Bios/CS 534 Project 1

Chenxi Cai

February 21, 2018

1 Problem 1

Class Y = 0:
mean vector:

$$\begin{bmatrix} 1.06242164 & 1.61910524 \end{bmatrix} \tag{1}$$

covariance matrix:

$$\begin{bmatrix} 4.79170095 & 0.90180838 \\ 0.90180838 & 1.2945715 \end{bmatrix} \tag{2}$$

Class Y = 1:
mean vector:

$$\begin{bmatrix} 1.13915258 & -1.18380439 \end{bmatrix} \tag{3}$$

covariance matrix:

$$\begin{bmatrix} 0.7560476 & -0.5093068 \\ -0.5093068 & 3.19387164 \end{bmatrix} \tag{4}$$

1.1 Equal prior case

Classification error rate on the testing data is

$$0.057692 \tag{5}$$

Scatter plots with boundary of the training data is shown as Figure 1.
The s in Class y = 0 and y = 1 is red "o" and blue "x", separately.

1.2 Prior calculated from the data case

Classification error rate on the testing data is

$$0.057692 \tag{6}$$

Scatter plots with boundary of the training data is shown as Figure 2

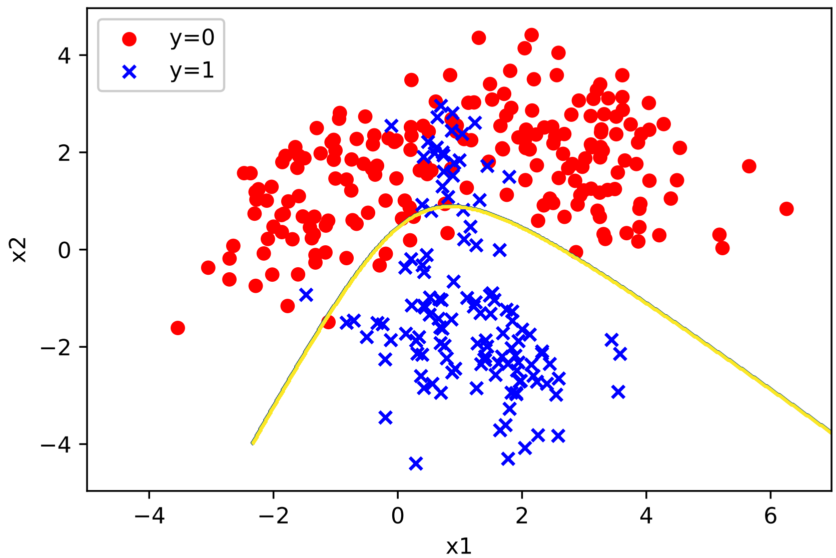

Figure 1: Scatter plots with boundary of equal prior

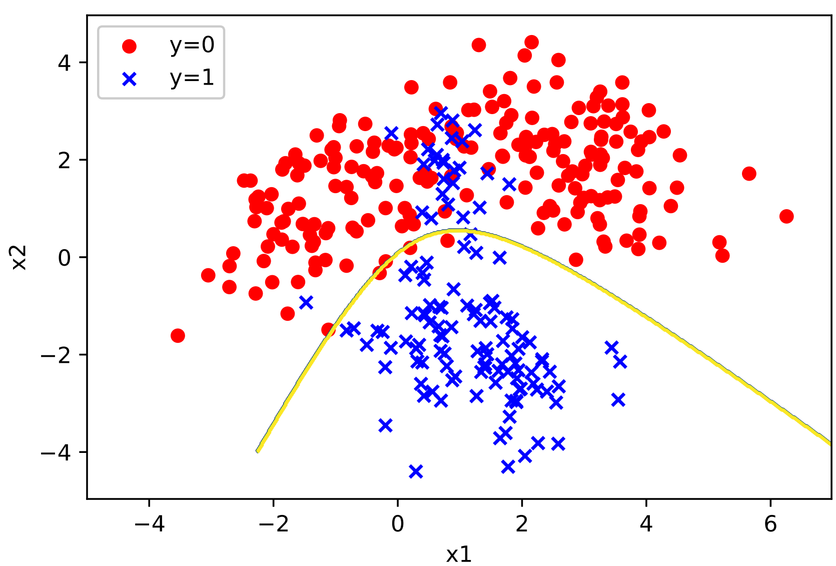

Figure 2: Scatter plots with boundary of prior calculated from the data

2 Problem 2

2.1 Bandwidth=10

Classification error rate on the testing data is

$$0.184615 \tag{7}$$

Scatter plots of the testing data is shown as Figure 3.
Right points are black and misclassified points are red. The shape of points in Class y = 0 and y = 1 is "o" and "x", separately.

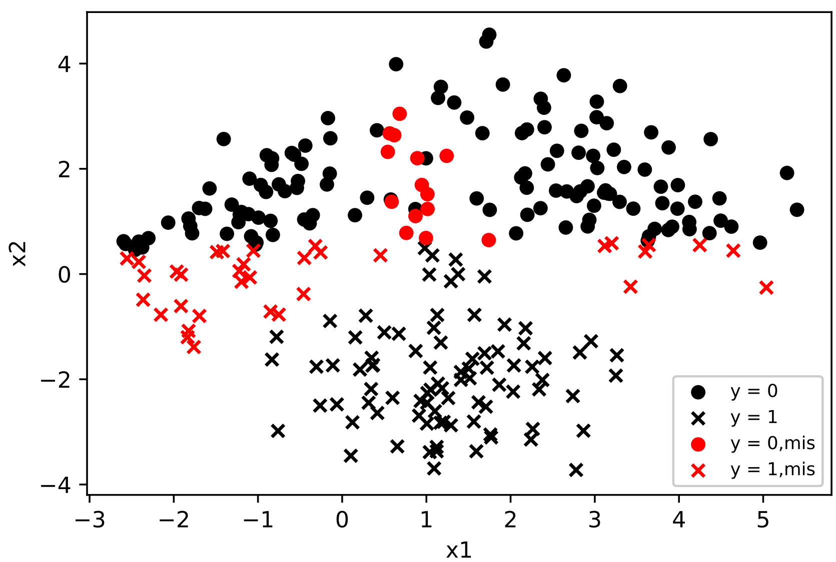

Figure 3: Scatter plots of bandwidth=10

2.2 Bandwidth=1

Classification error rate on the testing data is

$$0.053846 \tag{8}$$

Scatter plots of the testing data is shown as Figure 4.

2.3 Bandwidth=0.1

Classification error rate on the testing data is

$$0.038462 \tag{9}$$

Scatter plots of the training data is shown as Figure 5.

