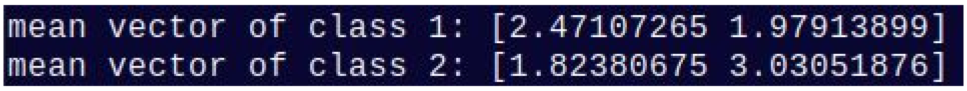
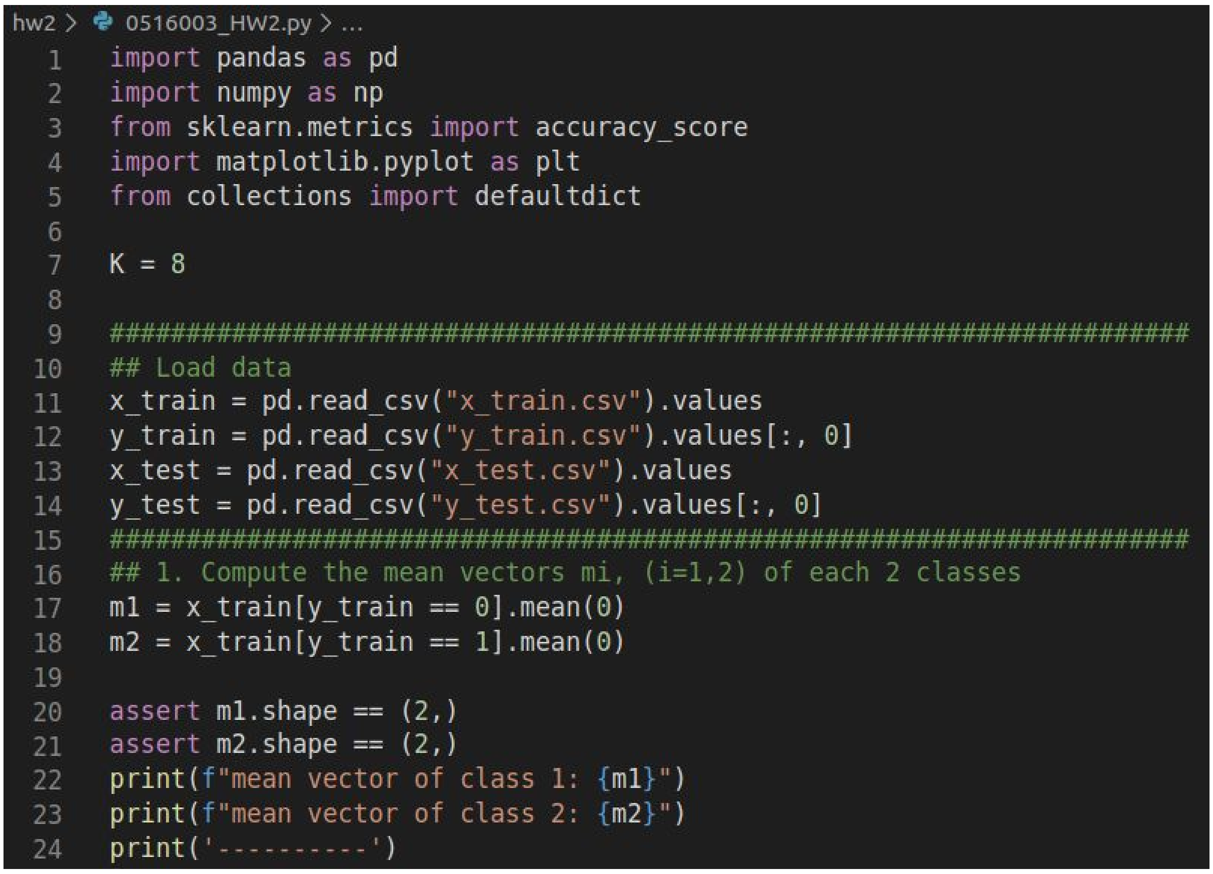
NCTU Pattern Recognition, Homework 2

