309551101 郭育麟

圖形識別 Pattern Recognition HW2

Part 1 Coding

1.1 Compute the mean vectors mi (i=1, 2) of each 2 classes on training data

1. Compute the mean vectors mi, (i=1,2) of each 2 classes

In [5]:
```
## Your code HERE
m1 = np.mean(x_train[y_train==0], axis = 0)
m2 = np.mean(x_train[y_train==1], axis = 0)
m_all = np.mean(x_train, axis = 0)
```

In [6]:
```
print(f"mean vector of class 1: {m1}", f"mean vector of class 2: {m2}")
```
```
mean vector of class 1: [ 1.3559426  -1.34746216] mean vector of class 2: [-1.29735587  1.29096203]
```

Use np.mean to calculate each mean .

1.2 Compute the within-class scatter matrix S_W training data

2. Compute the Within-class scatter matrix SW

In [7]:
```
## Your code HERE
sw1 = np.dot((x_train[y_train==0] - m1).T, (x_train[y_train==0] - m1))
sw2 = np.dot((x_train[y_train==1] - m2).T, (x_train[y_train==1] - m2))
sw = sw1 + sw2
```

In [8]:
```
assert sw.shape == (2,2)
print(f"Within-class scatter matrix SW: {sw}")
```
```
Within-class scatter matrix SW: [[ 388.64001349 -228.92177708]
 [-228.92177708  665.56910433]]
```

S_W represent the degree of separation of same class projection data points

$$S_W = \sum_{i=1}^{K} S_k$$

$$\text{where } S_k = \sum_{n \in C_k} (x_n - m_{C_k})(x_n - m_{C_k})^T$$

$$S_k \ can\ seen\ as\ k - class\ divergence\ matrix$$

1.3 Compute the between-class scatter matrix S_B on training data

3. Compute the Between-class scatter matrix SB

In [9]:
```
## Your code HERE
diff = (m2 - m1).reshape(-1,1)
#sb = np.dot(diff,diff.T)
sb = len(x_train[y_train==0]) * np.dot((m1-m_all).reshape(-1,1),(m1-m_all).reshape(-1,1).T) + len(
```

In [10]:
```
assert sb.shape == (2,2)
print(f"Between-class scatter matrix SB: {sb}")
```
```
Between-class scatter matrix SB: [[ 1319.66072786 -1312.26276299]
 [-1312.26276299  1304.90627081]]
```

S_B represent the degree of separation of different class projection data points

$$S_B = \sum_{i=1}^{K} Number_{C_k}(m_{C_k} - m)(m_{C_k} - m)^T$$

$$\text{where Number}_{C_k}\ is\ the\ number\ of\ k - class\ data\ points$$

$$\text{m}_{C_k}\ is\ the\ k - class\ mean\ vector$$

$$\text{m is the all data points mean vector}$$

1.4 Compute the Fisher's linear discriminant W training data

4. Compute the Fisher's linear discriminant

In [11]:

```
## Your code HERE
sw_inverse = np.linalg.inv(sw)
Array = np.dot(sw_inverse, sb)
eigenvalues, eigenvectors = np.linalg.eig(Array)
sort_index = np.argsort(-eigenvalues)
sort_index = sort_index[0]
eigenvalues = np.asarray(eigenvalues[sort_index].real , dtype = 'float')
eigenvectors = np.asarray(eigenvectors[:,sort_index].real , dtype = 'float')


w = eigenvectors
w = w.reshape(2,-1)
w = w*(-1)
print(f"w is \n{w}")
print("===========")


# w is proportional to sw_inverse * (m2-m1)
w_1 = np.dot(sw_inverse,m2-m1).reshape(2,-1)
w_1 = w_1 / np.sqrt(w_1[0]**2 + w_1[1]**2)
print(f"w_1 is \n{w_1}")
```

```
w is
[[-0.94096648]
 [ 0.33849976]]
===========
w_1 is
[[-0.94096648]
 [ 0.33849976]]
```

In [12]:

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

```
 Fisher's linear discriminant: [[-0.94096648]
 [ 0.33849976]]
```

We want S_B the bigger the better and want S_W the smaller the better . So the objective function will be $J(w) = \frac{w^T S_B W}{w^T S_W w}$, w represent the projection matrix . And by Rayleigh quotient and differential with respect to w

$$\frac{\partial J(w)}{\partial w} = 0 \ \rightarrow S_B w(w^T S_W w) = (w^T S_B w) S_W w$$

$$S_B w = \frac{w^T S_B w}{w^T S_W w} S_W w = J(w) S_W w$$

$$S_W^{-1} S_B w = J(w) w$$

$$\text{where } S_W^{-1} S_B\ is\ an\ array\ , J(w)\ is\ scalar$$

$$\text{it can seen as the defiction of eigenvalue and eigenvectors}$$

The equation $S_W^{-1} S_B W = \lambda W$, compute $S_W^{-1} S_B$ eigenvalue and eigenvectors , the optimal w is the eigenvalue of that corresponds to the largest eigenvalue .

