AML Homework

Chandrima Bhattacharya

Zeyu Wang

In [16]:

```
"""Problem 1"""
```

Out[16]:

```
'Problem 1'
```

In [1]:

```
import numpy as np
from scipy import misc
from matplotlib import pylab as plt
import matplotlib.cm as cm
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import os
%matplotlib inline
```

In [2]:

```
# a: Set path to downloaded dataset
os.chdir("F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW2")
print (os.getcwd())
```

```
F:\Annie\CornellMS\Semester 4\Machine Learning\Homework\HW2
```

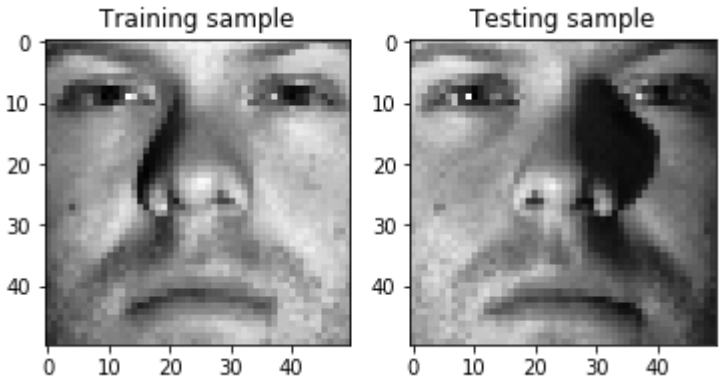
In [3]:

```
# b: Load training and test set into matrix
def load_data(path):
    data_values, label_values = [], []
    with open(path) as f:
        for line in f:
            im = misc.imread(line.strip().split()[0])
            data_values.append(im.reshape(2500,))
            label_values.append(line.strip().split()[1])
    data_values, label_values = np.array(data_values, dtype=float), np.array(label_valu
es, dtype=int)
    return data_values, label_values

train_data, train_labels = load_data('./faces/train.txt')
test_data, test_labels = load_data('./faces/test.txt')

print ("Training dataset shape is ", train_data.shape)
print ("Testing dataset shape is ", test_data.shape)
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: De
precationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.


Training dataset shape is  (540, 2500)
Testing dataset shape is  (100, 2500)
```

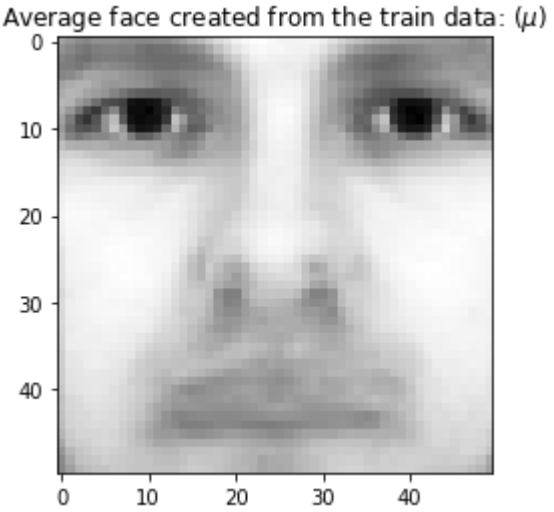
In [4]:

```
# Display random picture from train and test data
plt.subplot(1, 2, 1)
plt.imshow(train_data[1, :].reshape(50,50), cmap = cm.Greys_r)
plt.title('Training sample')

plt.subplot(1, 2, 2)
plt.imshow(test_data[1, :].reshape(50,50), cmap = cm.Greys_r)
plt.title('Testing sample')
plt.show()
```



