Assignment No:1
Sathiskumar J
20172092

Problem 1:
In normal perceptron we are updating the weights periodically which will change the weight vector which survived for longer time which will result in large misclassification while testing.

In voted perceptron if the weight vector changes periodically then there is no point in storing count of each weight vector used for classification,it can be used if weight vector doesn't change in short span of time.

Cancer Dataset:

```
import csv
import sys
import random
import random
import numpy as np

"""
function to read dataset of cancer
"""

def readDataset(file):
  DataSetList = []
  Y_list = []
  with open(file, 'rb') as f:
    reader = csv.reader(f)
    for row in reader:
      if "?" in row:
        continue
      else:
        y = row[-1]
        row.pop()
        num_row = list(map(float, row))
        num_row.append(1.0)
        if(y=='2'):
          num_row.append(2)
        else:
          num_row.append(4)
```

```
            DataSetList.append(num_row[0:len(num_row)])

    return DataSetList
##############################################################################
##################################################

"""
splitting the dataset
"""
def split_data(dataset, folds):
  splitted_data = list()
  dataset_copy = list(dataset)
  fold_size = int(len(dataset) / folds)

      for i in range(folds):
          fold = list()
          while len(fold) < fold_size:
              index = random.randrange(len(dataset_copy))
              fold.append(dataset_copy.pop(index))

          splitted_data.append(fold)

      return splitted_data
##############################################################################
##################################################
"""
function to calculate weight vector for voted perceptron
"""
def train_system(train_set,epoch):
    DataSetMatrix = np.array(train_set)
    eta = 1
    bias = 0;
    epochs = epoch
    count = 1

    w = np.zeros(len(DataSetMatrix[0][1])-1)
    listofweight=[]
    listofbias=[]
    listofcount=[]
    max=0
```

```
  for epoch in range(epochs):
    for row in DataSetMatrix:
      # to get each fold iterate througing fold
      for i in row:
        x=i[0:-1]
        if i[-1]==2:
          y=1
        else:
          y=-1
        tr=np.dot(y, np.dot(w, x))
        if (tr <= 0):
          listofweight.append(w)
          listofcount.append(count)
          w = np.add(w, np.dot(eta, np.dot(y, x)))
          # w = w + eta * x * y
          # bias = bias + y
          count = 1
          # print(w)
        else:
          count = count + 1

  return listofweight,listofcount

#############################################################################
#################################################

"""""""

function to get predicted value for give testset

"""""


def predict_value(test_set,weight,count):
  predicted =[]
  # print("length",len(test_set))
  for row in test_set:
    i=0
    t2=0
    t1=0
```

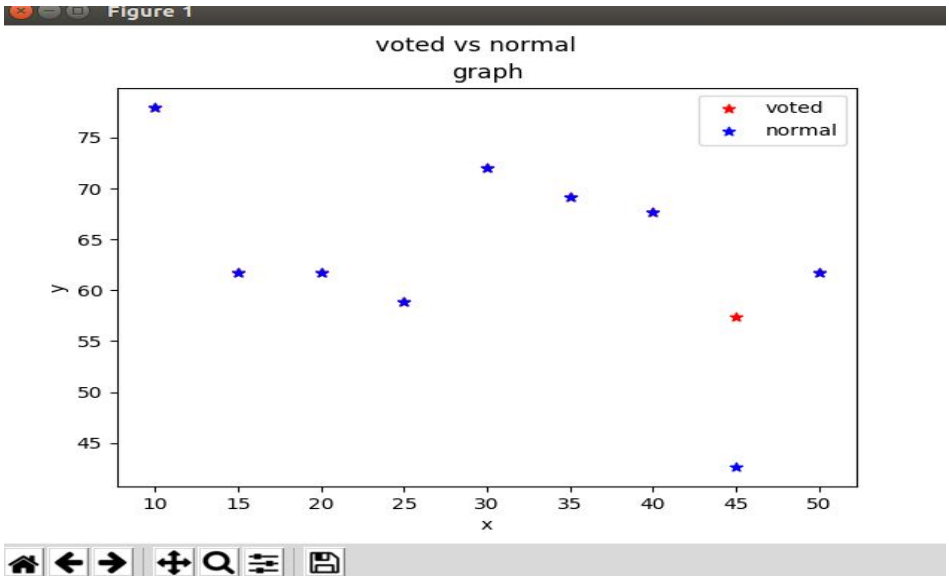
```
        for lw in weight:
            t1 = np.dot(row,lw)
            if(t1>=0):
                t3=1
            else:
                t3=-1
            t2+=(count[i]*t3)
            # print(count[i])
            i=i+1
        if(t2>=0):
            predicted.append(1)
        else:
            predicted.append(-1)
    return predicted

##########################################################################################
#################################################

"""""""

function to find accuracy

"""""


def check_correctnes(original,predicted):
    correct = 0
    for i in range(len(original)):
        if original[i] == predicted[i]:
            correct += 1
    return correct / float(len(original)) * 100.0

##########################################################################################
#################################################




def train_system_normal(train_set,epoch):
    DataSetMatrix = np.array(train_set)
```