Part 1, Coding (60%):

Q1: Compute the mean vectors m_i, (i=1,2) of each 2 classes on training data

- 

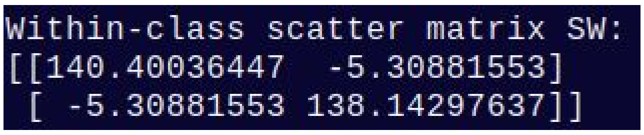
```
mean vector of class 1: [2.47107265 1.97913899]
mean vector of class 2: [1.82380675 3.03051876]
```

- 

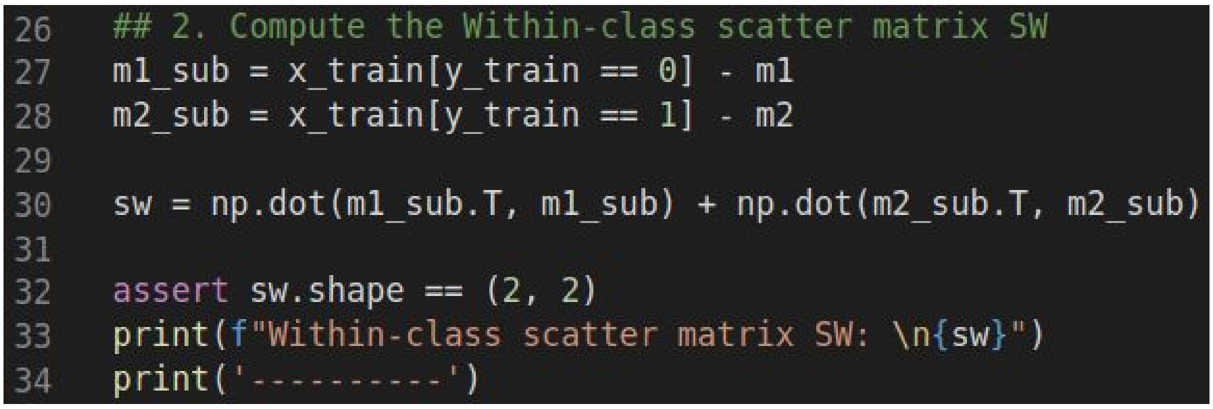
hw2 > 0516003_HW2.py > ...

```python
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from collections import defaultdict

K = 8

######################################################################
## Load data
x_train = pd.read_csv("x_train.csv").values
y_train = pd.read_csv("y_train.csv").values[:, 0]
x_test = pd.read_csv("x_test.csv").values
y_test = pd.read_csv("y_test.csv").values[:, 0]
######################################################################
## 1. Compute the mean vectors mi, (i=1,2) of each 2 classes
m1 = x_train[y_train == 0].mean(0)
m2 = x_train[y_train == 1].mean(0)

assert m1.shape == (2,)
assert m2.shape == (2,)
print(f"mean vector of class 1: {m1}")
print(f"mean vector of class 2: {m2}")
print('----------')
```

Q2: Compute the within-class scatter matrix S_W on training data

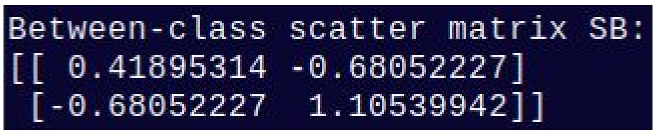
- 

```
Within-class scatter matrix SW:
[[140.40036447  -5.30881553]
 [ -5.30881553 138.14297637]]
```

