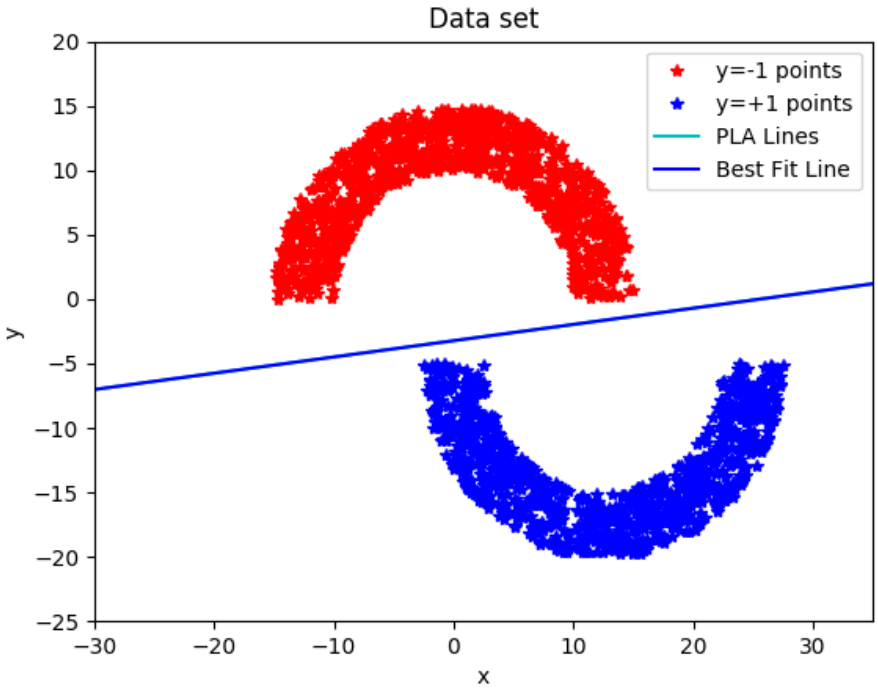

Figure 19: PLA best fit line of Linear Regression on Semi-Circle

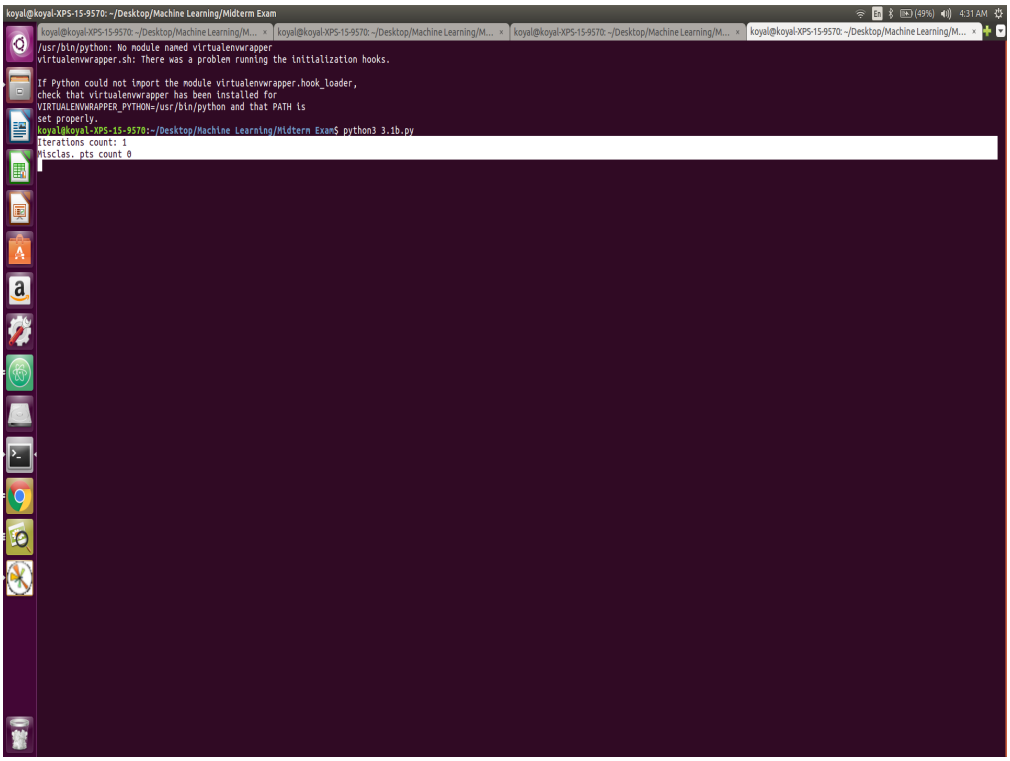

Figure 20: Terminal Output of iterations of Linear Regression on Semi-Circle

5 Problem 3.2 from LFD

Here as the no of separation between the semi-circles is increased, we notice that the no if iterations tends to decrease. This is because as the the weights gets more aligned with the optimal expected weight, the no of iterations required to converge is lesser. That is also the reason behind the ups and downs seen in the graph output of the no. of iterations vs sep. If a learning rate is also provided in the process of updation of the weights, the jumps will be lesser. This result can also be theoretically proven. If $\rho = min_{(1 \leq n \leq N)} y_n(w^{*T} x_n)$ and $R = max_{(1 \leq n \leq N)} ||x_n||$, the theoretical bound on PLA can be given be:

$$t \leq \frac{R^2 ||w^*||^2}{\rho^2} \tag{16}$$

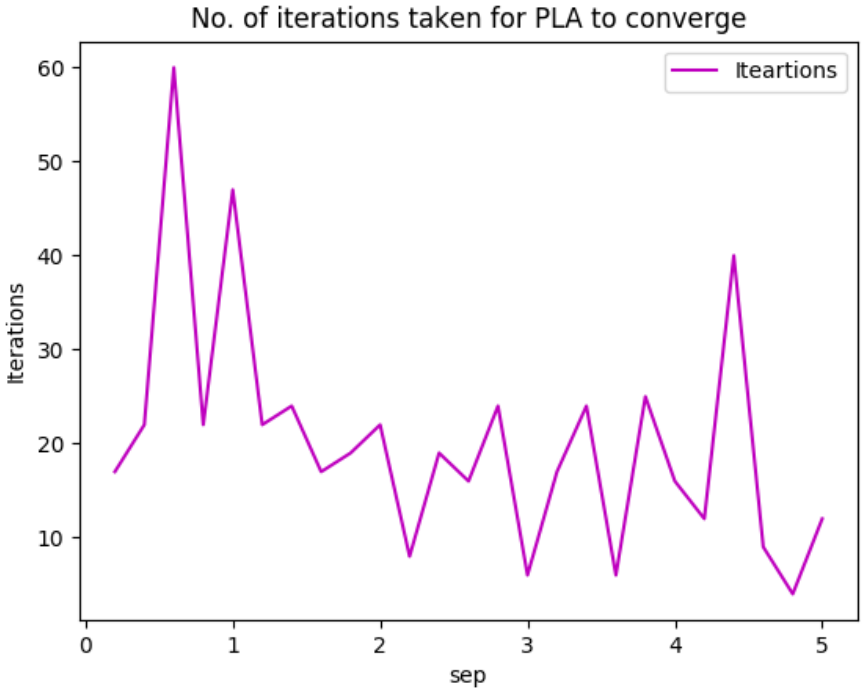

Figure 21: Graph of No. of Iterations vs sep of semi circle

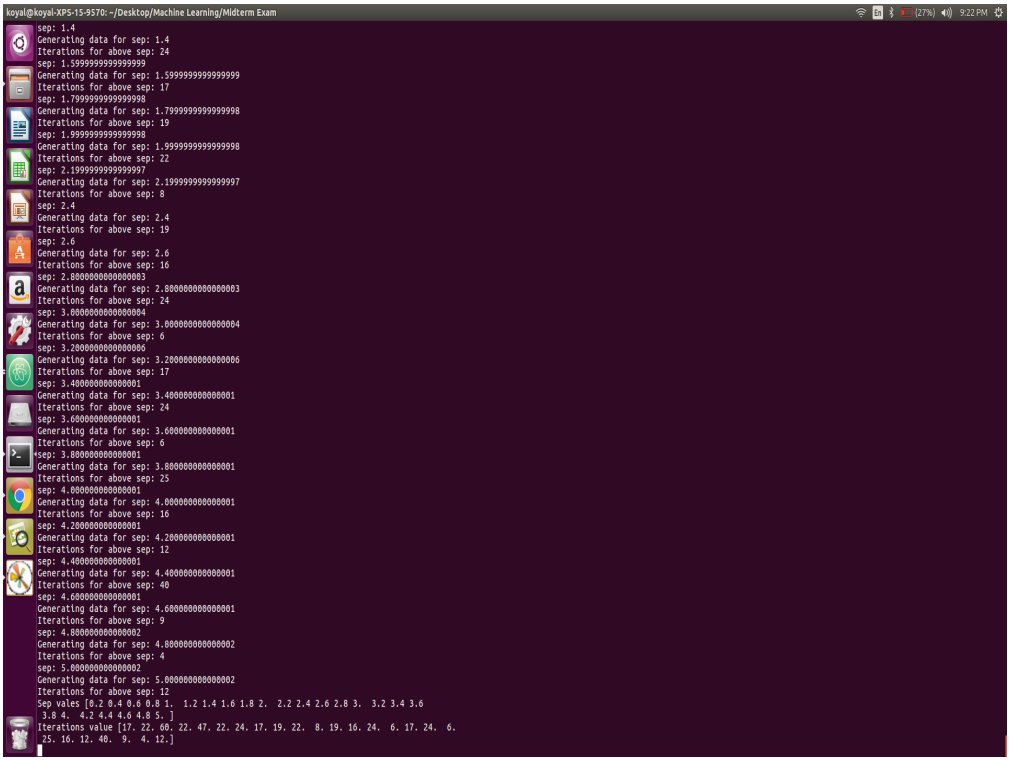

Figure 22: Output terminal of iterations of PLA on semi circle

Following is the output illustrating the variation of theoretical bound with the separation between the circles: As seen the trend of the the no of iterations varying with sep agrees with the variation of the theoretical bound on the iterations wrt sep.