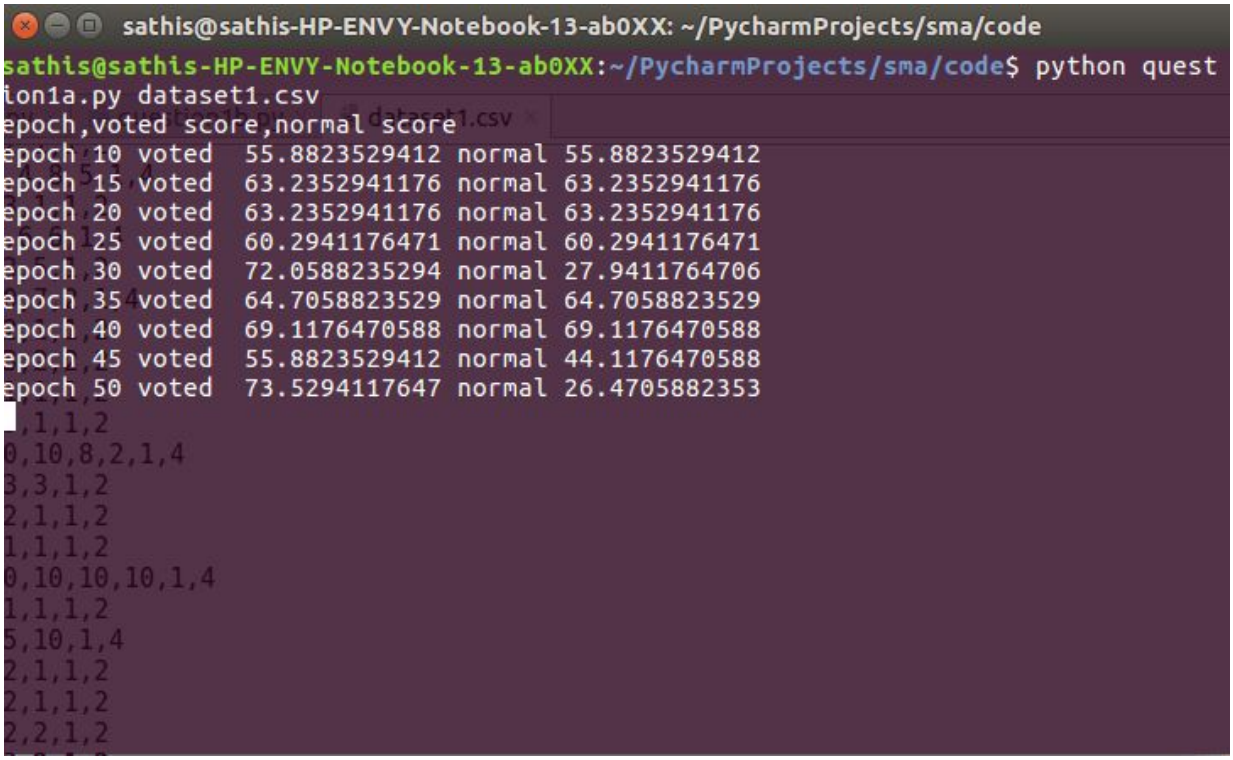
```
    eta = 1
    bias = 0;
    epochs = epoch
    count = 1
    w = np.zeros(len(DataSetMatrix[0][1]) - 1)
    for epoch in range(epochs):
        for row in DataSetMatrix:
            # to get each fold iterate througing fold
            for i in row:
                # print(i[0:-1])
                # print(i[-1])
                x = i[0:-1]
                if i[-1] == 2:
                    y = 1
                else:
                    y = -1

                tr = np.dot(y, np.dot(w, x))
                if (tr <= 0):
                    w = np.add(w, np.dot(eta, np.dot(y, x)))

        res=w
    return res

def predict_system_noraml(test_set,weight):
    predicted = []
    # print("length",len(test_set))
    for row in test_set:
        t1 = np.dot(row, weight)
        if (t1 >= 0):
            predicted.append(1)
        else:
            predicted.append(-1)

    return predicted

if __name__=="__main__":


    DataSet=readDataset(sys.argv[1])
```

```
list_of_epochs = [10,15,20,25,30,35,40,45,50]
# list_of_epochs = [10]

voted_list =[]
normal_list =[]
print("epoch,voted score,normal score")

for epoch in list_of_epochs:

    splitted_data=split_data(DataSet,10)

    test_set=splitted_data[0]

    splitted_data.remove(splitted_data[0])

    train_set = splitted_data

    weight,count = train_system(train_set,epoch)

    original =[]
    mtest =[];
    for row in test_set:
        # print(row)
        mtest.append(row[0:-1])
        # print(mtest)
        if(row[-1]==2):
            original.append(1)
        else:
            original.append(-1)

    predicted = predict_value(mtest, weight,count)

    score = check_correctnes(original,predicted)

    voted_score =score
##############################################################################
###############################################
    weight= train_system_normal(train_set, epoch)
```

```
    # print("weight")
    predicted=predict_system_noraml(mtest,weight)

    score = check_correctnes(original, predicted)
    normal_score=score
    # print "normal perceptron"
    print "epoch",epoch,"voted ",voted_score,"normal",normal_score
    voted_list.append(voted_score)
    normal_list.append(normal_score)


import matplotlib.pyplot as plt


plt.scatter(list_of_epochs, voted_list, label="voted", color="red", marker="*")
plt.scatter(list_of_epochs, normal_list, label="normal", color="blue", marker="*")

plt.suptitle('voted vs normal', fontsize=12)
plt.xlabel("x")
plt.ylabel("y")
plt.title("graph")
plt.legend()
plt.show()
```

