

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 > ...
1 import pandas as pd
2 import numpy as np
3 from sklearn.metrics import accuracy_score
4 import matplotlib.pyplot as plt
5 from collections import defaultdict
6
7 K = 8
8
9 #####
10 ## Load data
11 x_train = pd.read_csv("x_train.csv").values
12 y_train = pd.read_csv("y_train.csv").values[:, 0]
13 x_test = pd.read_csv("x_test.csv").values
14 y_test = pd.read_csv("y_test.csv").values[:, 0]
15 #####
16 ## 1. Compute the mean vectors m_i , ($i=1,2$) of each 2 classes
17 m1 = x_train[y_train == 0].mean(0)
18 m2 = x_train[y_train == 1].mean(0)
19
20 assert m1.shape == (2,)
21 assert m2.shape == (2,)
22 print(f"mean vector of class 1: {m1}")
23 print(f"mean vector of class 2: {m2}")
24 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]]

- ```
26 ## 2. Compute the Within-class scatter matrix SW
27 m1_sub = x_train[y_train == 0] - m1
28 m2_sub = x_train[y_train == 1] - m2
29
30 sw = np.dot(m1_sub.T, m1_sub) + np.dot(m2_sub.T, m2_sub)
31
32 assert sw.shape == (2, 2)
33 print(f"Within-class scatter matrix SW: \n{sw}")
34 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]]
```

-

```
36  ## 3.  Compute the Between-class scatter matrix SB  
37  
38  m_sub = (m2-m1)  
39  m_sub = np.expand_dims(m_sub, 1)  
40  sb = np.dot(m_sub, m_sub.T)  
41  
42  assert sb.shape == (2, 2)  
43  print(f"Between-class scatter matrix SB: \n{sb}")  
44  print('-----')
```

-

Q4: Compute the Fisher's linear discriminant W on [training data](#)

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

-

```
46  ## 4. Compute the Fisher's linear discriminant  
47  
48  swb = np.dot(np.linalg.inv(sw), sb)  
49  eig_val, eig_vec = np.linalg.eig(swb)  
50  
51  ## Here we only select largest eigenvector  
52  max_idx = np.argmax(eig_val)  
53  w = eig_vec[:, max_idx]  
54  w = np.expand_dims(w, 1)  
55  
56  assert w.shape == (2, 1)  
57  print(f"Fisher's linear discriminant: \n{w}")  
58  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](#)

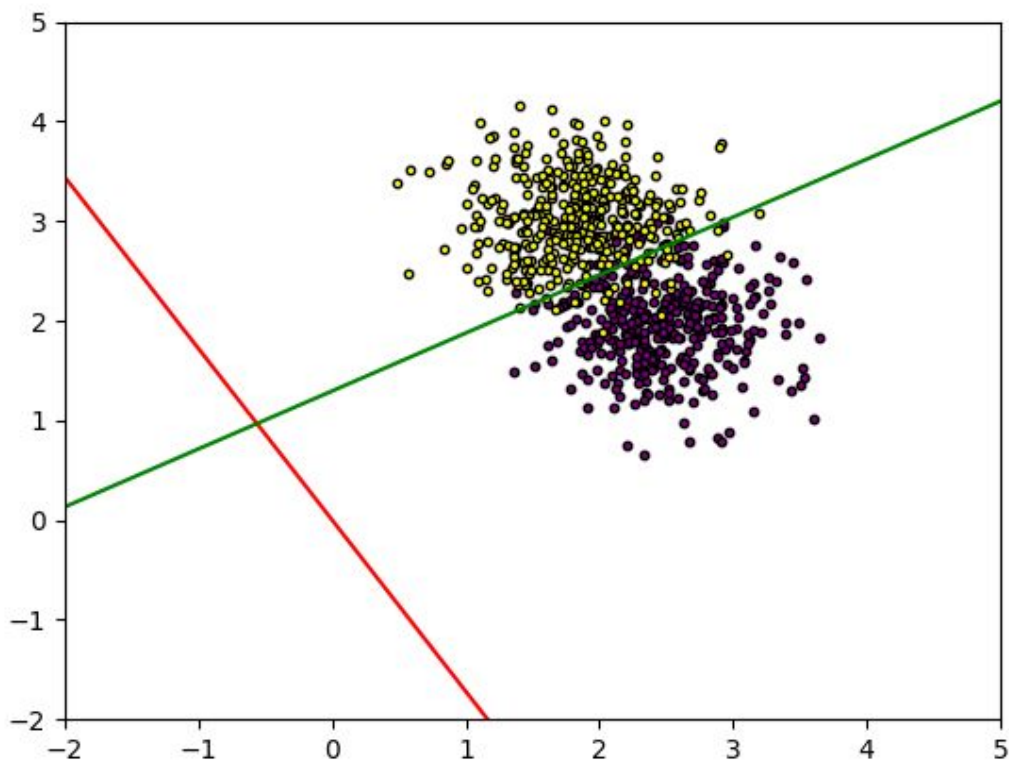
- Accuracy of test-set 0.912