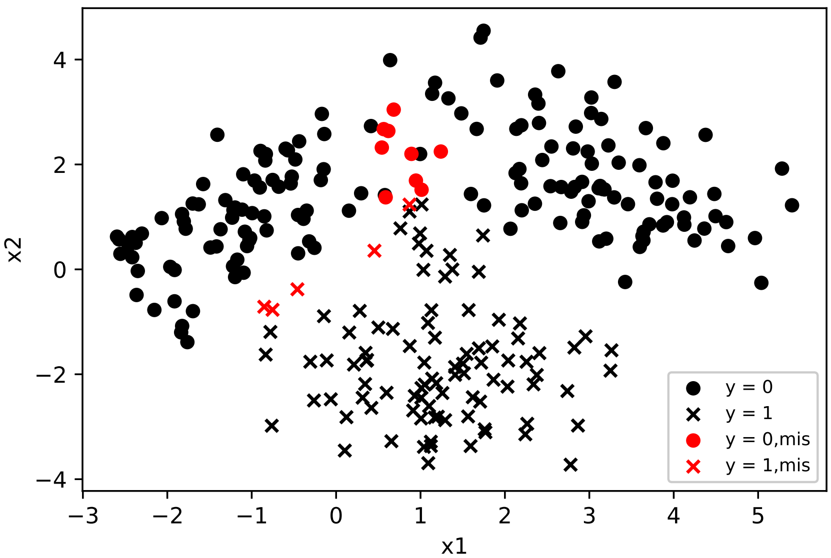

Figure 4: Scatter plots of bandwidth=1

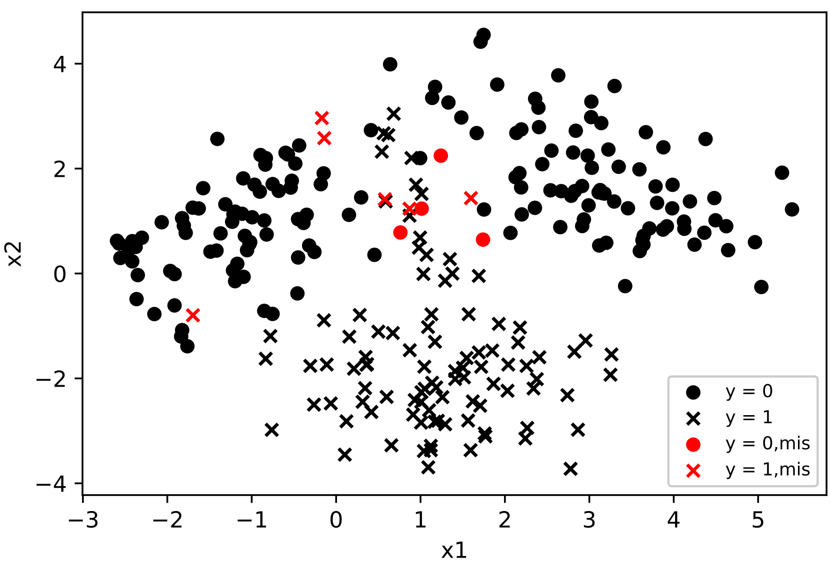

Figure 5: Scatter plots of bandwidth=0.1

3 Problem 3

3.1 K=1

sensitivity = 0.920000, specificity=0.975000, false discovery rate=0.041667
Scatter plots of the testing data is shown as Figure 6.Right points are black and misclassified points are red. The shape of points in Class y = 0 and y = 1 is "o" and "x", separately.

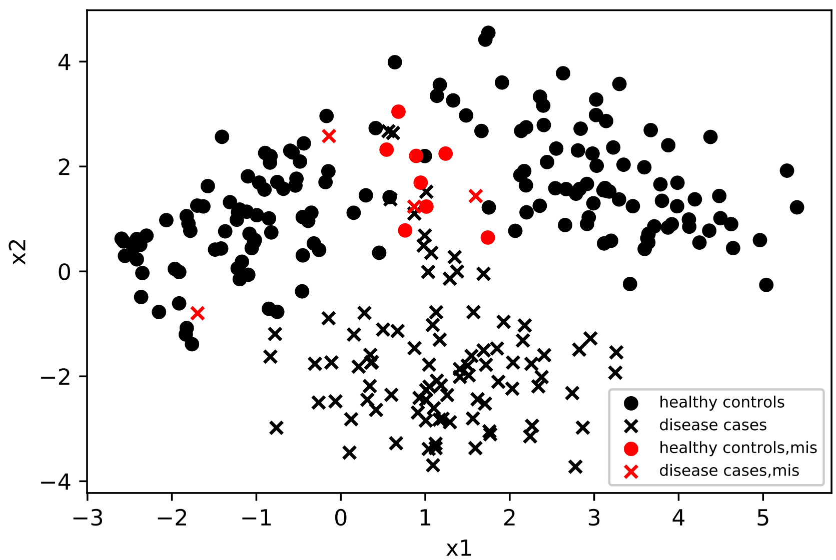

Figure 6: Scatter plots of K=1

3.2 K=5

sensitivity = 0.960000, specificity=0.981250, false discovery rate=0.030303
Scatter plots of the testing data is shown as Figure 7.

3.3 K=10

sensitivity = 0.950000, specificity=0.975000, false discovery rate=0.040404
Scatter plots of the testing data is shown as Figure 8.