Output:





Ionosphere Dataset:

```
import csv
import random
import random
import numpy as np

"""
function to read dataset of ionosphere
"""

def readDataset(file):

  DataSetList = []
  Y_list = []
  with open(file, 'rb') as f:
     reader = csv.reader(f)
     for row in reader:
        if "?" in row:
           continue
```

```
            else:
                y = row[-1]
                row.pop()
                num_row = list(map(float, row))
                num_row.append(1.0)
                if(y=='g'):
                    num_row.append(2)
                else:
                    num_row.append(4)
                DataSetList.append(num_row[0:len(num_row)])

    return DataSetList
#################################################################################
################################################

"""
splitting the dataset
"""
def split_data(dataset, folds):
  splitted_data = list()
  dataset_copy = list(dataset)
  fold_size = int(len(dataset) / folds)
      for i in range(folds):
          fold = list()
          while len(fold) < fold_size:
              index = random.randrange(len(dataset_copy))
              fold.append(dataset_copy.pop(index))
          splitted_data.append(fold)
      return splitted_data
#################################################################################
################################################
"""
function to calculate weight vector for voted perceptron
"""
def train_system(train_set,epoch):
    DataSetMatrix = np.array(train_set)

    eta = 1
    bias = 0;
    epochs = epoch
```

```
    count = 1

    w = np.zeros(len(DataSetMatrix[0][1])-1)
    listofweight=[]
    listofbias=[]
    listofcount=[]
    max=0
    for epoch in range(epochs):
        for row in DataSetMatrix:
            # to get each fold iterate througing fold
            for i in row:
                x=i[0:-1]
                if i[-1]==2:
                    y=1
                else:
                    y=-1

                 tr=np.dot(y, np.dot(w, x))
                if (tr <= 0):
                    listofweight.append(w)
                    # listofbias.append(bias)
                    listofcount.append(count)
                    w = np.add(w, np.dot(eta, np.dot(y, x)))
                    # w = w + eta * x * y
                    # bias = bias + y
                    count = 1
                    # print(w)
                else:
                    count = count + 1

    return listofweight,listofcount

##########################################################################################
#################################################

"""""""
function to get predicted value for give testset
"""""
```

