EE 525 – Assignment 2

Tanmay Gore - Tgore03@iastate.edu

Q.1

a $f(x) = \text{Heaviside}(x_{i_1} \cdot x_{i_2} \cdots x_{i_K} - 1)$

$f(x)$ is positive only when all x_i's are 1 else $f(x)$ is negative

b. $f(x) = \text{Heaviside}(x_{i_1} + x_{i_2} + \cdots x_{i_K} - \frac{k}{2})$

if sum of feature is.

$>$ half of k $\rightarrow$ $f(x) = 1$

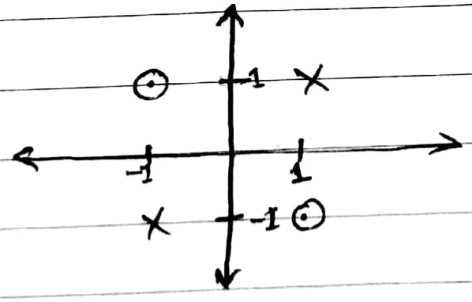
$<$ half of k $\rightarrow$ $f(x) = -1$

$=$ half of k $\rightarrow$ $f(x) = 0.5$

$\rightarrow$ we can bias it towards negative

Q.2

a



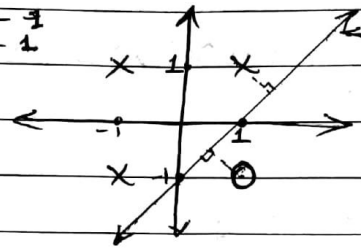
Since the 4 point lie in 4 different ~~decision~~ quadrant, a single perceptron cannot be used for linear classification.

Q-2

b.

(i)

x	y
-1	0
1	0



(0,-1)

← hyperplane passes through

slope = 1

$y = mx + c$

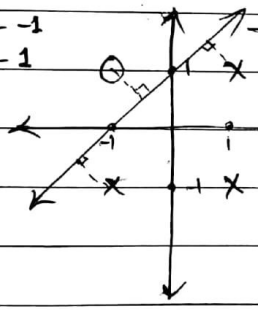
$\therefore y + 1 = mx$ $(c = -1)$

$\therefore x - y - 1 = 0$ $(m = 1)$

here. $\omega_1 = 1$, $\omega_2 = -1$, $b = -1$

(ii)

x	y
-1	0
1	0



→ seperating hyper plane passing through (0,1)

slope = 1

$y = mx + c$

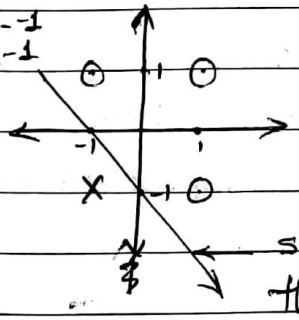
$\therefore y = mx + 1$ $(c = 1)$

$\therefore x - y + 1 = 0$ $(m = 1)$

$\therefore \omega_1 = 1, \omega_2 = -1, b = 1$

(iii)

x	y
-1	0
1	0



← seperating hyperplane passing through (0,-1)

slope = -1

$y = mx + c$

$\therefore y = -x - 1$

$\therefore x + y + 1 = 0$

$\therefore \omega_1 = 1 \quad \omega_2 = 1 \quad b = 1$

Q.2

c X_1 (XOR) X_2

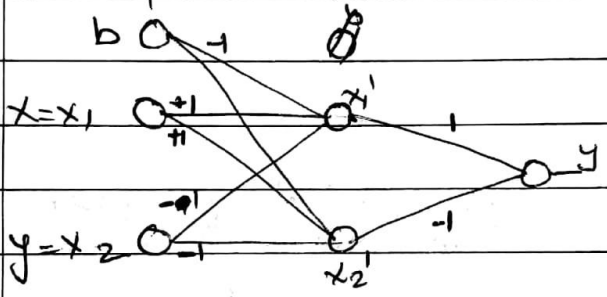
$= (X_1 \text{ AND } (\text{NOT } X_2)) \text{ OR } ((\text{NOT } X_1) \text{ AND } X_2)$

$= x - y - 1 + x - y + 1$

$= x - y$

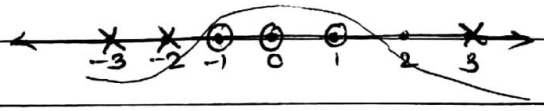
∴ equation of line $\Rightarrow x = y$.