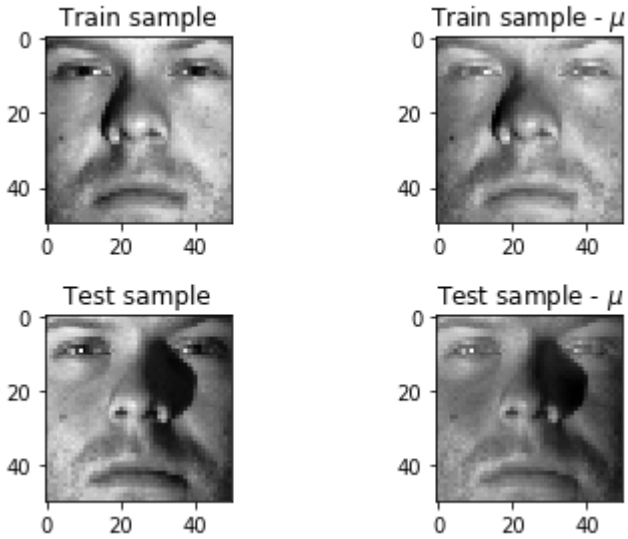
In [5]:

```
# c: Average face for training data
mu = np.apply_along_axis(np.mean, 0, train_data)
plt.imshow(mu.reshape(50,50), cmap = cm.Greys_r)
plt.title('Average face created from the train data: ($\mu$)')
plt.show()
```



In [6]:

```
# d: Subtract mu from all train and test samples
train_avg = train_data - mu
test_avg = test_data - mu

plt.subplot(2,2,1)
plt.imshow(train_data[1, :].reshape(50,50), cmap = cm.Greys_r)
plt.title("Train sample")

plt.subplot(2,2,2)
plt.imshow(train_avg[1,:].reshape(50,50), cmap = cm.Greys_r)
plt.title("Train sample - $\mu$")

plt.subplot(2,2,3)
plt.imshow(test_data[1,:].reshape(50,50), cmap = cm.Greys_r)
plt.title("Test sample")

plt.subplot(2,2,4)
plt.imshow(test_avg[1,:].reshape(50,50), cmap = cm.Greys_r)
plt.title("Test sample - $\mu$")

plt.tight_layout()
plt.show()
```



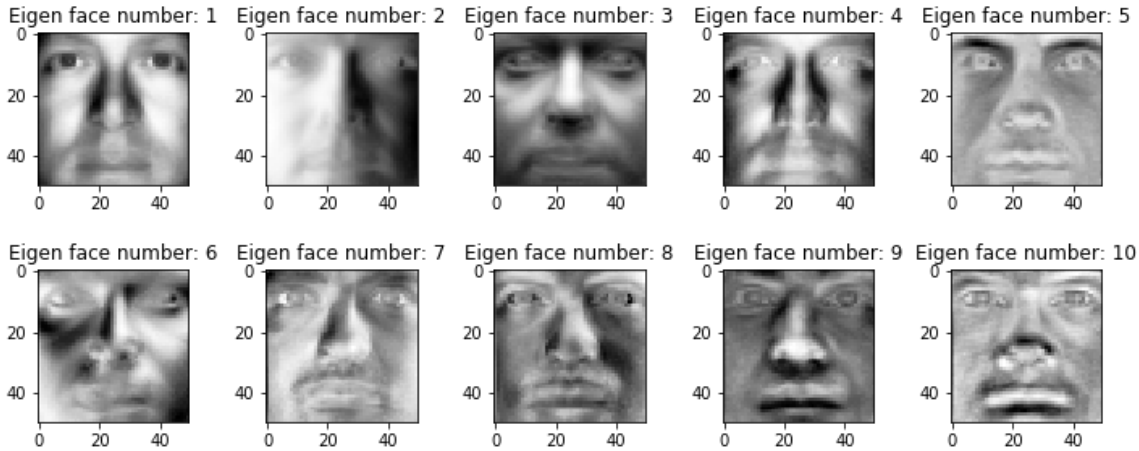
In [7]:

```
# e: SVD decomposition
u, s, vh = np.linalg.svd(train_avg)
print (u.shape, s.shape, vh.shape)
```

```
(540, 540) (540,) (2500, 2500)
```

In [8]:

```
# Plot first 10 eigen face after SVD decomposition
plt.figure(figsize=(10, 6))
for i in range(10):
    plt.subplot(3, 5, i + 1)
    plt.imshow(vh[i,:].reshape(50,50), cmap = cm.Greys_r)
    plt.title('Eigen face number: %d' % (i+1))
plt.tight_layout()
plt.show()
```