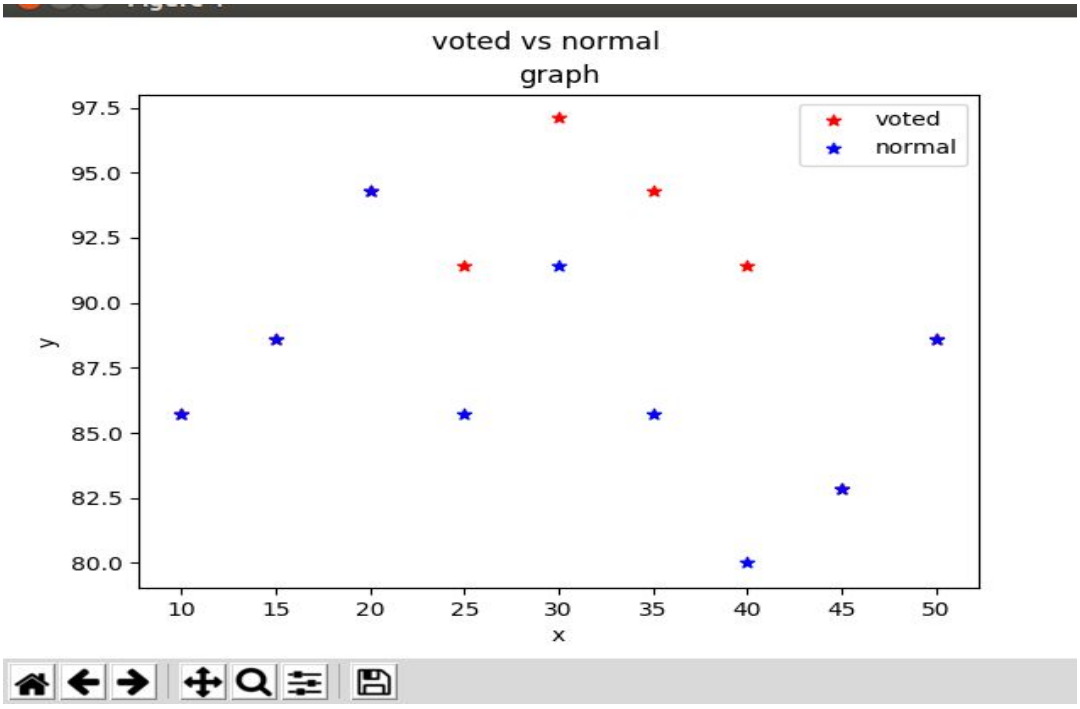
```
def predict_value(test_set,weight,count):

  predicted =[]
  # print("length",len(test_set))
  for row in test_set:
    i=0
    t2=0
    t1=0
    for lw in weight:
      t1 = np.dot(row,lw)
      if(t1>=0):
        t3=1
      else:
        t3=-1
      t2+=(count[i]*t3)
      # print(count[i])
      i=i+1
    if(t2>=0):
      predicted.append(1)
    else:
      predicted.append(-1)
  return predicted

##########################################################################################
#################################################

"""""""
function to find accuracy
"""""


def check_correctnes(original,predicted):
  correct = 0
  for i in range(len(original)):
    if original[i] == predicted[i]:
      correct += 1
  return correct / float(len(original)) * 100.0

##########################################################################################
#################################################
```

```
def train_system_normal(train_set,epoch):
  DataSetMatrix = np.array(train_set)

  eta = 1
  bias = 0;
  epochs = epoch
  count = 1
  w = np.zeros(len(DataSetMatrix[0][1]) - 1)



  for epoch in range(epochs):
    for row in DataSetMatrix:
      # to get each fold iterate througing fold
      for i in row:
        # print(i[0:-1])
        # print(i[-1])
        x = i[0:-1]
        if i[-1] == 2:
          y = 1
        else:
          y = -1
        tr = np.dot(y, np.dot(w, x))

        if (tr <= 0):
          w = np.add(w, np.dot(eta, np.dot(y, x)))
    res=w
  return res



def predict_system_noraml(test_set,weight):
  predicted = []
  # print("length",len(test_set))
  for row in test_set:
    t1 = np.dot(row, weight)
    if (t1 >= 0):
      predicted.append(1)
```