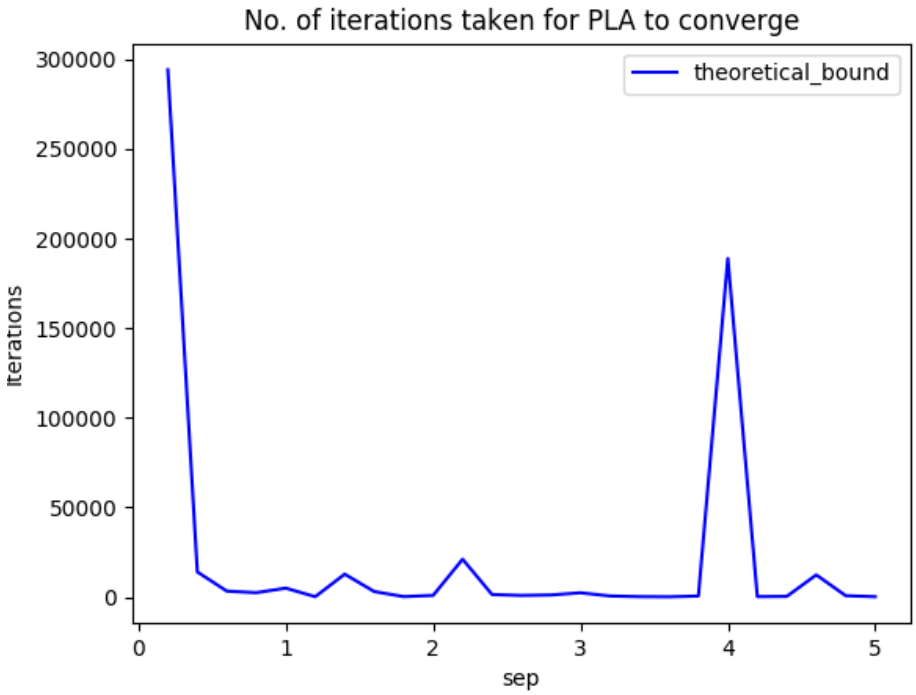

Figure 23: Graph of Theoretical Bound vs sep of semi circle

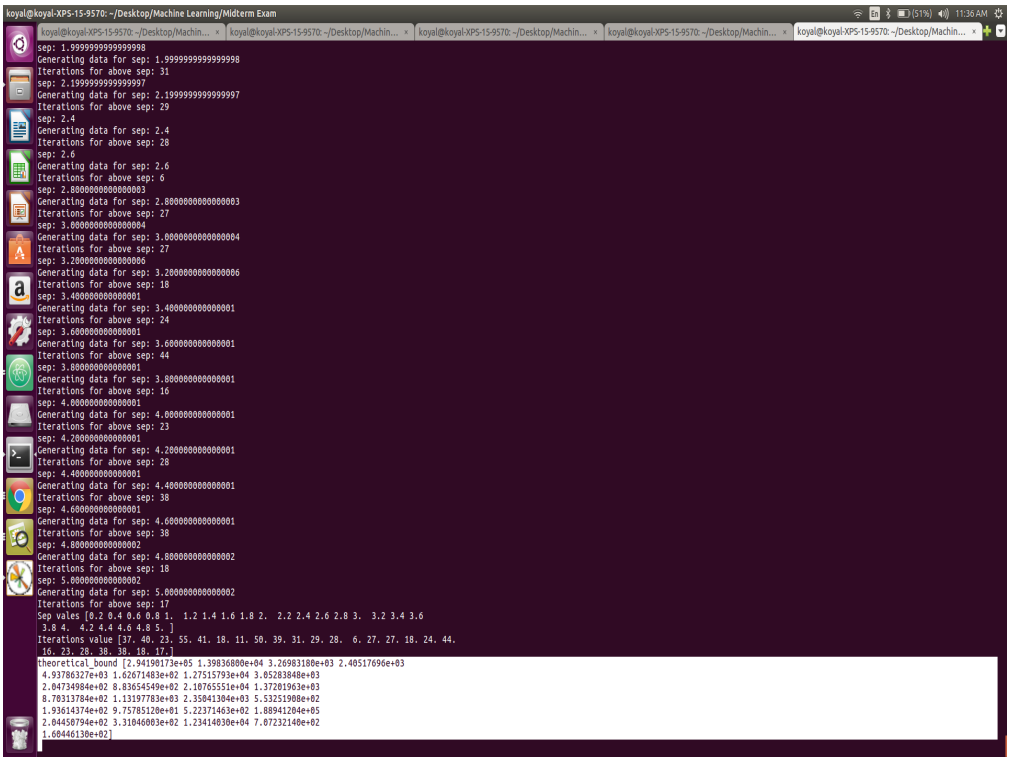

Figure 24: Output terminal of theoretical bound using PLA semi circle

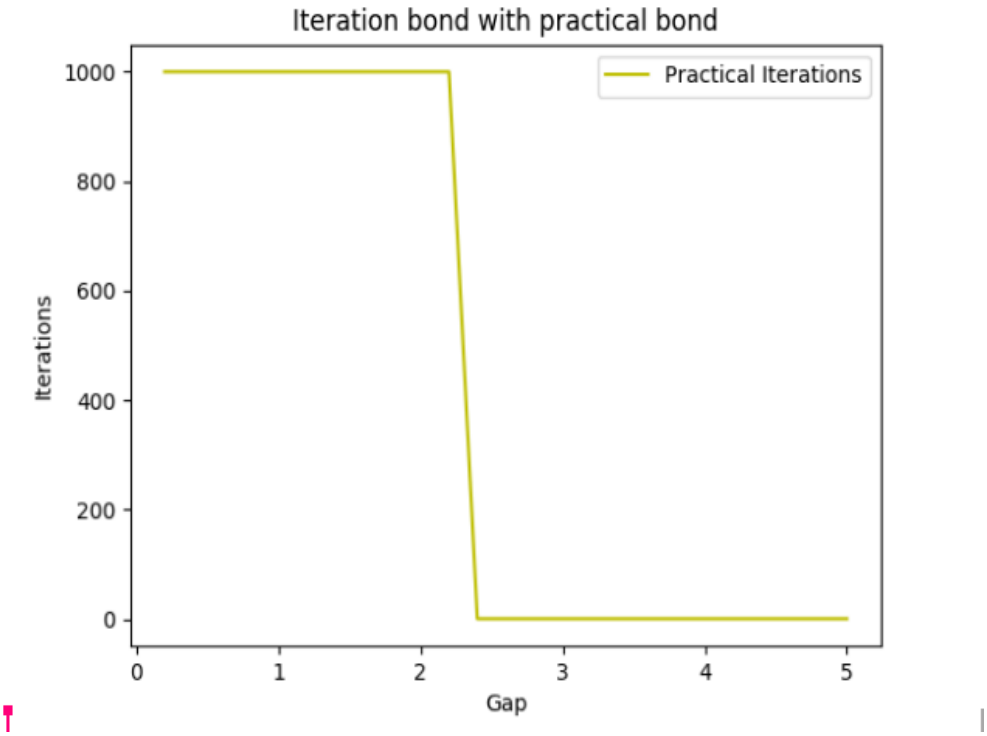

Figure 25: Graph of Iteration vs sep

Also, if Linear Regression is run on the varying values of sep, as noticed in the previous case, the no if iterations remains 1 as it tries to minimize the square error. The theoretical bound shown above is the bound for all types of hypothesis.

Note: Codes for each of the questions is attached below for reference.