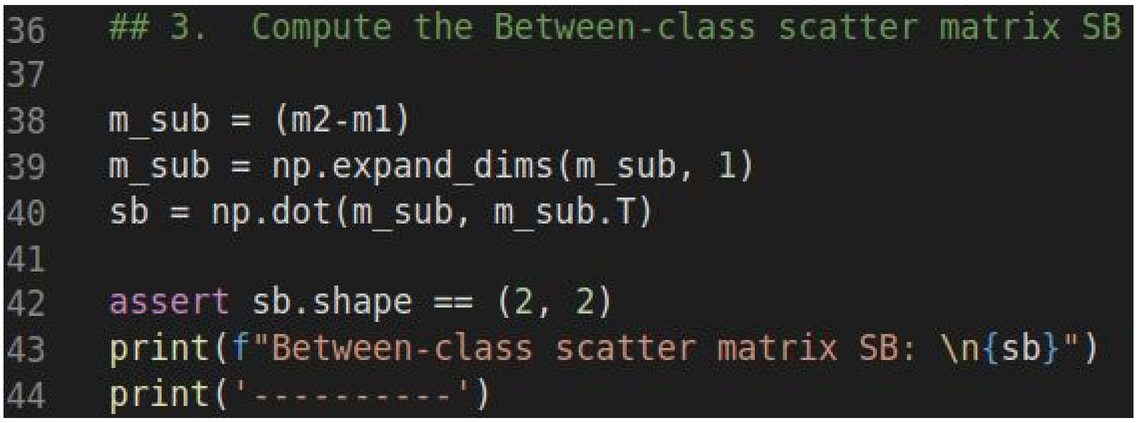
- 

```python
## 2. Compute the Within-class scatter matrix SW
m1_sub = x_train[y_train == 0] - m1
m2_sub = x_train[y_train == 1] - m2

sw = np.dot(m1_sub.T, m1_sub) + np.dot(m2_sub.T, m2_sub)

assert sw.shape == (2, 2)
print(f"Within-class scatter matrix SW: \n{sw}")
print('----------')
```

Q3: Compute the between-class scatter matrix S_B on training data

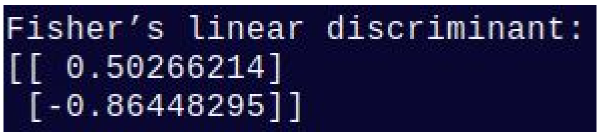
- 

```
Between-class scatter matrix SB:
[[ 0.41895314 -0.68052227]
 [-0.68052227  1.10539942]]
```

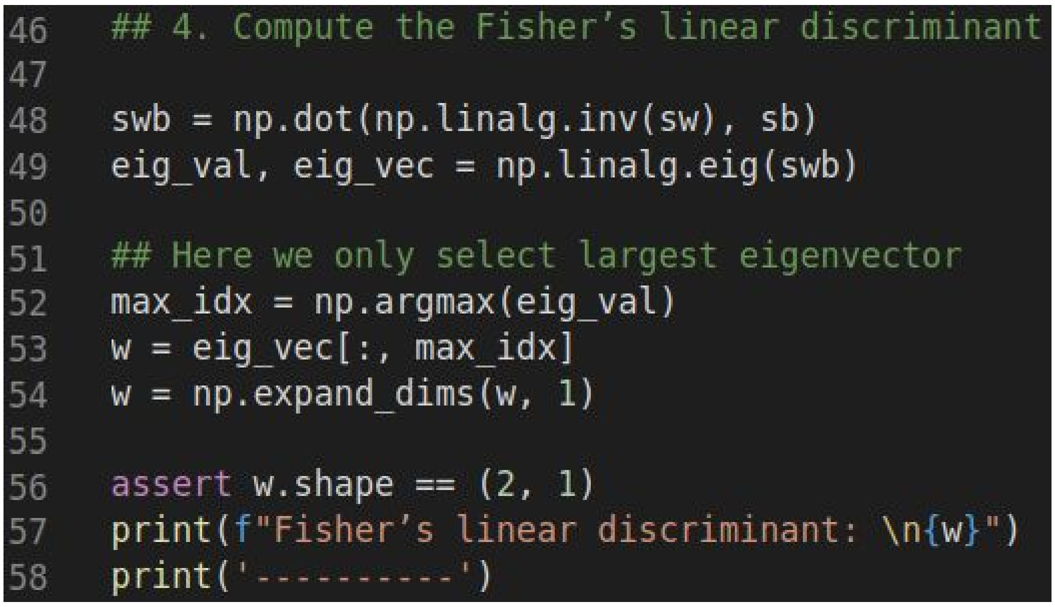
- 

```
## 3.  Compute the Between-class scatter matrix SB

m_sub = (m2-m1)
m_sub = np.expand_dims(m_sub, 1)
sb = np.dot(m_sub, m_sub.T)

assert sb.shape == (2, 2)
print(f"Between-class scatter matrix SB: \n{sb}")
print('----------')
```