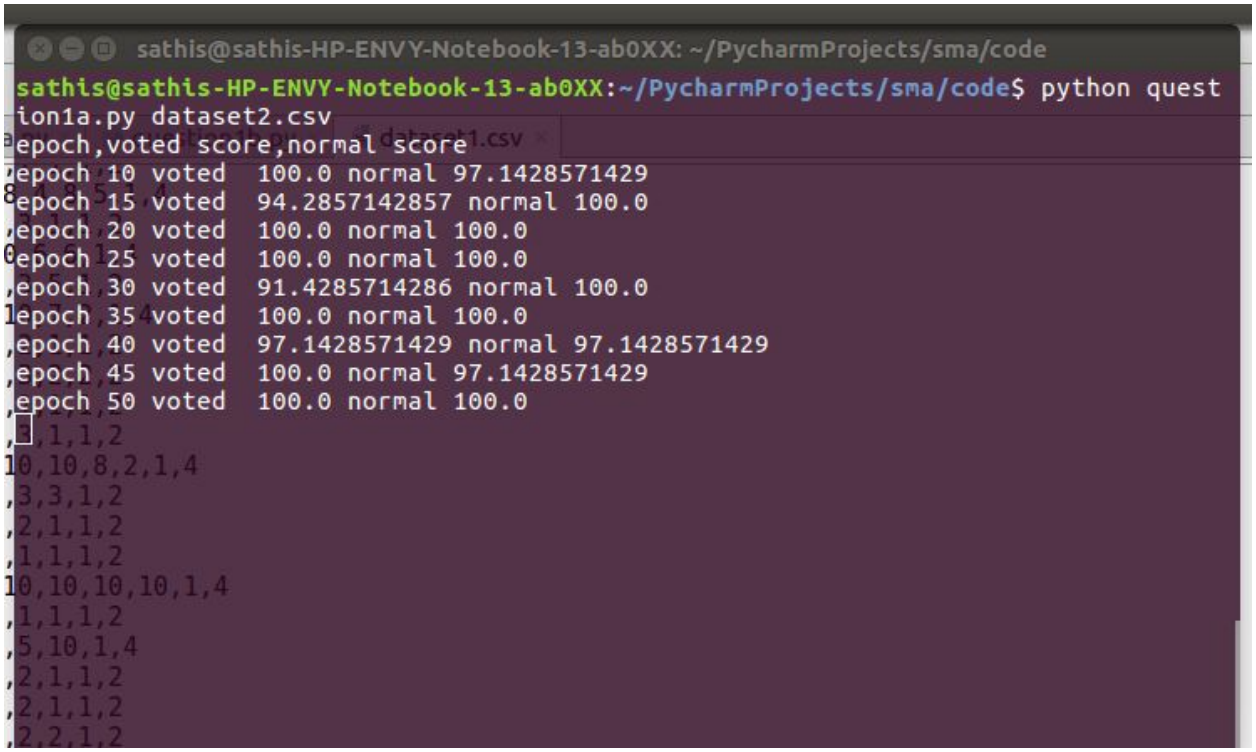
```
        else:
            predicted.append(-1)

    return predicted

if __name__=="__main__":


    DataSet=readDataset(sys.argv[1])
    list_of_epochs = [10,15,20,25,30,35,40,45,50]
    # list_of_epochs = [10]

    voted_list =[]
    normal_list =[]
    print("epoch,voted score,normal score")

    for epoch in list_of_epochs:

        splitted_data=split_data(DataSet,10)
        test_set=splitted_data[0]
        splitted_data.remove(splitted_data[0])
        train_set = splitted_data
        weight,count = train_system(train_set,epoch)

        original =[]
        mtest =[];
        for row in test_set:
            # print(row)
            mtest.append(row[0:-1])
            # print(mtest)
            if(row[-1]==2):
                original.append(1)
            else:
                original.append(-1)

        predicted = predict_value(mtest, weight,count)
        score = check_correctnes(original,predicted)
        voted_score =score
##############################################################################
##################################################
```

```
    weight= train_system_normal(train_set, epoch)
    # print("weight")
    predicted=predict_system_noraml(mtest,weight)
    score = check_correctnes(original, predicted)
    normal_score=score
    # print "normal perceptron"
    print "epoch",epoch,"voted ",voted_score,"normal",normal_score
    voted_list.append(voted_score)
    normal_list.append(normal_score)

import matplotlib.pyplot as plt
plt.scatter(list_of_epochs, voted_list, label="voted", color="red", marker="*")
plt.scatter(list_of_epochs, normal_list, label="normal", color="blue", marker="*")
plt.suptitle('voted vs normal', fontsize=12)
plt.xlabel("x")
plt.ylabel("y")
plt.title("graph")
plt.legend()
plt.show()
```

Output:



Output:



Problem:2

In least square approach we will calculate minimum distance of points(data) from the line(classifier),which will not classify the data points correctly sometimes.

In LDA we try to find classifier by using two property.
1.maximum distance between clusters centroid.
2.minimum distance between data points within the cluster