6 Codes 1(Problem 1.5 from LFD)

```
#
#  Copyright 2019 Koyal Bhartia
#  @file     AdalineAlgorithm.py
#  @author   Koyal Bhartia
#  @date     03/30/2019
#  @version 1.0
#
#  @brief This is the code for Problem 1.5 from "Learning from Data"
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random

#Generation of random data
def generate_data(N):
    X_data=np.column_stack((np.ones(N),np.ones(N))) # X1,X2 data
    fx=np.zeros((N)) #Corresponding y
    for i in range(len(X_data)):
        X_data[i,0]=random.uniform(-1,1)
        X_data[i,1]=random.uniform(-1,1)
        if (X_data[i,0]*t_slope+t_intercept<X_data[i,1]):
            a=i
            fx[i]=1
        else:
            b=i
            fx[i]=-1
    return X_data,fx,a,b

def plotdata(X_data,fx,a,b):
    for i in range(0,len(X_data)):
        if(fx[i]==1):
            plt.plot(X_data[i,0],X_data[i,1],'*g')
        else:
            plt.plot(X_data[i,0],X_data[i,1],'oy')
```

```
    plt.plot(X_data[a,0],X_data[a,1],'*g', label='y=1 points')
    plt.plot(X_data[b,0],X_data[b,1],'oy', label='y=-1 points')


def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

#Checks for the misclassification of points for each hypothesis
def misclassified(X_data,fx,w):
    break_point=1 # Flag to keep track if all points are classified correctly
    misclassify=[]
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    for i in range(0,len(X)):
        mul=w*X.transpose()
        sign=signCheck(float(mul[0,i]))
        # Check for misclassification
        if(sign!=fx[i]):
            misclassify=np.append([misclassify],[i])
    if(len(misclassify)==0):
        break_point = 0
        a=-5
        length=0
    if(break_point == 1):
        point=random.randint(1,len(misclassify))
        a=int(misclassify[point-1])
    print("Misclas. pts count",len(misclassify))
    length=len(misclassify)
    return a,break_point,length,mul

# Runs the PLA model
def PLA(w):
    count=0
    break_point=1
    X_data,fx,a,b=generate_data(N_train)
    plotdata(X_data, fx,a,b)
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    while(break_point==1 and count<1000):
        count=count+1
        print("Iterations count:",count)
```

```
        index,break_point,length,mul=misclassified(X_data,fx,w)
        if(index!=-5):
            randd=random.randint(0,99)
            if(mul[0,randd]*fx[randd]<=1):
                w_new=np.mat([0,0,0])
                w_new= w + eta*X[randd,:]*(fx[randd]-mul[0,randd]) #Adaline algorithm
                w=w_new
            m=float(-w_new[0,1]/w_new[0,2])
            c=float(-w_new[0,0]/w_new[0,2])
            x_line1=np.mat([[-1],[1]])
            y_line1=m*x_line1+c
            plt.plot(x_line1,y_line1,'-c')
            plt.xlabel('x')
            plt.ylabel('y')
            plt.axis([-1,1,-1,1])
            plt.title('Data_set')
    if count==1:
        w_new=w
    return w_new

if __name__ == '__main__':
    N_train=100
    N_test=10000
    N_trials=1000
    t_slope=0.8
    t_intercept=0.2
    eta=1
    w=np.mat([0,0,0])
    w_new=PLA(w)
    # Check on the test data
    X_data,fx,a,b=generate_data(N_test)
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    misclassify=0
    for i in range(0,len(X)):
        mul=w_new*X.transpose()
        sign=signCheck(float(mul[0,i]))
        if(sign!=fx[i]):
            misclassify+=1
    print("Error_on_the_test_data:",misclassify/10000)

    #Plot for thte purpose of creating the legend
    m=float(-w_new[0,1]/w_new[0,2])
    c=float(-w_new[0,0]/w_new[0,2])
```

```
x_line1=np.mat([[-1],[1]])
y_line1=m*x_line1+c
x_linet=x_line1
y_linet=t_slope*x_linet+t_intercept
plt.plot(x_line1,y_line1,'-c',label='PLA Lines')
plt.plot(x_line1,y_line1,'-b',label='Best Fit Line')
plt.plot(x_linet,y_linet,'-m',label='target_function')
plt.xlabel('x')
plt.ylabel('y')
plt.axis([-1,1,-1,1])
plt.title(' Training Data set')
plt.legend()
plt.show()

#Plot the test data
plotdata(X_data, fx,a,b)

#Plot the best fit line and target function on the test data
plt.plot(x_line1,y_line1,'-b',label='Best Fit Line')
plt.plot(x_linet,y_linet,'-m',label='target_function')
plt.axis([-1,1,-1,1])
plt.title('Test Data set')
plt.legend()
plt.show()
```