Q4: Compute the Fisher’s linear discriminant W on training data

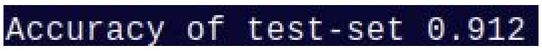
- 

```
Fisher’s linear discriminant:
[[ 0.50266214]
 [-0.86448295]]
```

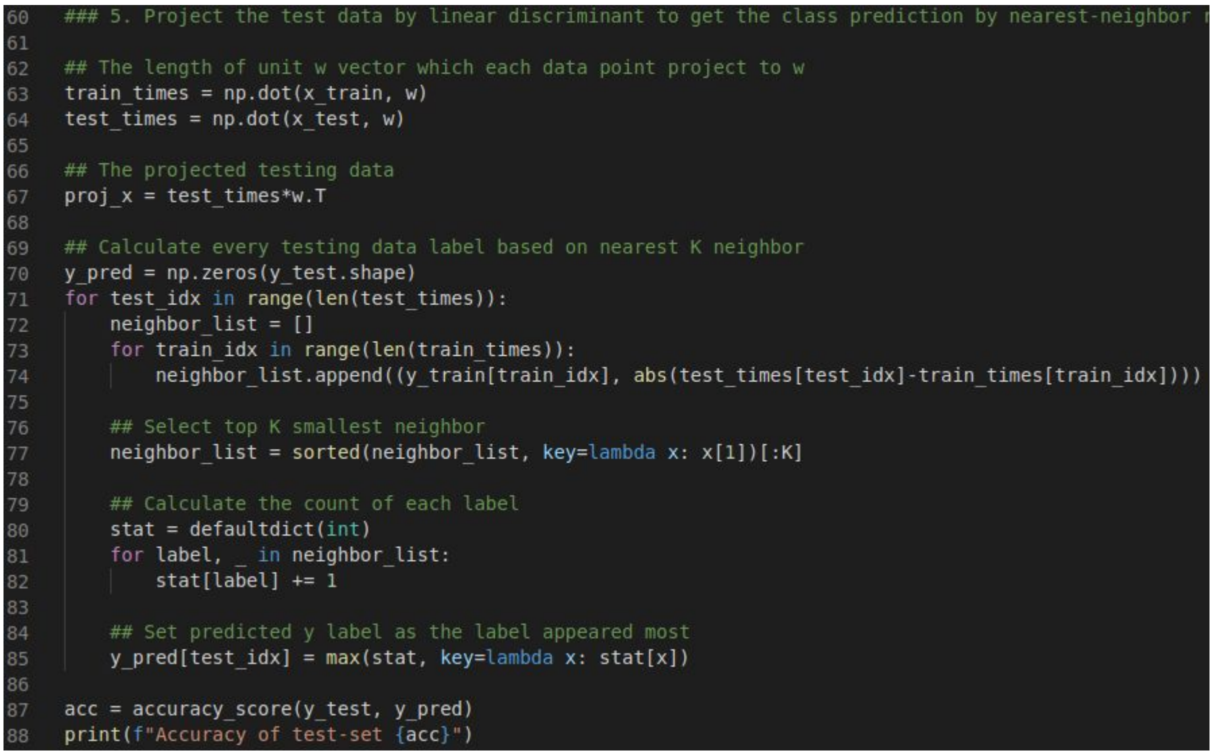
- 

```
## 4. Compute the Fisher’s linear discriminant

swb = np.dot(np.linalg.inv(sw), sb)
eig_val, eig_vec = np.linalg.eig(swb)

## Here we only select largest eigenvector
max_idx = np.argmax(eig_val)
w = eig_vec[:, max_idx]
w = np.expand_dims(w, 1)

assert w.shape == (2, 1)
print(f"Fisher’s linear discriminant: \n{w}")
print('----------')
```

