109550017_HW2_黃品云

Part. 1, Coding (60%):

1. (5%) Compute the mean vectors mi (i=1, 2) of each 2 classes on **training data**

```
mean vector of class 1: [[ 0.99253136 -0.99115481]] mean vector of class 2: [[-0.9888012   1.00522778]]
```

```
# m1, m2 are the mean vector for class 1 and 2

# m1 = (1/N1)*sum(x in C1), N1 = the number of points in C1
# m2 = (1/N2)*sum(x in C2), N2 = the number of points in C2
m1 = np.mean(x_train[y_train == 0], axis = 0)
m2 = np.mean(x_train[y_train == 1], axis = 0)
```

```
print(f"mean vector of class 1: {m1}", f"mean vector of class 2: {m2}")
```

2. (5%) Compute the within-class scatter matrix SW on **training data**

```
Within-class scatter matrix SW: [[ 4337.38546493 -1795.55656547]
 [-1795.55656547  2834.75834886]]
```

```
# SW is the within-class covariance matrix
# SW = sum((x-m1)·(x-m1).T, x in C1) + sum((x-m2)·(x-m2).T, x in C2)

# first, we separate the data into class 1 and 2
x1 = x_train[y_train == 0]
x2 = x_train[y_train == 1]

# SW = sum((x-m1).T·(x-m1), x in C1) + sum((x-m2).T·(x-m2), x in C2)
# I changed the order the transpose to make the shape consistent
SW = np.dot((x1 - m1).T, (x1 - m1)) + np.dot((x2 - m2).T, (x2 - m2))
```

```
print(f"Within-class scatter matrix SW: {SW}")
```

3. (5%) Compute the between-class scatter matrix SB on **training data**

```
Between-class scatter matrix SB: [[ 3.92567873 -3.95549783]
 [-3.95549783  3.98554344]]
```

```
# SB is the between-class covariance matrix
# SB = (m2 - m1)·(m2 - m1).T
# I changed the order the transpose to make the shape consistent

SB = np.dot((m2 - m1).T, (m2 - m1))
```

```
print(f"Between-class scatter matrix SB: {SB}")
```

4. (5%) Compute the Fisher's linear discriminant ˬ on **training data**

```
Fisher’s linear discriminant: [[ 0.37003809]
 [-0.92901658]]
```

```
# The optimal W is the eigenvector of inv(SW)·SB that corresponds to
# the largest eigenvalue

# compute inv(SW)·SB
inv_SW = np.linalg.inv(SW)
A = np.dot(inv_SW, SB)
# get eigenvalue and eigenvector
eigenvalue, eigenvector = np.linalg.eig(A)
# the optimal W is the eigenvector corresponds to the largest eigenvalue
max_eigenvalue_index = np.argmax(eigenvector)
W = eigenvector[:, max_eigenvalue_index]
```

```
print(f" Fisher’s linear discriminant: {W}")
```

5. (20%) Project the **testing data** by Fisher's linear discriminant to get the class prediction by K-Nearest-Neighbor rule and report the accuracy score on **testing data** with K values from 1 to 5 (you should get accuracy over 0.9)

```
Accuracy of test-set 0.8488
Accuracy of test-set 0.8488
Accuracy of test-set 0.8792
Accuracy of test-set 0.8824
Accuracy of test-set 0.8912
```

```python
# compute the distance of x1, x2
def euclidean_distance(x1, x2):
    distance = np.sqrt(np.sum((x1 - x2)**2))
    return distance

# for single element
def _predict(t, K):

    # compute the distance
    distances = [euclidean_distance(t, x) for x in train]
    # get the closest K
    K_indices = np.argsort(distances)[:K]
    K_nearest_labels = [y_train[i] for i in K_indices]
    # majority vote
    pred = max(K_nearest_labels, key = K_nearest_labels.count)
    return pred

# for whole set
def predict(X, K):
    y_pred = [_predict(x, K) for x in X]
    return y_pred
```