7 Codes 2(Problem 3.1 Part(a) from LFD)

```
#
#  Copyright 2019 Koyal Bhartia
#  @file     Semi-Cicle.py
#  @author   Koyal Bhartia
#  @date     03/30/2019
#  @version 1.0
#
#  @brief This is the code for Problem 3.1 from "Learning from Data"
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random


def generate_data():
    c1in=0
    c1out=0
    c2in=0
    c2out=0
    # X1, X2 data
    X_data=np.column_stack((np.ones(Total),np.ones(Total)))
    # fx to store classification value
    fx=np.zeros((Total))
    count1=0
    count2=0
    index=0
    while(count1<=999 or count2<=999):
            x1=random.uniform(-maxhori/2,maxhori/2+thk)
            x2=random.uniform(-maxverti/2-sep,maxverti/2)

            c1out=math.pow(x1,2)+math.pow(x2,2)-math.pow(rad+thk,2)
            c1in=math.pow(x1,2)+math.pow(x2,2)-math.pow(rad,2)

            c2in=math.pow(x1-(rad+thk/2),2)+math.pow(x2+sep,2)-math.pow(rad,2)
            c2out=math.pow(x1-(rad+thk/2),2)+math.pow(x2+sep,2)-math.pow(rad+thk,2)
```

```
            if(x2>=0 and c1in>=0 and c1out<=0 and count1<=999):
                X_data[index,0]=x1
                X_data[index,1]=x2
                a=index
                fx[index]=-1
                count1+=1
                index+=1
                plt.plot(x1,x2,'*r')
            if(x2+sep<=0 and c2in>=0 and c2out<=0 and count2<=999):
                X_data[index,0]=x1
                X_data[index,1]=x2
                b=index
                fx[index]=+1
                count2+=1
                index+=1
                plt.plot(x1,x2,'*b')

    plt.plot(X_data[a,0],X_data[a,1],'*r', label='y=-1 points')
    plt.plot(X_data[b,0],X_data[b,1],'*b', label='y=+1 points')
    return X_data,fx

def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

def misclassified(X_data,fx,w):
    break_point=1
    misclassify=[]
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    for i in range(0,len(X)):
        mul=w*X.transpose()
        sign=signCheck(float(mul[0,i]))
        if(sign!=fx[i]):
            misclassify=np.append([misclassify],[i])
    if(len(misclassify)==0):
        break_point = 0
        a=-5
        length=0
    if(break_point == 1):
        point=random.randint(1,len(misclassify))
        a=int(misclassify[point-1])
```

```
    print("Misclas. pts count",len(misclassify))
    length=len(misclassify)
    return a,break_point,length,mul

def PLA(w):
    count=0
    break_point=1
    X_data,fx=generate_data()
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    while(break_point==1):
        count=count+1
        print("Iterations count:",count)
        pos,break_point,length,mul=misclassified(X_data,fx,w)
        if(pos!=-5):
            w_new=np.mat([0,0,0])
            w_new= w + fx[pos]*X[pos,:]
            w=w_new
            m=float(-w_new[0,1]/w_new[0,2])
            c=float(-w_new[0,0]/w_new[0,2])
            x_line1=np.mat([[-100],[100]])
            y_line1=m*x_line1+c
            plt.plot(x_line1,y_line1,'-c')
            plt.xlabel('x')
            plt.ylabel('y')
            plt.axis([-maxhori/2,maxhori/2+thk,-maxverti/2-sep,maxverti/2])
            plt.title('Data set')
    if count==1:
        w_new=w
    return w_new

if __name__ == '__main__':

    centerx1,centerx2=0,0
    rad=10
    thk=5
    sep=5
    Total=2000
    Class=1000
    maxhori=4*(rad+thk)
    maxverti=2*(rad+thk+sep)
    w=np.mat([0,0,0])
    w_new=PLA(w)
```

```
#Plot the best fit line and target function on the test data
m=float(-w_new[0,1]/w_new[0,2])
c=float(-w_new[0,0]/w_new[0,2])
x_line1=np.mat([[-100],[100]])
y_line1=m*x_line1+c
plt.plot(x_line1,y_line1,'-c',label='PLA Lines')
plt.plot(x_line1,y_line1,'-b',label='Best Fit Line')
plt.xlabel('x')
plt.ylabel('y')
plt.axis([-maxhori/2,maxhori/2+thk,-maxverti/2-sep,maxverti/2])
plt.title('Data set')
plt.legend()
plt.show()
```