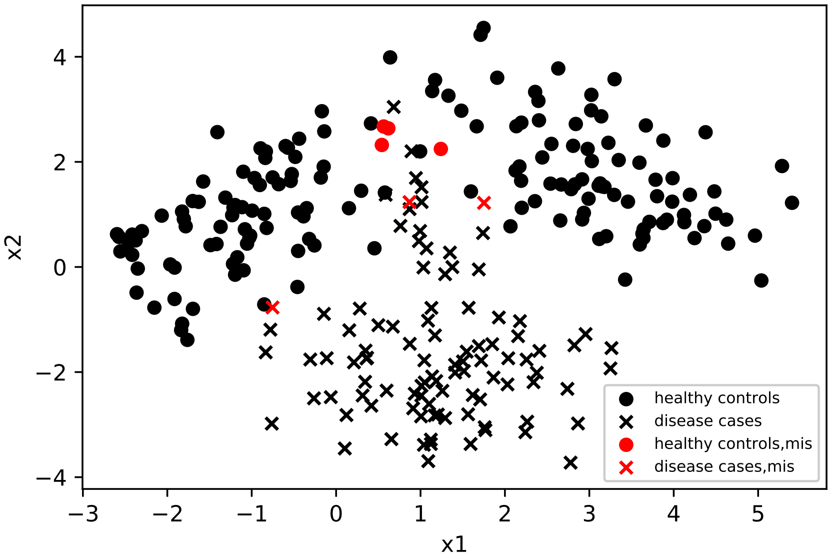

Figure 7: Scatter plots of K=5

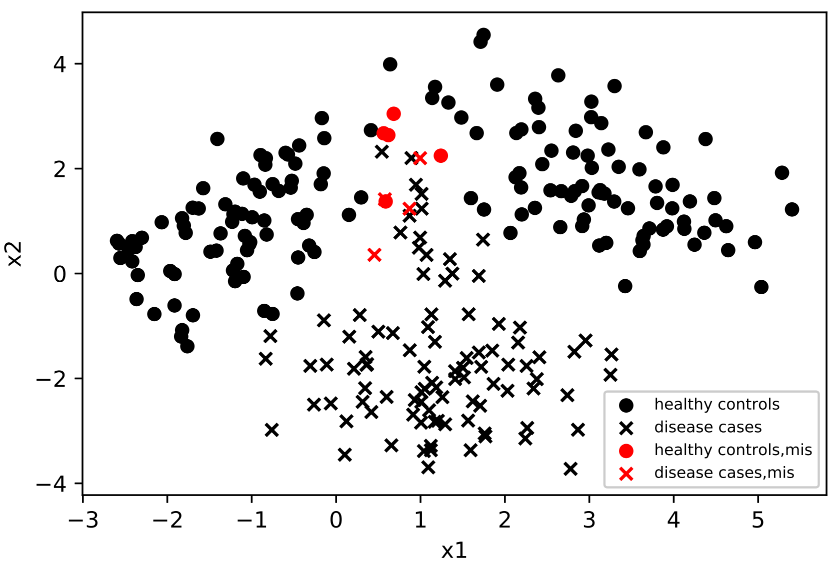

Figure 8: Scatter plots of K=10

Python codes of problem1:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


#define functions of parameters and decision
def discrtminant_func(f,y):
```

```
    mean_vectors = np.zeros((1,2))      #mean vectro and covariance matrix
    covariance = np.zeros((2,2))
    mean_vectors = np.mean(f.loc[lambda f : f[2] == y,[0,1]].values, axis=0)
    covariance = np.cov(f.loc[lambda f : f[2] == y,[0,1]].values.T)
    return mean_vectors, covariance

def decision_function(x,x_m,x_c,prior_probability):
    sig_inv = np.linalg.inv(x_c)          #decision function
    return -.5*np.log( np.linalg.det(x_c))-.5*np.dot(np.dot((x-x_m),sig_inv),(x-x_m).T) + np.log(prior_probability)