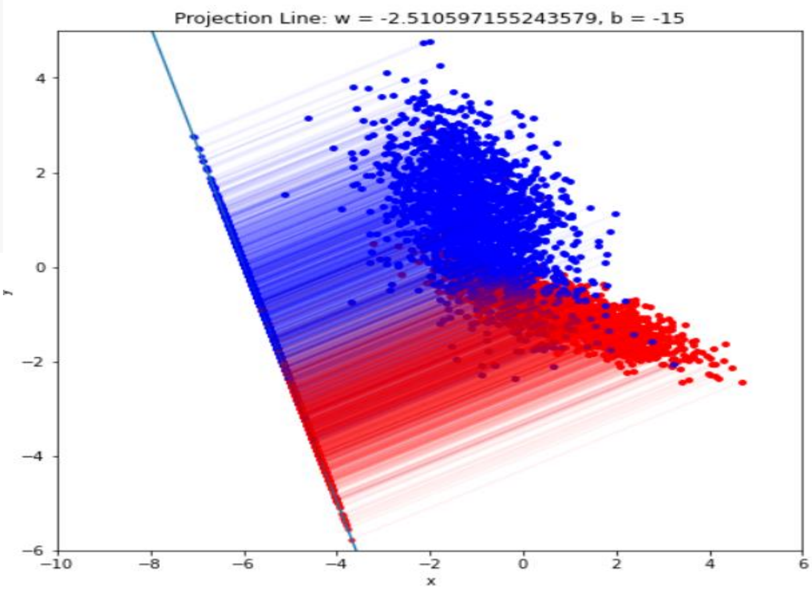
```python
for i in range(1,6):
    y_pred = predict(test, i)
    acc = accuracy_score(y_test, y_pred)
    print(f"Accuracy of test-set {acc}")
```

6. (20%) Plot the **1) best projection line** on the **training data** and show the slope and intercept on the title *(you can choose any value of* ***intercept*** *for better visualization)*
   2) colorize the data with each class **3)** project all data points on your projection line. Your result should look like the below image (This image is for reference, not the answer)

```python
# projection line: ax+by-+c=0
# the point we want to project: (px, py)
# projection point on line: (px - a*(a*px+b*py+c)/(a**2+b**2), py - b*(a*px+b*py+c)/(a**2+b**2))

x = x_train[i][0]
y = x_train[i][1]
a = W[1][0]
b = -W[0][0]
c = -15*W[0][0]

# point after projection
p_x = x - a*(a*x+b*y+c)/(a**2+b**2)
p_y = y - b*(a*x+b*y+c)/(a**2+b**2)
projection = np.array([p_x, p_y])
```

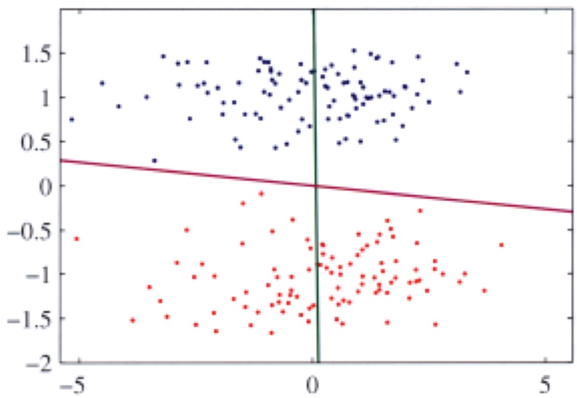


(10%) 1. What's the difference between the Principle Component Analysis and Fisher's Linear Discriminant?

- PCA is unsupervised and depends only on the value x_n whereas Fisher linear discriminant also uses class_label information.

- PCA focuses on capturing the direction of maximum variation in the data set, while FLD focuses on finding a feature subspace that maximizes the separability between the groups.

- The disparity between the data group is modeled by FLD, while the PCA does not detect such a disparity between groups.



PCA: magenta curve
FLD: green curve

PCA chooses the direction of maximum variance, which leads to strong class overlap.