Perceptron Network:



Q.3

a.

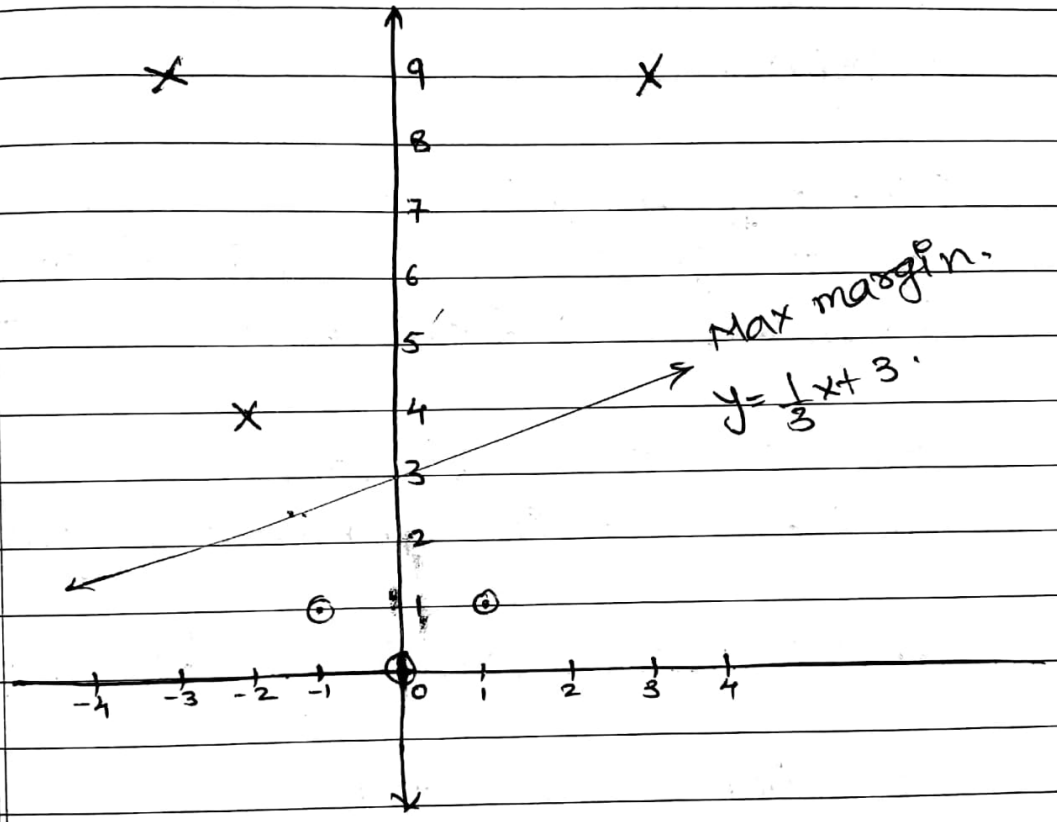


Since features is 1 dimensional scalar & distribution of class variables is as seen above, a linear seperator cannot exist.

Q.3

b. $\phi(X_-) = \{(0,0), (1,1), (-1,1)\}$

$\phi(X_+) = \{(-3,9), (3,9), (-2,4)\}$



Perfect linear seperator exists for $\phi(x)$. as seen above.

Q.3 c. $k(\phi(x_1), \phi(x_2)) = \langle \phi(x_1), \phi(x_2) \rangle$

$= \langle \{x_1, x_1^2\}, \{x_2, x_2^2\} \rangle$

$= x_1 * x_2 + x_1^2 * x_2^2$

Q.3

d. As seen in graph of Q.3b, points (-1, 1) and (-2, 4) are closest to any linear seperator. Therefore they will be used for finding maximum margin seperator.