def predict(X,m1,c1,m2,c2,frac1,frac2):
    s = X.shape[0]
    new_Y = np.zeros((s,1))
    X_new = np.zeros((s,2))
    j = 0
    l = s-1
    for i in range(s):
        k = []
        k.append(decision_function(X[i,:],m1,c1,frac1))    #result of decision function of class 1
        k.append(decision_function(X[i,:],m2,c2,frac2))    #result of decision function of class 1
        #classify
        new_Y[i] = np.argmax(k)      #find the index that has bigger results(results in which class is the biggest)
        if np.argmax(k) == 0:
            X_new[j,:] = X[i,:]      #rearrange testing points accroding to predict class
            j += 1
        elif new_Y[i] == 1:         #the predict class of points in testing file
            X_new[l,:] = X[i,:]
            l -= 1

    return X_new,new_Y,j,l

def Error_rate(Y,new_Y):
    s = Y.shape[0]
    n = 0
    for i in range(s):
        if (Y[i] != new_Y[i]):      #error_rate = #points whose predict class is different form original one/#total points
            n += 1
    return n/float(s)

def plot(x,m1,c1,m2,c2,frac1,frac2):
    #build grid of xx1,xx2
    xx1,xx2 = np.meshgrid(np.arange(-5, 7,0.02 ),
                          np.arange(-4, 4,0.02))
    #z is the new_Y of xx1 and xx2 field
    Z = predict(np.array([xx1.ravel(), xx2.ravel()]).T,m1,c1,m2,c2,frac1,frac2)[1]

    #change Z to a matrix with the same shape of xx1
    Z = Z.reshape(xx1.shape)
    #draw the boundary line
    plt.contour(xx1, xx2, Z, alpha=0.3)

    #plot points
    plt.scatter(x[:200,0], x[:200,1],30,
            color='red', marker='o', label='y=0')
    plt.scatter(x[200:,0], x[200:,1],30,
    color='blue', marker='x', label='y=1')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend(loc='upper left')
    return 0
```

```python


#read data
f1 = pd.read_csv('/Users/ccai28/Desktop/HW_1_training.txt',sep='\t', header = None, skiprows = 1)
f2 = pd.read_csv('/Users/ccai28/Desktop/HW_1_testing.txt',sep='\t', header = None, skiprows = 1)

#mean vector and covariance matrix
x = f1.loc[:,[0,1]].values
m1,c1 = discrtminant_func(f1,0)
m2,c2 = discrtminant_func(f1,1)
print m1,m2,c1,c2
#read testing data
X = f2.loc[:,[0,1]].values
Y = f2.loc[:,[2]].values

#equal prior
X_new_e,new_Y_e,j_e,l_e = predict(X,m1,c1,m2,c2,0.5,0.5)
error_rate_e = Error_rate(Y,new_Y_e)
print('Equal prior: \nerror rate = {:f} \n'.format(error_rate_e))
plot(x,m1,c1,m2,c2,0.5,0.5)
plt.savefig('Equal prior .eps',format='eps')
plt.show()

#from data
X_new_d,new_Y_d,j_d,l_d = predict(X,m1,c1,m2,c2,200/float(325),125/float(325))
error_rate_d = Error_rate(Y,new_Y_d)
print('Prior calculated from the data: \nerror rate = {:f} \n'.format(error_rate_e))
plot(x,m1,c1,m2,c2,200/float(325),125/float(325))
plt.savefig('Prior calculated from the data.eps',format='eps')
plt.show()
```

Python codes of problem2:

```python
from sklearn.neighbors.kde import KernelDensity
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def kernel(X,x,bandwidth):
    kde_1 = KernelDensity(kernel='gaussian', bandwidth=bandwidth).fit(x)
    d = 10**(kde_1.score_samples(X)) #using scleaner to calculate density p(x)
    return d

def predict(d1,d2,frac):
    sum = d1*frac+d2*frac
    k = []
    k.append(d1*frac/float(sum))   #calculate posterior according to (density * prior)/evidence
    k.append(d2*frac/float(sum))
    #classify
    new_y = np.argmax(k)           #find the index with 1 -> predict value
    return new_y

def Error_rate(Y,new_Y):
    s = Y.shape[0]
    n = 0
    for i in range(s):
        if (Y[i] != new_Y[i]):         #error_rate = #points of testing predict to another class/#totla points
            n += 1
    return n/float(s)