FLD takes account of the class labels and leads to a projection onto the green curve giving much better class separation

(10%) 2. Please explain in detail how to extend the 2-class FLD into multi-class FLD (the number of classes is greater than two).

Assume D is dimensionality, K is class ($K>2$).

We introduce $D'>1$ linear "features" $y_k = \mathbf{w}_k^T \mathbf{x}$ where $k=1,\dots,D'$. These feature values can conveniently be grouped together to form a vector $\mathbf{y}$. Similarly, the weight vectors $\{\mathbf{w}_k\}$ can be considered to be the columns of a matrix $\mathbf{W}$.

so we can get: $y = W^T x$

The generalization of the within-class covariance matrix to the case of K classes:

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

where

$$S_k = \sum_{n \in C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T$$

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{x}_n$$, N_k is the number of patterns in class C_k.

In order to find a generalization of the between-class covariance matrix, we follow Duda and Hart (1973) and consider first the total covariance matrix.

$$S_T = \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{m})(\mathbf{x}_n - \mathbf{m})^T, \quad N = \Sigma_k N_k$$

where $\mathbf{m}$ is the mean of the total data set

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n = \frac{1}{N} \sum_{k=1}^{K} N_k \mathbf{m}_k, \quad N = \Sigma_k N_k$$

The total covariance matrix can be composed into the sum of the within-class covariance matrix plus between-class covariance matrix.

$$S_T = S_W + S_B$$

where $$S_B = \sum_{k=1}^{K} N_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T$$

These covariance matrices have been defined in the original x-space. We can now define similar matrices in the projected D'-dimensional y-space.

$$S_W = \sum_{k=1}^{K} \sum_{n \in C_k} (\mathbf{y}_n - \mu_k)(\mathbf{y}_n - \mu_k)^T$$

and

$$S_B = \sum_{k=1}^{K} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

where

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{y}_n \qquad \boldsymbol{\mu} = \frac{1}{N} \sum_{k=1}^{K} N_k \boldsymbol{\mu}_k$$

Again we wish to construct a scalar that is large when the between-class covariance is large and when the within-class covariance is small.

$$J(\mathbf{W}) = \mathrm{Tr}\left\{ S_W^{-1} S_B \right\}$$

This criterion can be rewritten as an explicit function of the projection matrix $\mathbf{W}$ in the form

$$J(\mathbf{w}) = \mathrm{Tr}\left\{ (\mathbf{W} \mathbf{S}_W \mathbf{W}^T)^{-1} (\mathbf{W} \mathbf{S}_B \mathbf{W}^T) \right\}$$

(6%) 3. By making use of Eq (1) ~ Eq (5), show that the Fisher criterion Eq (6) can be written in the form Eq (7).

$$y = \mathbf{w}^{\mathrm{T}}\mathbf{x} \qquad \text{Eq (1)}$$

$$\mathbf{m}_1 = \frac{1}{N_1}\sum_{n\in\mathcal{C}_1}\mathbf{x}_n \qquad \mathbf{m}_2 = \frac{1}{N_2}\sum_{n\in\mathcal{C}_2}\mathbf{x}_n \qquad \text{Eq (2)}$$

$$m_2 - m_1 = \mathbf{w}^{\mathrm{T}}(\mathbf{m}_2 - \mathbf{m}_1) \qquad \text{Eq (3)}$$

$$m_k = \mathbf{w}^{\mathrm{T}}\mathbf{m}_k \qquad \text{Eq (4)}$$

$$s_k^2 = \sum_{n\in\mathcal{C}_k}(y_n - m_k)^2 \qquad \text{Eq (5)}$$

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \qquad \text{Eq (6)}$$

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{B}}\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{W}}\mathbf{w}} \qquad \text{Eq (7)}$$

$$J(w) = \frac{(m_2-m_1)^2}{s_1^2+s_2^2} = \frac{(w^Tm_2 - w^Tm_1)^2 \text{ — ①}}{\sum_{n\in C_1}(y_n-m_1)^2 + \sum_{n\in C_2}(y_n-m_2)^2 \text{ — ②}}$$