For DataSet1:

```
import numpy as np
import sys

def leastSquareApproach(inputdata):
  size = len(inputData)
```

```
b = [1 for x in range(size)]

x1 = []
y1 = []
x2 = []
y2 = []

for row in inputData:
    if (row[0] == 1):
        x1.append(row[1])
        y1.append(row[2])
    else:
        x2.append(row[1])
        y2.append(row[2])

b_transpose = np.matrix(b).getT()

matrix_a = np.matrix(inputData)
matrix_a_transpose = matrix_a.getT()

a_transpose_a = matrix_a_transpose * matrix_a
a_transpose_a_inverse = a_transpose_a.getI()
matrix_x = a_transpose_a_inverse * matrix_a_transpose
result = matrix_x * b_transpose

w0 = result.flat[0]
w1 = result.flat[1]
w2 = result.flat[2]

lx=x1+x2
# print(lx)
ly=[]

for i in lx:
    t = -(w0 + w1 * i) / w2
    ly.append(t)

import matplotlib.pyplot as plt
plt.plot(lx, ly)
```

```
    plt.scatter(x1, y1, label="c1", color="red", marker="*")
    plt.scatter(x2, y2, label="c2", color="blue", marker="*")
    plt.suptitle('least square approach', fontsize=12)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("graph")
    plt.legend()
    plt.show()


##########################################################################
##############################################

def lda(inputdata):
    class1 = []
    class2 = []
    x1 = []
    y1 = []
    x2 = []
    y2 = []

    for row in inputData:
        if (row[0] == 1):
            class1.append(row)
            x1.append(row[1])
            y1.append(row[2])
        else:
            class2.append(row)
            x2.append(row[1])
            y2.append(row[2])

    class1_matrix = np.matrix(class1)
    class2_matrix = np.matrix(class2)

    mclass1_matrix = class1_matrix[:, 1:3]
    mclass2_matrix = class2_matrix[:, 1:3]

    tmc1 = mclass1_matrix - mclass1_matrix.mean(axis=0)
    tmc2 = mclass2_matrix - mclass2_matrix.mean(axis=0)
```

```
    cov_c1 = tmc1.getT() * tmc1
    cov_c2 = tmc2.getT() * tmc2
    Sw = cov_c1 + cov_c2
    inverse_of_Sw = Sw.getI()
    mean_of_matrix = mclass1_matrix.mean(axis=0) - mclass2_matrix.mean(axis=0)
    result = inverse_of_Sw * mean_of_matrix.getT()
    w0 = result.flat[0]
    w1 = result.flat[1]
    lx = x1 + x2
    ly = []

    for i in lx:
        t = (w0 + w1 * i)
        ly.append(t)

    import matplotlib.pyplot as plt

    plt.plot(lx, ly)
    plt.scatter(x1, y1, label="c1", color="red", marker="*")
    plt.scatter(x2, y2, label="c2", color="blue", marker="*")
    plt.suptitle('linear discriminant analysis', fontsize=12)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("graph")
    plt.legend()
    plt.show()




if __name__=='__main__':

    inputData = [[1, 3, 3], [1, 3, 0], [1, 2, 1], [1, 0, 2], [-1, -1, 1], [-1, 0, 0], [-1, -1, -1], [-1, 1, 0]]

    if(sys.argv[1]=='1'):
        leastSquareApproach(inputData)
    elif(sys.argv[1]=='2'):
        lda(inputData)
```

```
    else:
        print("1.leastSquareApproach ")
        print("2.linear discriminant analysis")
```

Datasetset2:

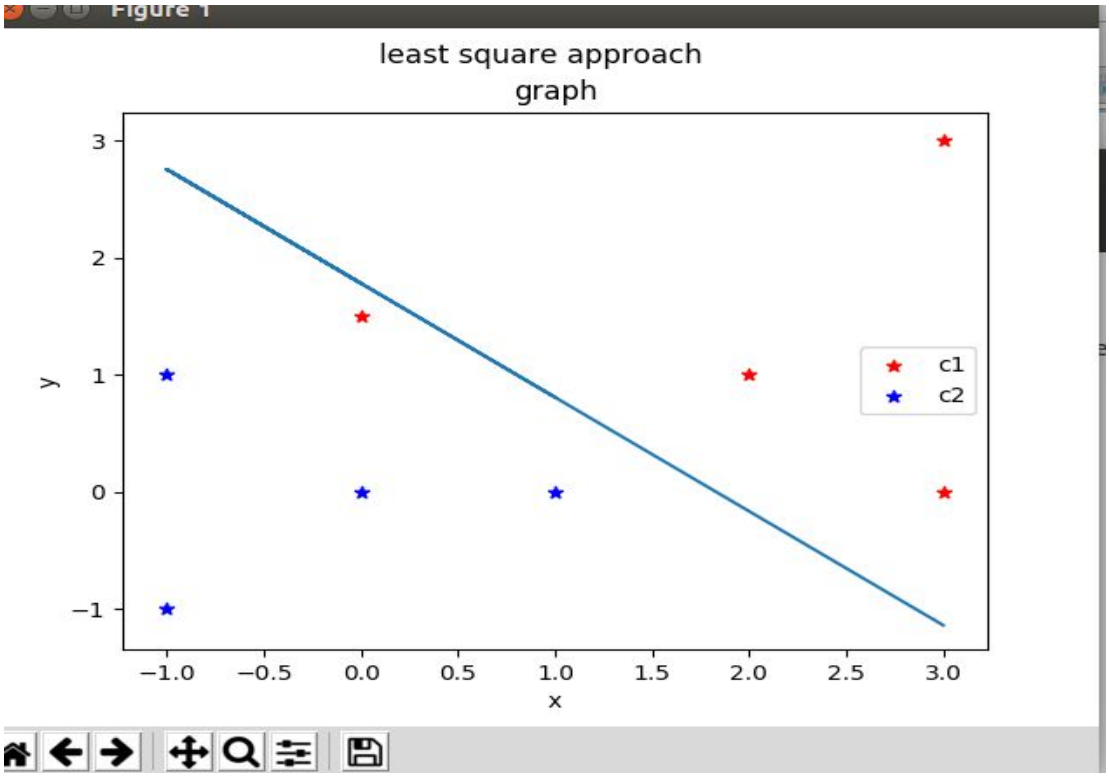
```
import numpy as np
import sys

def leastSquareApproach(inputdata):
    size = len(inputData)
    b = [1 for x in range(size)]
    x1 = []
    y1 = []
    x2 = []
    y2 = []
    for row in inputData:
        if (row[0] == 1):
            x1.append(row[1])
            y1.append(row[2])
        else:
            x2.append(row[1])
            y2.append(row[2])

    b_transpose = np.matrix(b).getT()
    matrix_a = np.matrix(inputData)
    matrix_a_transpose = matrix_a.getT()
    a_transpose_a = matrix_a_transpose * matrix_a
    a_transpose_a_inverse = a_transpose_a.getI()
    matrix_x = a_transpose_a_inverse * matrix_a_transpose
    result = matrix_x * b_transpose
    w0 = result.flat[0]
    w1 = result.flat[1]
    w2 = result.flat[2]

    lx=x1+x2
    # print(lx)
    ly=[]
```

```
    for i in lx:
        t = -(w0 + w1 * i) / w2
        ly.append(t)

    import matplotlib.pyplot as plt

    plt.plot(lx, ly)
    plt.scatter(x1, y1, label="c1", color="red", marker="*")
    plt.scatter(x2, y2, label="c2", color="blue", marker="*")
    plt.suptitle('least square approach', fontsize=12)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("graph")
    plt.legend()
    plt.show()


################################################################################
##############################################

def lda(inputdata):
    class1 = []
    class2 = []
    x1 = []
    y1 = []
    x2 = []
    y2 = []

    for row in inputData:
        if (row[0] == 1):
            class1.append(row)
            x1.append(row[1])
            y1.append(row[2])
        else:
            class2.append(row)
            x2.append(row[1])
            y2.append(row[2])

    class1_matrix = np.matrix(class1)
    class2_matrix = np.matrix(class2)
```

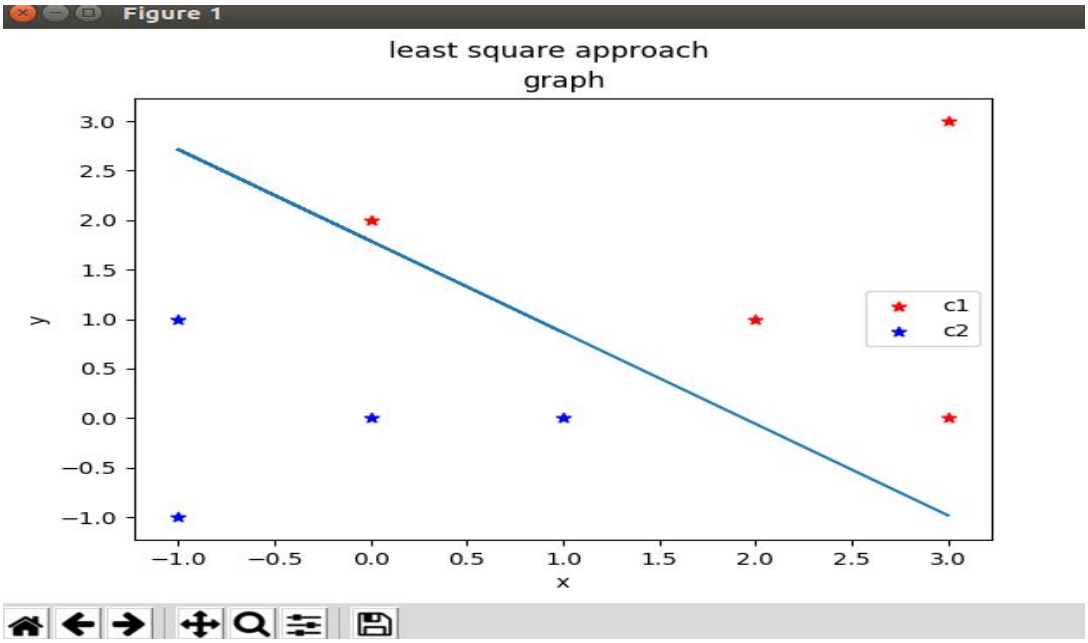
```
    mclass1_matrix = class1_matrix[:, 1:3]
    mclass2_matrix = class2_matrix[:, 1:3]
    tmc1 = mclass1_matrix - mclass1_matrix.mean(axis=0)
    tmc2 = mclass2_matrix - mclass2_matrix.mean(axis=0)

    cov_c1 = tmc1.getT() * tmc1
    cov_c2 = tmc2.getT() * tmc2
    Sw = cov_c1 + cov_c2

    inverse_of_Sw = Sw.getI()

    mean_of_matrix = mclass1_matrix.mean(axis=0) - mclass2_matrix.mean(axis=0)
    result = inverse_of_Sw * mean_of_matrix.getT()

    w0 = result.flat[0]
    w1 = result.flat[1]
    lx = x1 + x2
    ly = []

    for i in lx:
        t = (w0 + w1 * i)
        ly.append(t)
    import matplotlib.pyplot as plt
    plt.plot(lx, ly)
    plt.scatter(x1, y1, label="c1", color="red", marker="*")
    plt.scatter(x2, y2, label="c2", color="blue", marker="*")
    plt.suptitle('linear discriminant analysis', fontsize=12)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("graph")
    plt.legend()
    plt.show()

if __name__=='__main__':

    inputData = [[1, 3, 3], [1, 3, 0], [1, 2, 1], [1, 0, 1.5], [-1, -1, 1], [-1, 0, 0], [-1, -1, -1], [-1, 1, 0]]

    if(sys.argv[1]=='1'):
        leastSquareApproach(inputData)
```