I also try $w \propto S_W^{-1} * (m_2 - m_1)$, and the answer is "w_1"

1.5 Project the testing data by fisher's linear discriminant to get the class prediction by nearest-neighbor rule and calculate your accuracy score on testing data

In [13]:

```
from sklearn.metrics import accuracy_score

def KNN(k, train, label, test):
    belongs = np.zeros((test.shape[0]))
    for i in range(len(test)):
        dist = []
        for j in range(len(train)):
            distance = ((test[i] - train[j])**2)[0]
            dist.append(distance)
        sort_index = np.argsort(dist)
        sort_index = sort_index[0:k]
        c1 = np.count_nonzero(label[sort_index] == 0)
        c2 = np.count_nonzero(label[sort_index] == 1)
        if c1 > c2:
            belongs[i] = 0
        else:
            belongs[i] = 1
    return belongs

train_project = np.dot(x_train_plot, w)
test_project = np.dot(x_test_plot, w)
train_label = np.copy(y_train)
k = 1
y_pred = KNN(k, train_project, train_label, test_project)
acc = accuracy_score(y_test, y_pred)
```

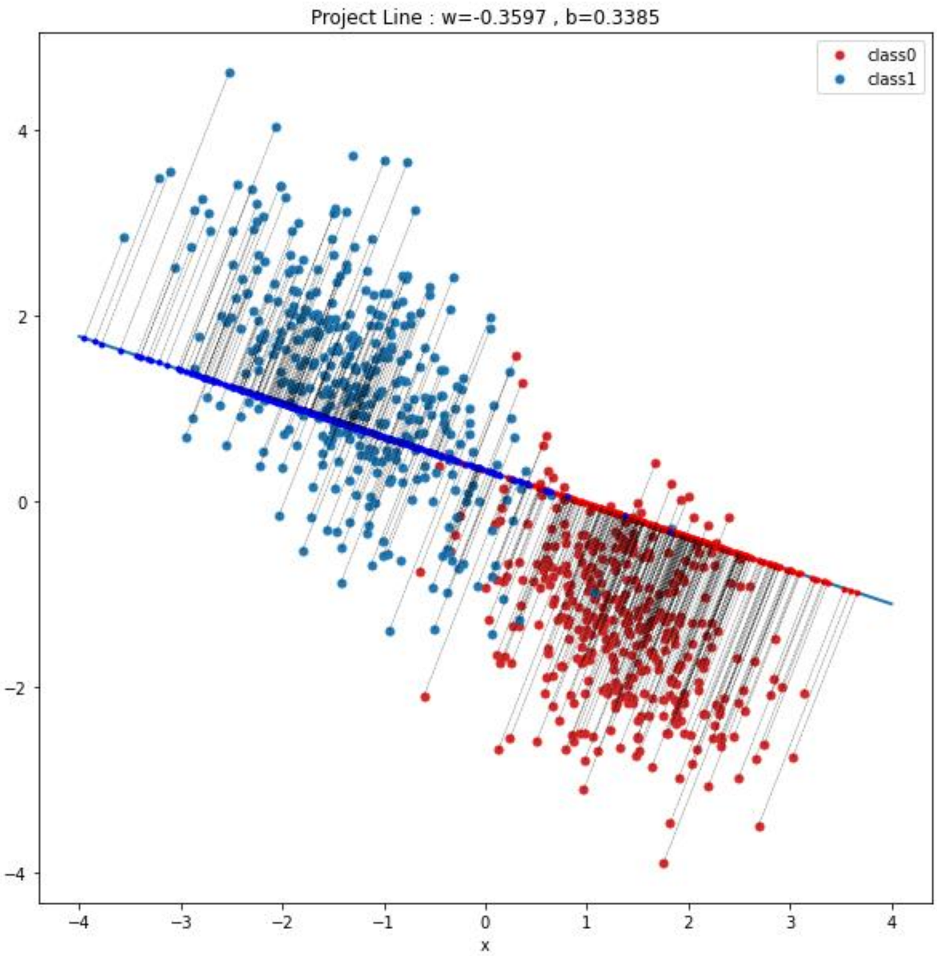
In [14]:

```
print(f"Accuracy of test-set {acc}")
```

```
Accuracy of test-set 0.916
```

First , project testing data and training data by using previous eigenvector , and I use KNN algorithm(k = 1) to classification and get 91.6% accuracy .

1.6 Plot the 1) best projection line on the training data and show the slope and intercept on the title (you can choose and value of intercept for better visualization) 2) colorize the data with each class 3) project all data points on your projection line .



Part2 Questions

2.1 Show that maximization of the class separation criterion given by $\mathbf{L}(\boldsymbol{\lambda}, \mathbf{w}) = \boldsymbol{w}^{\boldsymbol{T}}(\boldsymbol{m2} - \boldsymbol{m1}) + \boldsymbol{\lambda}(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{w} - \mathbf{1})$ with respect to w , using a Lagrange multiplier to enforce the constraint $\mathbf{w}^{\mathbf{T}}\boldsymbol{w} = \mathbf{1}$, leads to the result that $\mathbf{w}\ \boldsymbol{\alpha}\ (\mathbf{m2} - \mathbf{m1})$

$$\frac{\partial \mathrm{L}}{\partial w} = (m_2 - m_1) + 2 * \lambda * w$$

$$\mathrm{w} = -\frac{1}{2 * \lambda}(m_2 - m_1)$$

$$\therefore \mathrm{w}\ \alpha\ (\mathrm{m}_2 - m_1)$$

2.2 Show that the logistic sigmoid function satisfies the property

$\sigma(-a) = 1 - \sigma(a)$ and its inverse is given by $\sigma^{-1}(y) = \ln\{\frac{y}{1-y}\}$

(a)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma(-x) = \frac{1}{1+e^{x}}$$

$$1 - \sigma(x) = \frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} = \frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{x}}$$

(b)

$$y = \frac{1}{1+e^{-x}}$$

$$\rightarrow y(1+e^{-x}) = 1$$

$$\rightarrow y + y * e^{-x} = 1$$

$$\rightarrow e^{-x} = \frac{1-y}{y}$$

$$\rightarrow -x * \ln e = \ln\frac{1-y}{y}$$

$$\rightarrow x = \ln\left(\frac{y}{1-y}\right) = \sigma^{-1}(y)$$