8 Codes 3(Problem 3.1 Part(b) from LFD)

```
#
#  Copyright 2019 Koyal Bhartia
#  @file     Semi-Cicle.py
#  @author   Koyal Bhartia
#  @date     03/30/2019
#  @version 1.0
#
#  @brief This is the code for Problem 3.1b from "Learning from Data"
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random

def generate_data():
    c1in=0
    c1out=0
    c2in=0
    c2out=0

    X_data=np.column_stack((np.ones(N),np.ones(N)))
    fx=np.zeros((N))

    count1=0
    count2=0
    index=0
    while(count1<=999 or count2<=999):
            x1=random.uniform(-maxhori/2,maxhori/2)
            x2=random.uniform(-maxverti/2,maxverti/2)

            c1out=math.pow(x1,2)+math.pow(x2,2)-math.pow(rad+thk,2)
            c1in=math.pow(x1,2)+math.pow(x2,2)-math.pow(rad,2)

            c2in=math.pow(x1-(rad+thk/2),2)+math.pow(x2+sep,2)-math.pow(rad,2)
            c2out=math.pow(x1-(rad+thk/2),2)+math.pow(x2+sep,2)-math.pow(rad+thk,2)

            if(x2>=0 and c1in>=0 and c1out<=0 and count1<=999):
```

```
                X_data[index,0]=x1
                X_data[index,1]=x2
                a=index
                fx[index]=-1
                count1+=1
                index+=1
                plt.plot(x1,x2,'*r')
            if(x2+sep<=0 and c2in>=0 and c2out<=0 and count2<=999):
                X_data[index,0]=x1
                X_data[index,1]=x2
                b=index
                fx[index]=+1
                count2+=1
                index+=1
                plt.plot(x1,x2,'*b')

    plt.plot(X_data[a,0],X_data[a,1],'*r', label='y=-1 points')
    plt.plot(X_data[b,0],X_data[b,1],'*b', label='y=+1 points')
    return X_data,fx

def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

def misclassified(X_data,fx,w):
    break_point=1
    misclassify=[]
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    for i in range(0,len(X)):
        mul=np.matmul(w,X.transpose())
        sign=signCheck((mul[0,i]))
        if(sign!=(fx[i])):
            misclassify=np.append([misclassify],[i])
    if(len(misclassify)==0):
        break_point = 0
        a=-5
        length=0
    if(break_point == 1):
        point=random.randint(1,len(misclassify))
        a=int(misclassify[point-1])
    print("Misclas. pts count",len(misclassify))
```

```
    length=len(misclassify)
    return a,break_point,length,mul

def PLA():
    mis_counter=0
    mis_avg=0
    count=0
    break_point=1
    X_data,fx=generate_data()
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    w0=np.matmul(np.transpose(X),X)
    w1=LA.inv(w0)
    w2=np.matmul(w1,np.transpose(X))
    w3=np.matmul(w2,fx)
    w=np.array([w3])
    count=0
    while(break_point==1 and count<1000):
        count=count+1
        print("Iterations count:",count)
        pos,break_point,length,mul=misclassified(X_data,fx,w)
        if(pos!=-5):
            w_new=np.mat([0,0,0])
            w_new= w + fx[pos]*X[pos,:]
            #print(w_new)
            w=w_new
            print(w_new)
            #print("shape",np.shape(w_new))
            m=float(-w_new[0,1]/w_new[0,2])
            c=float(-w_new[0,0]/w_new[0,2])
            x_line1=np.mat([[-100],[100]])
            y_line1=m*x_line1+c
            plt.plot(x_line1,y_line1,'-c')
            plt.xlabel('x')
            plt.ylabel('y')
            plt.axis([-maxhori/2,maxhori/2,-maxverti/2,maxverti/2])
            plt.title('Data set')
    if count==1:
        w_new=w
    return w_new

if __name__ == '__main__':

    centerx1,centerx2=0,0
```

```
rad=10
thk=5
sep=5
Total=2000
Class=1000
N=2000
leftfit=rad+thk+1
rightfit=rad+(thk)/2+rad+thk+1
downfit=sep+rad+thk+1
upfit=rad+thk+1
maxhori=4*(rad+thk)
maxverti=2*(rad+thk+sep)

w_new=PLA()
m=float(-w_new[0,1]/w_new[0,2])
c=float(-w_new[0,0]/w_new[0,2])
x_line1=np.mat([[-100],[100]])
y_line1=m*x_line1+c
#x_linet=x_line1
#y_linet=t_slope*x_linet+t_intercept
plt.plot(x_line1,y_line1,'-c',label='PLA Lines')
plt.plot(x_line1,y_line1,'-b',label='Best Fit Line')
#plt.plot(x_linet,y_linet,'-m',label='target_function')
plt.xlabel('x')
plt.ylabel('y')
plt.axis([-maxhori/2,maxhori/2+thk,-maxverti/2-sep,maxverti/2])
plt.title('Data set')
plt.legend()
plt.show()
```