Q5: Project the **testing data** by linear discriminant to get the class prediction by nearest-neighbor rule and calculate your accuracy score on **testing data**

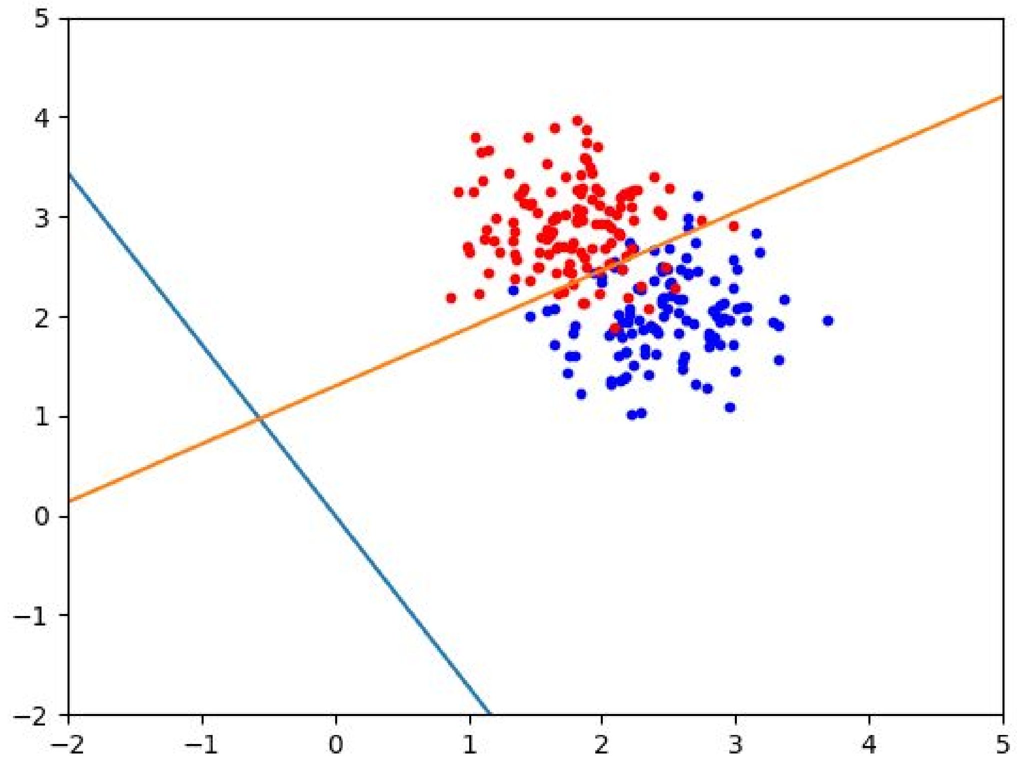
- 

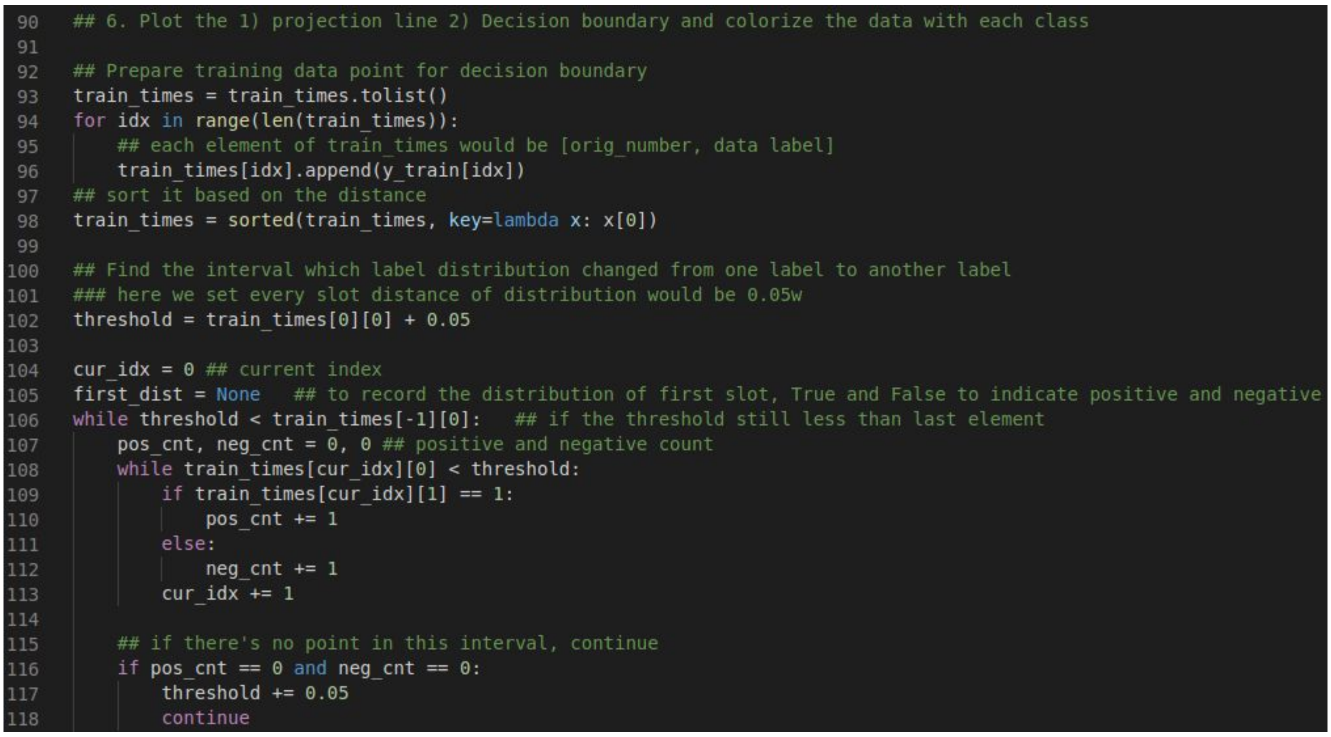
```
Accuracy of test-set 0.912
```

- 

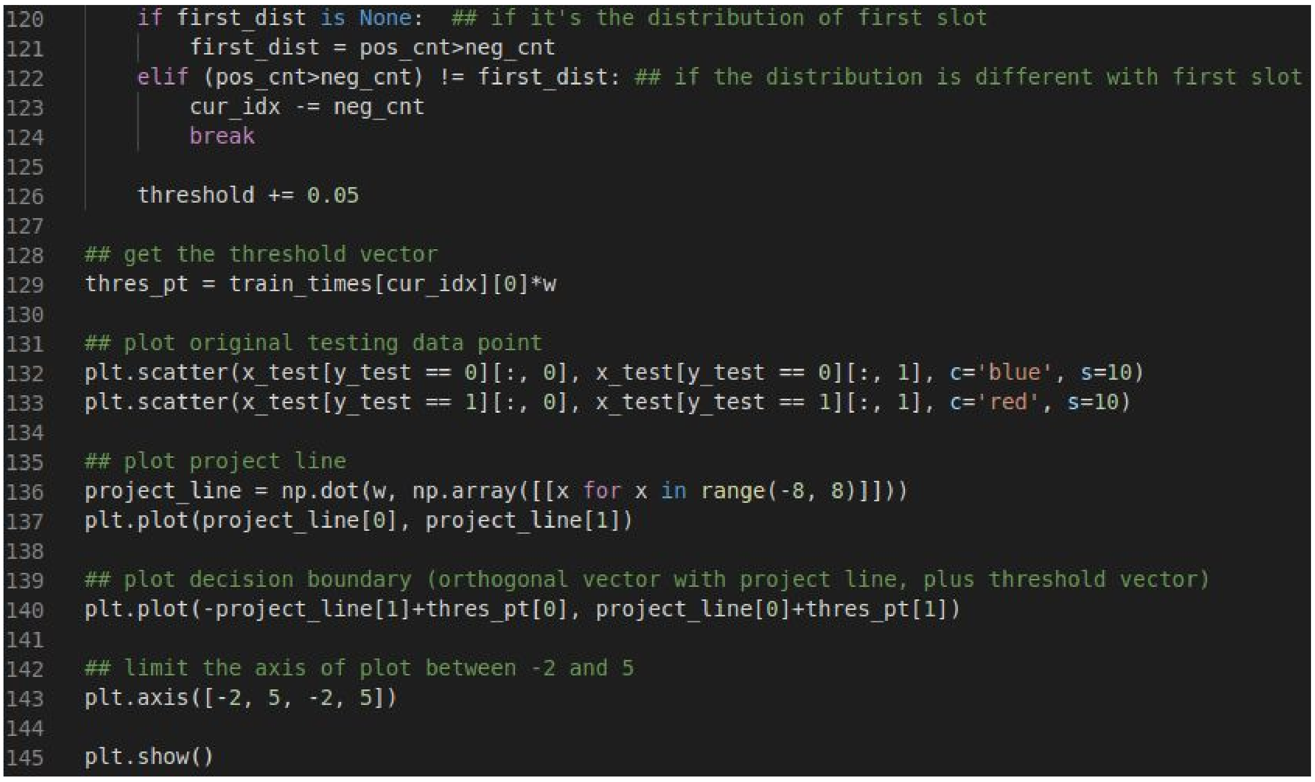
```python
### 5. Project the test data by linear discriminant to get the class prediction by nearest-neighbor

## The length of unit w vector which each data point project to w
train_times = np.dot(x_train, w)
test_times = np.dot(x_test, w)

## The projected testing data
proj_x = test_times*w.T

## Calculate every testing data label based on nearest K neighbor
y_pred = np.zeros(y_test.shape)
for test_idx in range(len(test_times)):
    neighbor_list = []
    for train_idx in range(len(train_times)):
        neighbor_list.append((y_train[train_idx], abs(test_times[test_idx]-train_times[train_idx])))

    ## Select top K smallest neighbor
    neighbor_list = sorted(neighbor_list, key=lambda x: x[1])[:K]

    ## Calculate the count of each label
    stat = defaultdict(int)
    for label, _ in neighbor_list:
        stat[label] += 1

    ## Set predicted y label as the label appeared most
    y_pred[test_idx] = max(stat, key=lambda x: stat[x])

acc = accuracy_score(y_test, y_pred)
print(f"Accuracy of test-set {acc}")
```