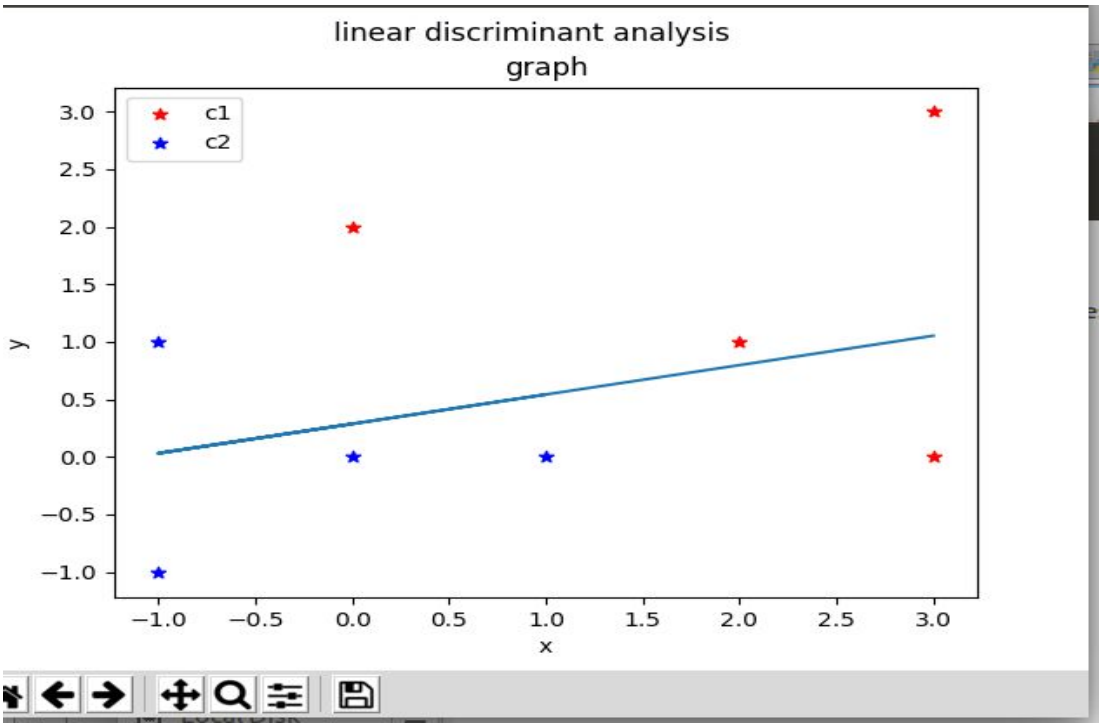
```
elif(sys.argv[1]=='2'):
    lda(inputData)
else:
    print("1.leastSquareApproach ")
    print("2.linear discriminant analysis")
```

Output
For Dataset2:



Output for Dataset1:





For Dataset2:LDA