```

60  ### 5. Project the test data by linear discriminant to get the class prediction by nearest-neighbor
61
62  ## The length of unit w vector which each data point project to w
63  train_times = np.dot(x_train, w)
64  test_times = np.dot(x_test, w)
65
66  ## The projected testing data
67  proj_x = test_times*w.T
68
69  ## Calculate every testing data label based on nearest K neighbor
70  y_pred = np.zeros(y_test.shape)
71  for test_idx in range(len(test_times)):
72      neighbor_list = []
73      for train_idx in range(len(train_times)):
74          neighbor_list.append((y_train[train_idx], abs(test_times[test_idx]-train_times[train_idx])))
75
76      ## Select top K smallest neighbor
77      neighbor_list = sorted(neighbor_list, key=lambda x: x[1])[:K]
78
79      ## Calculate the count of each label
80      stat = defaultdict(int)
81      for label, _ in neighbor_list:
82          stat[label] += 1
83
84      ## Set predicted y label as the label appeared most
85      y_pred[test_idx] = max(stat, key=lambda x: stat[x])
86
87  acc = accuracy_score(y_test, y_pred)
88  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.




```

90  ## 6. Plot the 1) projection line 2) Decision boundary and colorize the data with each class
91
92  ## Prepare training data point for decision boundary
93  train_times = train_times.tolist()
94  for idx in range(len(train_times)):
95      ## each element of train_times would be [orig_number, data label]
96      train_times[idx].append(y_train[idx])
97  ## sort it based on the distance
98  train_times = sorted(train_times, key=lambda x: x[0])
99
100 ## Find the interval which label distribution changed from one label to another label
101 ### here we set every slot distance of distribution would be 0.05w
102 threshold = train_times[0][0] + 0.05
103
104 cur_idx = 0 ## current index
105 first_dist = None ## to record the distribution of first slot, True and False to indicate positive and negative
106 while threshold < train_times[-1][0]: ## if the threshold still less than last element
107     pos_cnt, neg_cnt = 0, 0 ## positive and negative count
108     while train_times[cur_idx][0] < threshold:
109         if train_times[cur_idx][1] == 1:
110             pos_cnt += 1
111         else:
112             neg_cnt += 1
113         cur_idx += 1
114
115     ## if there's no point in this interval, continue
116     if pos_cnt == 0 and neg_cnt == 0:
117         threshold += 0.05
118         continue
119
120     if first_dist is None: ## if it's the distribution of first slot
121         first_dist = pos_cnt > neg_cnt
122     elif (pos_cnt > neg_cnt) != first_dist: ## if the distribution is different with first slot
123         cur_idx -= neg_cnt
124         break
125
126     threshold += 0.05
127
128 ## get the threshold vector
129 thres_pt = train_times[cur_idx][0]*w
130
131 ## plot original training data point
132 plt.scatter(x_train[y_train == 0][:, 0], x_train[y_train == 0][:, 1], c='purple', s=10, edgecolors='black')
133 plt.scatter(x_train[y_train == 1][:, 0], x_train[y_train == 1][:, 1], c='yellow', s=10, edgecolors='black')
134
135 ## plot project line
136 project_line = np.dot(w, np.array([[x for x in range(-8, 8)]]))
137 plt.plot(project_line[0], project_line[1], c='red')
138
139 ## plot decision boundary (orthogonal vector with project line, plus threshold vector)
140 plt.plot(-project_line[1]+thres_pt[0], project_line[0]+thres_pt[1], c='green')
141
142 ## limit the axis of plot between -2 and 5
143 plt.axis([-2, 5, -2, 5])
144
145 plt.show()

```

Part 2, Questions (40%):

Q1: Show that maximization of the class separation criterion given by

$L(\lambda, w) = w^T (m_2 - m_1) + \lambda(w^T w - 1)$ with respect to w , using a Lagrange multiplier to enforce the constraint $w^T w = 1$, leads to the result that $w \propto (m_2 - m_1)$.

1.

$$\begin{aligned} \max_w L(\lambda, w) &= w^T (m_2 - m_1) + \lambda(w^T w - 1) \\ &= w^T m_2 - w^T m_1 + \lambda w^T w - \lambda \end{aligned}$$

$$\frac{\partial L(\lambda, w)}{\partial w} = m_2 - m_1 + 2\lambda w = 0$$

$$\Rightarrow 2\lambda w = m_1 - m_2$$

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

$$\Rightarrow \underline{w \propto (m_2 - m_1)} \quad \#$$

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 w_0 .

2.

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

$$\alpha = \ln \frac{p(x|c_1)p(c_1)}{p(x|c_2)p(c_2)} = \ln \frac{p(x|c_1)}{p(x|c_2)} + \ln \frac{p(c_1)}{p(c_2)}$$

$$= -\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T \Sigma^{-1}(x-\mu_2) + \ln \frac{p(c_1)}{p(c_2)}$$

$$= -\frac{1}{2}(x^T \Sigma^{-1} x - 2x^T \Sigma^{-1} \mu_1 + \mu_1^T \Sigma^{-1} \mu_1)$$

$$+ \frac{1}{2}(x^T \Sigma^{-1} x - 2x^T \Sigma^{-1} \mu_2 + \mu_2^T \Sigma^{-1} \mu_2) + \ln \frac{p(c_1)}{p(c_2)}$$

$$= x^T \Sigma^{-1} (\mu_1 - \mu_2) - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + \ln \frac{p(c_1)}{p(c_2)}$$

$$= w^T x + w_0$$

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

$$w_0 = -\frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + \ln \frac{p(c_1)}{p(c_2)} \quad \#$$

$$\Rightarrow p(c_1|x) = \sigma(\alpha) = \sigma(w^T x + w_0) \quad \#$$