Q6: Plot the 1) **best projection line** 2) **decision boundary** on the **training data** and colorize the data with each class.





```python
## 6. Plot the 1) projection line 2) Decision boundary and colorize the data with each class

## Prepare training data point for decision boundary
train_times = train_times.tolist()
for idx in range(len(train_times)):
    ## each element of train_times would be [orig_number, data label]
    train_times[idx].append(y_train[idx])
## sort it based on the distance
train_times = sorted(train_times, key=lambda x: x[0])

## Find the interval which label distribution changed from one label to another label
### here we set every slot distance of distribution would be 0.05w
threshold = train_times[0][0] + 0.05

cur_idx = 0 ## current index
first_dist = None   ## to record the distribution of first slot, True and False to indicate positive and negative
while threshold < train_times[-1][0]:   ## if the threshold still less than last element
    pos_cnt, neg_cnt = 0, 0 ## positive and negative count
    while train_times[cur_idx][0] < threshold:
        if train_times[cur_idx][1] == 1:
            pos_cnt += 1
        else:
            neg_cnt += 1
        cur_idx += 1

    ## if there's no point in this interval, continue
    if pos_cnt == 0 and neg_cnt == 0:
        threshold += 0.05
        continue
```



```python
    if first_dist is None:  ## if it's the distribution of first slot
        first_dist = pos_cnt>neg_cnt
    elif (pos_cnt>neg_cnt) != first_dist: ## if the distribution is different with first slot
        cur_idx -= neg_cnt
        break

    threshold += 0.05

## get the threshold vector
thres_pt = train_times[cur_idx][0]*w

## plot original testing data point
plt.scatter(x_test[y_test == 0][:, 0], x_test[y_test == 0][:, 1], c='blue', s=10)
plt.scatter(x_test[y_test == 1][:, 0], x_test[y_test == 1][:, 1], c='red', s=10)

## plot project line
project_line = np.dot(w, np.array([[x for x in range(-8, 8)]]))
plt.plot(project_line[0], project_line[1])

## plot decision boundary (orthogonal vector with project line, plus threshold vector)
plt.plot(-project_line[1]+thres_pt[0], project_line[0]+thres_pt[1])

## limit the axis of plot between -2 and 5
plt.axis([-2, 5, -2, 5])

plt.show()
```

Part 2, Questions (40%):

Q1: Show that maximization of the class separation criterion given by $L(\lambda, w) = w^T (m2 - m1) + \lambda(w^T w - 1)$ with respect to w, using a Lagrange multiplier to enforce the constraint $w^T w = 1$, leads to the result that $w \propto (m2 - m1)$.

1、

$$\max L(\lambda, w) = w^T(m_2 - m_1) + \lambda(w^T w - 1)$$
$$= w^T m_2 - w^T m_1 + \lambda w^T w - \lambda$$

$$\frac{dL(\lambda, w)}{dw} = m_2 - m_1 + 2\lambda w = 0$$
$$\Rightarrow 2\lambda w = m_1 - m_2$$
$$\Rightarrow w = -\frac{1}{2\lambda}(m_2 - m_1)$$
$$\Rightarrow \underline{w \propto (m_2 - m_1)} \quad \#$$

Q2: Using (eq 1) and (eq 2), derive the result (eq 3) for the posterior class probability in the two-class generative model with Gaussian densities, and verify the results (eq 4) and (eq 5) for the parameters w and w0.

2、

Gaussian density : $P(x|C_i) = \frac{1}{(2\pi)^{\frac{D}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma^{-1}(x-\mu_i)}$

$$a = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} = \ln \frac{P(x|C_1)}{P(x|C_2)} + \ln \frac{P(C_1)}{P(C_2)}$$

$$= -\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T \Sigma^{-1}(x-\mu_2) + \ln\frac{P(C_1)}{P(C_2)}$$

$$= -\frac{1}{2}(x^T\Sigma^{-1}x - 2x^T\Sigma^{-1}\mu_1 + \mu_1^T\Sigma^{-1}\mu_1)$$
$$+ \frac{1}{2}(x^T\Sigma^{-1}x - 2x^T\Sigma^{-1}\mu_2 + \mu_2^T\Sigma^{-1}\mu_2) + \ln\frac{P(C_1)}{P(C_2)}$$

$$= x^T\Sigma^{-1}(\mu_1 - \mu_2) - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln\frac{P(C_1)}{P(C_2)}$$

$$= w^T x + w_0$$

where $w = \Sigma^{-1}(\mu_1 - \mu_2)$

$w_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln\frac{P(C_1)}{P(C_2)}$ #

$$\Rightarrow P(C_1|x) = \sigma(a) = \sigma(w^T x + w_0) \#$$