def misclassified_point(X,Y,new_Y):
    mis_0 = []
```

```python
    mis_1 = []
    right_0 = []
    right_1 = []
    s = Y.shape[0]
    for i in range(s):
        if (Y[i] != new_Y[i] and new_Y[i] == 0):        #find original class 1 training point -> predict class 0
            mis_0.append(X[i,:])
        elif (Y[i] != new_Y[i] and new_Y[i] == 1):      #find original class 0 training point -> predict class 1
            mis_1.append(X[i,:])
        elif (Y[i] == new_Y[i] and new_Y[i] == 0):      #find original class 0 training point -> predict class 0
            right_0.append(X[i,:])                      #find original class 1 training point -> predict class 1
        else:
            right_1.append(X[i,:])
    mis_0 = np.matrix(mis_0)
    right_0 = np.matrix(right_0)
    mis_1 = np.matrix(mis_1)
    right_1 = np.matrix(right_1)
    return mis_0,mis_1,right_0,right_1

def plot(X,x1,x2,bandwidth,Y,frac):

    d1 = kernel(X,x1,bandwidth)
    d2 = kernel(X,x2,bandwidth)

    #predict
    s = X.shape[0]
    new_Y = np.zeros((s,1))
    for i in range(s):
        new_Y[i] = predict(d1[i],d2[i],frac)      #calculate predict value for every point in testing file

    #error_rate
    print('When bandwidth={:f}, \nerror rate = {:f} \n'.format(bandwidth, Error_rate(Y,new_Y)))

    #mis points
    mis_0,mis_1,right_0,right_1 = misclassified_point(X,Y,new_Y)

    plt.scatter(right_0[:,0],right_0[:,1],30,
            color='black', marker='o', label='y = 0')
    plt.scatter(right_1[:,0],right_1[:,1],30,
            color='black', marker='x', label='y = 1')
    plt.scatter(mis_0[:,0],mis_0[:,1],30,
            color='red', marker='o', label='y = 0,mis')
    plt.scatter(mis_1[:,0],mis_1[:,1],30,
            color='red', marker='x', label='y = 1,mis')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend(loc='lower right',
         fontsize = 7)
    plt.savefig('bandwidth={:f}.eps'.format(bandwidth),format='eps')
    plt.show()
    return 0

#read files
f1 = pd.read_csv('/Users/ccai28/Desktop/HW_1_training.txt',sep='\t', header = None, skiprows = 1)
f2 = pd.read_csv('/Users/ccai28/Desktop/HW_1_testing.txt',sep='\t', header = None, skiprows = 1)

#read testing data
X = f2.loc[:,[0,1]].values
Y = f2.loc[:,[2]].values
```

```

#read traing data and kernel density estimator
# y = 0
x1 = f1.loc[lambda f : f[2] == 0,[0,1]].values
#y = 1
x2 = f1.loc[lambda f : f[2] == 1,[0,1]].values