The max seperator should be a perpendicular bisector to line joining (-1, 1) and (-2, 4) and should be equidistant from the two points.

~~let~~ : Mid point = $(\frac{-3}{2}, \frac{5}{2})$

and slope of line joining two points = -3

∴ slope of perpendicular bisector = $\frac{1}{3} = m'$

∴ equation of seperator:

$$(y - \frac{5}{2}) = m'(x + \frac{3}{2})$$

$$\therefore (y - \frac{5}{2}) = \frac{1}{3}(x + \frac{3}{2})$$

$$\therefore y = \frac{5}{2} + \frac{1}{3}x + \frac{1}{2}$$

$$\therefore y = \frac{1}{3}x + 3.$$

⇒ The value of margin = $\sqrt{(4-1)^2 + (-2+1)^2}$

$= \sqrt{3^2 + (-1)^2}$

$= \sqrt{10}$

Q.4 $n = d = 10^{10}$

a) Perceptron:

Training time: $O(nd^* T)$

since $T = \frac{1}{\gamma^2}$ & $\gamma = \arg\min (w, x_i)$

~~time = O(nd)~~

Test time = $O(d)$

b) Nearest Neighbour

Training Time: $O(1)$ (No preprocessing done)

Testing Time: $O(nd)$

c) Kernel Perceptron with polynomial kernel of order 4.

Training Time: $O(ndT)$

Testing Time: $O(d)$

d) kernel perceptron with gaussian kernel.

Training Time: $O(ndT)$

Testing Time: $O(d)$

e) Linear Support Vector Machine.

Training Time: $O(nd)$

Testing Time: $O(d)$

Decreasing order

(i) Training Time

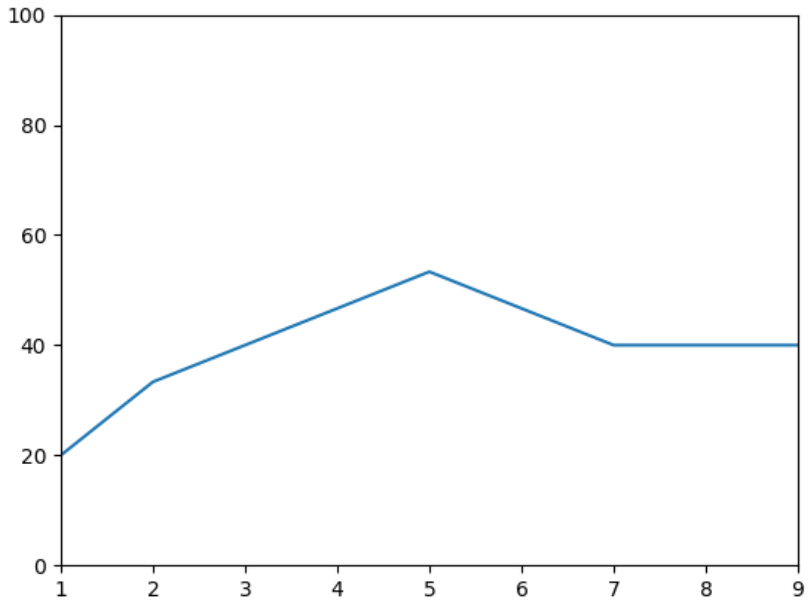
$a = c = d > e > b$

(ii) Testing Time

$b > a = c = d = e$

Q5)
KNN – Without Normalization

```
>>>
 RESTART: C:/Tanmay/Assignments - Github folder/Semester 3/Data Analytics/Assign
ment 2/KNN.py
Error Percentage for k = 1 : 20.0
Error Percentage for k = 2 : 33.3333333333
Error Percentage for k = 3 : 40.0
Error Percentage for k = 4 : 46.6666666667
Error Percentage for k = 5 : 53.3333333333
Error Percentage for k = 6 : 46.6666666667
Error Percentage for k = 7 : 40.0
Error Percentage for k = 8 : 40.0
Error Percentage for k = 9 : 40.0
```