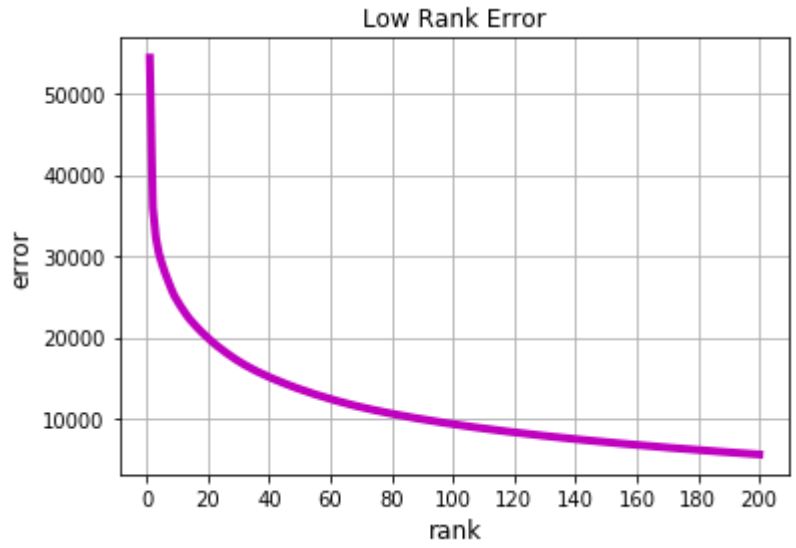


In [9]:

```
# f: Low rank approx
low_rank_error = []
for i in range(1,201):
    Xr = u[:,:i].dot(np.diag(s[:i]).dot(vh[:i,:]))
    low_rank_error.append(np.linalg.norm(Xr-train_avg))
```

In [10]:

```
plt.plot(range(1, 201), low_rank_error, 'm', linewidth=4)
plt.xticks(range(0, 201, 20))
plt.title("Low Rank Error", fontsize=12)
plt.xlabel('rank', fontsize=12)
plt.ylabel('error', fontsize=12)
plt.grid()
```



In [11]:

```
# g: Eigenface feature
def eigenface_feature(V, X, r=1):
    return X.dot(V[:r, :].T)

def get_eigenface_feature(v, train, test, r=1):
    return eigenface_feature(v, train, r), eigenface_feature(v, test, r)
```

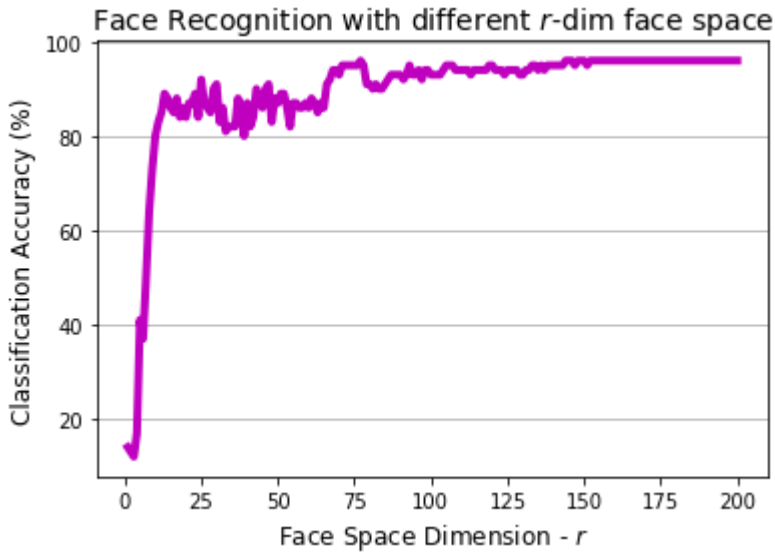
In [12]:

```
# h: Face Recognization
clf = LogisticRegression(solver='lbfgs', random_state=0, multi_class='ovr')
accuracy = []
for r in range(1, 201):
    training, testing = get_eigenface_feature(vh, train_avg, test_avg, r)
    clf.fit(training, train_labels)
    accuracy.append(clf.score(testing, test_labels))

print (len(accuracy))
```

```
200
```