plot(X,x1,x2,10,Y,0.5)
plot(X,x1,x2,1,Y,0.5)
plot(X,x1,x2,0.1,Y,0.5)
```

Python codes of problem3:

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def distance(x,y):
    d = math.sqrt((x[0]-y[0])**2+(x[1]-y[1])**2) #distance between testing point and train point
    return d

def KNN(X,x,y,K):
    s = x.shape[0]
    S = X.shape[0]
    new_Y = np.zeros((S,1))
    for i in range(S):
        d = np.zeros((s,1))
        for j in range(s):
            d[j] = distance(X[i,:],x[j,:]) #distance for all the points respect with one point i in testing file
        index = np.argsort(d,axis = 0)      #sort according to the distance and index include index of distance
        count = 0                           #from smallest to longest
        for k in range(K):    #find k nearest points in training respect to testing i
            y_1 = y[index[k]] #find the class of testing point i with its index
            if y_1 == 0:      #if the nearest test points are in class 0, and more than half of these points are in
                count += 1    # class 0, the i training point is ->predict into class 0
        if count > K*0.5:     # else ->predict into class 1
            new_y = 0
        else:
            new_y = 1
        new_Y[i] = new_y      #build the new class of traiing points after classfied

    return new_Y

def misclassified_point(X,Y,new_Y):
    mis_0 = []
    mis_1 = []
    right_0 = []
    right_1 = []
    s = Y.shape[0]
    for i in range(s):
        if (Y[i] != new_Y[i] and new_Y[i] == 0):   #find original class 1 training point -> predict class 0
            mis_0.append(X[i,:])
        elif (Y[i] != new_Y[i] and new_Y[i] == 1): #find original class 0 training point -> predict class 1
            mis_1.append(X[i,:])
        elif (Y[i] == new_Y[i] and new_Y[i] == 0): #find original class 0
```

```
    training point -> predict class 0
            right_0.append(X[i,:])                #find original class 1
    training point -> predict class 1
        else:
            right_1.append(X[i,:])
    mis_0 = np.matrix(mis_0)
    right_0 = np.matrix(right_0)
    mis_1 = np.matrix(mis_1)
    right_1 = np.matrix(right_1)
    return mis_0,mis_1,right_0,right_1

def plot(X,Y,x,y,K):
    new_Y = KNN(X,x,y,K)

    #mis points
    mis_0,mis_1,right_0,right_1 = misclassified_point(X,Y,new_Y)


    plt.scatter(right_0[:,0],right_0[:,1],30,
            color='black', marker='o', label='healthy controls')
    plt.scatter(right_1[:,0],right_1[:,1],30,
            color='black', marker='x', label='disease cases')
    plt.scatter(mis_0[:,0],mis_0[:,1],30,
            color='red', marker='o', label='healthy controls,mis')
    plt.scatter(mis_1[:,0],mis_1[:,1],30,
            color='red', marker='x', label='disease cases,mis')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend(scatterpoints=1,
           loc='lower right',
           ncol=1,
           fontsize=7)
    plt.savefig('k={:d}.eps'.format(K),format='eps')
    plt.show()

    return 0

def parameters(X,Y,x,y,K):
    new_Y = KNN(X,x,y,K)
    right0 = [] #True Negtive, TN
    right1 = [] #True Positive, TP
    mis0 = [] #class 0 -> predict class 1 #False Negtive, FN
    mis1 = [] #class 1 -> predict class 0 #False Positive, FP

    for i in range(Y.shape[0]):
        if np.allclose(new_Y[i],0):
            # Class 0
            if np.allclose(new_Y[i],Y[i]): #Correct class0
                right0.append(new_Y[i])
            else:
                mis0.append(new_Y[i])
        else:
            if np.allclose(new_Y[i],Y[i]):
                right1.append(new_Y[i])
            else:
                mis1.append(new_Y[i])
    TN = len(right0)
    TP = len(right1)
    FN = len(mis0)
    FP = len(mis1)
    sensitivity = TP / float((TP + FN))
    specificity = TN / float((TN + FP))
    false_discovery_rate = FP / float((FP + TP))

    print('When k={:d}, \nsensitivity = {:f},\nspecificity={:f}, \nfalse
    discovery rate={:f} \n'.format(K,sensitivity,specificity,
    false_discovery_rate))
```

```
#read data
f1 = pd.read_csv('/Users/ccai28/Desktop/HW_1_training.txt',sep='\t', header =
    None, skiprows = 1)
f2 = pd.read_csv('/Users/ccai28/Desktop/HW_1_testing.txt',sep='\t', header =
    None, skiprows = 1)

#read testing data
X = f2.loc[:,[0,1]].values
Y = f2.loc[:,[2]].values

#read traing data
x = f1.loc[:,[0,1]].values
y = f1.loc[:,[2]].values

#parameters,k=1
parameters(X,Y,x,y,1)
#plot,k=1
plot(X,Y,x,y,1)

#k=5
parameters(X,Y,x,y,5)
plot(X,Y,x,y,5)

#k=10
parameters(X,Y,x,y,10)
plot(X,Y,x,y,10)
```