Code:

```
import csv
import numpy as np
import random
from sklearn import neighbors
import sys
import matplotlib.pyplot as plt

#Variable definition
CLASS1 = 59
CLASS2 = 71
CLASS3 = 48
totalrecords = 178
train_rows = (CLASS1-5)+(CLASS2-5)+(CLASS3-5)
train_cols = 13
```

```
global train_data, train_label, test_data, test_label, label, data
global knn, wrong_class, neighbors, error

def main():
   #readcsv data
   data = np.genfromtxt('wine.data.csv', delimiter=',')
   label = data[:, 0]
   data = data[:,1:]

   #train data
   global wrong_class, neighbors, error
   error=[0]*10
   splitdata(data, label)

   for i in range(1,10):
      train_model(i)

      #predict data label
      iter_no = 100
      wrong_class=0
      predict_label(i, iter_no)
      error_prob(i, 15, iter_no)
   plot_error()

def plot_error():
   plt.plot(error[:])
   plt.axis([1, 9, 0, 100])
   plt.show()

def error_prob(k, total_data, iter_no):
   global wrong_class, error
   error[k] = (wrong_class / (total_data*iter_no)) * 100.0
   print "Error Percentage for k =",k,":", error[k]


def predict_label(k, iter_no):
   global test_data, test_label
   global knn, wrong_class

   for i in range(iter_no):
      #result = KNN(k)
      result = knn.predict(test_data)

      for i in range(0, len(result)):
         if(result[i] != test_label[i]):
            wrong_class+=1.0

def train_model(k):
   global train_data, train_label
```

```
    global knn

    #Train Model
    knn = neighbors.KNeighborsClassifier(n_neighbors = k, algorithm='kd_tree')
    knn.fit(train_data, train_label)



def splitdata(data, label):
    global train_data, train_label, test_data, test_label
    train_data = data
    train_label = label

    test_data = np.empty([15, train_cols])
    test_label = np.empty([15, 1])

    no1 = random.sample(range(0, 59-1), 5)
    no2 = random.sample(range(59, 130-1), 5)
    no3 = random.sample(range(130, totalrecords-1), 5)
    no4 = no1 + no2 + no3
    train_data = np.delete(train_data, (no4), axis=0)
    train_label = np.delete(train_label, (no4), axis=0)

    i=0
    for no in no4:
        test_data[i] = data[no]
        test_label[i] = label[no]
        i+=1

main()



def calc_EHdist(train_row, test_row):
    dist=0.0
    for i in range(0, len(train_row)):
        dist += pow((train_row[i]-test_row[i]),2)
    #print dist
    return dist

def kneighbors(min_val, dist, row_index, k):

    largest=0.0
    #print min_val
    index=0
    for i in range(0,k):
        if min_val[i][0] > largest:
            largest = min_val[i][0]
            index = i
    if min_val[index][0] > dist:
```

```
        min_val[index][0] = dist
        min_val[index][1] = row_index
    return min_val

def KNN(k):
    #compute manhattan distance between test data and training data
    global test_data, train_data
    result = []
    #For each test record
    for test_row in test_data:
        #argmin of manhattan distance
        min_val=[]
        for i in range(0,k):
            if(i==0):
                min_val=[[sys.float_info.max,0]]
                continue

            min_val.append([sys.float_info.max, 0])

        row_index=0
        for train_row in train_data:
            dist = calc_EHdist(train_row, test_row)
            min_val = kneighbors(min_val, dist, row_index,k)
            row_index+=1

        #return max label
        a, b, c = 0,0,0

        for minv in min_val:
            label = train_label[minv[1]]
            if label == 1:
                a+=1
            elif label == 2:
                b+=1
            else:
                c+=1
        print a,b,c
        if (a>=b & a>=c):
            result.append(1)
        elif (b>a & b>=c):
            result.append(2)
        else:
            result.append(3)
    return result
```

Q.7)
Time spent = 12 hours

Acknowledgement: Discussed with Nitesh Gupta.