9 Codes 3(Problem 3.2from LFD)

```
#
#  Copyright 2019 Koyal Bhartia
#  @file     AdalineAlgorithm.py
#  @author   Koyal Bhartia
#  @date     03/30/2019
#  @version 1.0
#
#  @brief This is the code for Problem 3.2 from "Learning from Data"
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random

def generate_data(sep):
    print("Generating data for sep:",sep)
    maxhori=4*(rad+thk)
    maxverti=2*(rad+thk+sep)
    c1in=0
    c1out=0
    c2in=0
    c2out=0

    X_data=np.column_stack((np.ones(N),np.ones(N)))
    fx=np.zeros((N))

    count1=0
    count2=0
    index=0
    while(count1<=999 or count2<=999):
            x1=random.uniform(-maxhori/2,maxhori/2+thk)
            x2=random.uniform(-maxverti/2-sep,maxverti/2)

            c1out=math.pow(x1,2)+math.pow(x2,2)-math.pow(rad+thk,2)
            c1in=math.pow(x1,2)+math.pow(x2,2)-math.pow(rad,2)

            c2in=math.pow(x1-(rad+thk/2),2)+math.pow(x2+sep,2)-math.pow(rad,2)
```

```
            c2out=math.pow(x1-(rad+thk/2),2)+math.pow(x2+sep,2)-math.pow(rad+thk,2)

            if(x2>=0 and c1in>=0 and c1out<=0 and count1<=999):
                X_data[index,0]=x1
                X_data[index,1]=x2
                a=index
                fx[index]=-1
                count1+=1
                index+=1
            if(x2+sep<=0 and c2in>=0 and c2out<=0 and count2<=999):
                X_data[index,0]=x1
                X_data[index,1]=x2
                b=index
                fx[index]=+1
                count2+=1
                index+=1
    return X_data,fx

def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

def misclassified(X_data,fx,w):
    break_point=1
    misclassify=[]
    X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
    for i in range(0,len(X)):
        mul=w*X.transpose()
        sign=signCheck((mul[0,i]))
        if(sign!=(fx[i])):
            misclassify=np.append([misclassify],[i])
    if(len(misclassify)==0):
        break_point = 0
        a=-5
        length=0
    if(break_point == 1):
        point=random.randint(1,len(misclassify))
        a=int(misclassify[point-1])
    #print("Misclas. pts count",len(misclassify))
    return a,break_point
```

```
def PLA(w):
    Iterations=[]
    sep=[]
    theoretical_bound=[]
    sepvalue=0.2
    while(sepvalue<=5.1):
        w=np.mat([0,0,0])
        print("sep:",sepvalue)
        X_data,fx=generate_data(sepvalue)
        X=np.column_stack((np.ones(len(X_data),dtype=int),X_data[:,0],X_data[:,1]))
        count=0
        break_point=1
        while(break_point==1):
            count=count+1
            #print("Iterations  count:",count)
            pos,break_point=misclassified(X_data,fx,w)
            if(pos!=-5):
                w_new=np.mat([0,0,0])
                w_new= w + fx[pos]*X[pos,:]
                w=w_new
        print("Iterations for above sep:",count)
        #pho=np.min(fx.transpose()*(w*X.transpose()))
        fx=np.mat([fx])

        pho=np.min(np.multiply(fx.transpose(),np.matmul(X,w.transpose())))
        R=np.max(X[:,0]+X[:,0]+X[:,0])
        w_sum=np.sum(w)
        bound=(R*R*w_sum*w_sum)/(pho*pho)
        #print(np.shape(np.matmul(X,w.transpose())),'shape  of  wx')

        #print(pho)
        #print(np.shape(R))
        theoretical_bound=np.append([theoretical_bound],[bound])
        Iterations=np.append([Iterations],[count])
        sep=np.append([sep],[sepvalue])
        sepvalue+=0.2
    return sep,Iterations,theoretical_bound

if __name__ == '__main__':
    centerx1,centerx2=0,0
    rad=10
    thk=5
    Total=2000
```

```
Class=1000
N=2000
w=np.mat([0,0,0])
sep,Iterations,theoretical_bound=PLA(w)
print("Sep vales",sep)
print("Iterations value",Iterations)
print("theoretical_bound",theoretical_bound)

plt.plot(sep,Iterations,'-m',label='Iteartions')
plt.xlabel('sep')
plt.ylabel('Iterations')
plt.title('No. of iterations taken for PLA to converge')
plt.legend()
plt.show()
plt.plot(sep,theoretical_bound,'-b',label='theoretical_bound')
plt.xlabel('sep')
plt.ylabel('Iterations')
plt.title('No. of iterations taken for PLA to converge')
plt.legend()
plt.show()
```