①:
$$\begin{aligned}(w^Tm_2 - w^Tm_1)^2 &= (w^Tm_2-w^Tm_1)(w^Tm_2-w^Tm_1)^T \\ &= (w^Tm_2-w^Tm_1)[w^T(m_2-m_1)]^T \\ &= w^T(m_2-m_1)(m_2-m_1)^Tw\end{aligned}$$

②:
$$\begin{aligned}&\sum_{n\in C_1}(y_n-m_1)^2 + \sum_{n\in C_2}(y_n-m_2)^2 \\ &= \sum_{n\in C_1}(w^Tx_n - w^Tm_1)^2 + \sum_{n\in C_2}(w^Tx_n - w^Tm_2)^2 \\ &\overset{\text{apply the result from ①}}{=} \sum_{n\in C_1} w^T(x_n-m_1)(x_n-m_1)^Tw + \sum_{n\in C_2} w^T(x_n-m_2)(x_n-m_2)^Tw \\ &= w^T\Big[\sum_{n\in C_1}(x_n-m_1)(x_n-m_1)^T + \sum_{n\in C_2}(x_n-m_2)(x_n-m_2)^T\Big]w\end{aligned}$$

assume
$$S_B = (m_2-m_1)(m_2-m_1)^T$$
$$S_W = \sum_{n\in C_1}(x_n-m_1)(x_n-m_1)^T + \sum_{n\in C_2}(x_n-m_2)(x_n-m_2)^T$$

and apply ①, ②, we can get

$$J(w) = \frac{(m_2-m_1)^2}{s_1^2+s_2^2}$$
$$= \frac{w^T(m_2-m_1)(m_2-m_1)^Tw}{w^T\Big[\sum_{n\in C_1}(x_n-m_1)(x_n-m_1)^T + \sum_{n\in C_2}(x_n-m_2)(x_n-m_2)^T\Big]w} = \frac{w^TS_Bw}{w^TS_Ww}$$

#

(7%) 4. Show the derivative of the error function Eq (8) with respect to the activation a_k for an output unit having a logistic sigmoid activation function satisfies Eq (9).

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \qquad \text{Eq (8)}$$

$$\frac{\partial E}{\partial a_k} = y_k - t_k \qquad \text{Eq (9)}$$

We know that $y_k = \frac{1}{1+e^{-a_k}}$

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial a_k} = -\left(\frac{t_k}{y_k} + \frac{t_k - 1}{1 - y_k}\right) \cdot \frac{e^{-a_k}}{(1+e^{-a_k})^2}$$

$$= -\frac{t_k(1-y_k) + y_k(t_k - 1)}{y_k(1-y_k)} \cdot y_k(1-y_k)$$

$$= -\frac{t_k - y_k}{y_k(1-y_k)} \cdot y_k(1-y_k) = y_k - t_k \quad \#$$

(7%) 5. Show that maximizing likelihood for a multiclass neural network model in which the network outputs have the interpretation $y_k(x, w) = p(t_k = 1 \mid x)$ is equivalent to the minimization of the cross-entropy error function Eq (10).

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) \qquad \text{Eq (10)}$$

The binary target variables $t_k \in \{0,1\}$ have a 1-of-K coding scheme indicating the class, and the network outputs are interpreted as $y_k(\mathbf{X}, \mathbf{W}) = p(t_k = 1 \mid \mathbf{X})$,

then we can get $p(\mathbf{t} \mid \mathbf{X}, \mathbf{W}) = \prod_{k=1}^{K} y_k(\mathbf{X}, \mathbf{W})^{t_k}$

Taking the negative logarithm of the corresponding likelihood function then gives the following error function

$$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{X}_n, \mathbf{W})$$

Since it's "negative logarithm", we convert the statement from "maximizing" likelihood function into "minimizing" the error function.