In [13]:

```
plt.plot(range(1,201), np.array(accuracy) * 100, 'm', linewidth=4)
plt.xticks(range(0, 201, 25))
plt.grid(axis='y')
plt.xlabel("Face Space Dimension - $r$", fontsize=12)
plt.ylabel("Classification Accuracy (%)", fontsize=12)
plt.title("Face Recognition with different $r$-dim face space", fontsize=14)
plt.show()
```



In [14]:

```
"""
References:
https://github.com/sidxiong/cornell-cs5785-ml
"""
```

Out[14]:

```
'\nReferences:\nhttps://github.com/sidxiong/cornell-cs5785-ml\n'
```

In [16]:

```
"""Problem 2"""
```

Out[16]:

```
'Problem 2'
```

In [1]:

```
import pandas as pd
import numpy as np

from sklearn.metrics import (
    classification_report,
    confusion_matrix
)
from sklearn.model_selection import (
    train_test_split,
    KFold,
)
from sklearn.naive_bayes import (
    GaussianNB,
    BernoulliNB
)
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
```

In [2]:

```
# a: Download test and train dataset
train = pd.read_json(r'F:\Annie\CornellMS\Semester 4\Machine Learning\Homework\HW2\What
s Cooking\train.json')
test = pd.read_json(r'F:\Annie\CornellMS\Semester 4\Machine Learning\Homework\HW2\Whats
Cooking\test.json')
```

In [3]:

```
train.head()
```

Out[3]:

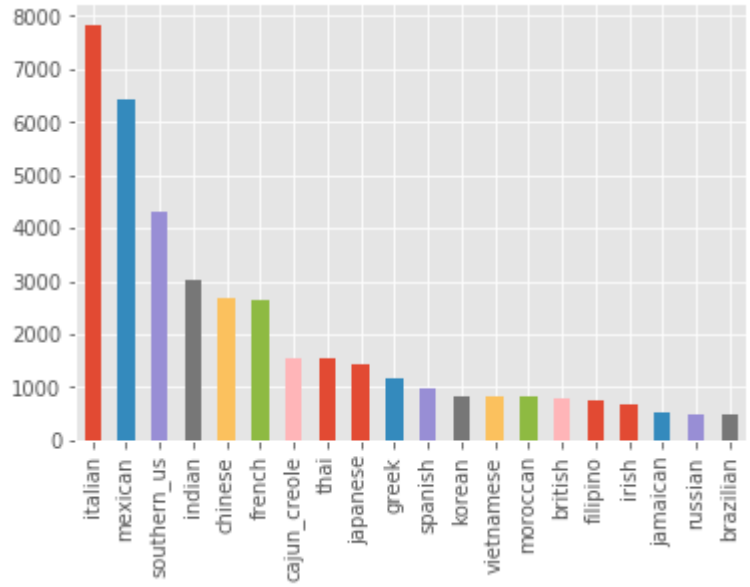
	cuisine	id	ingredients
0	greek	10259	[romaine lettuce, black olives, grape tomatoes...
1	southern_us	25693	[plain flour, ground pepper, salt, tomatoes, g...
2	filipino	20130	[eggs, pepper, salt, mayonaise, cooking oil, g...
3	indian	22213	[water, vegetable oil, wheat, salt]
4	indian	13162	[black pepper, shallots, cornflour, cayenne pe...

In [4]:

```
# b: Visualizing data - Cuisines
plt.style.use('ggplot')
train['cuisine'].value_counts().plot(kind='bar')
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14cd2a8be48>
```



In [5]:

```
# Printing types of cusines in the training data
print ("There are ", len(train), " number of samples of food in training set")
print ("There are ", len(set(train['cuisine'])), " cuisines in training set, including
 ", set(train['cuisine']))
```

```
There are  39774  number of samples of food in training set
There are  20  cuisines in training set, including  {'indian', 'southern_u
s', 'brazilian', 'korean', 'spanish', 'moroccan', 'irish', 'jamaican', 'fr
ench', 'chinese', 'japanese', 'thai', 'filipino', 'cajun_creole', 'vietnam
ese', 'british', 'mexican', 'russian', 'greek', 'italian'}
```

In [6]:

```
# Printing total number of unique ingredients present
ingredients = set([])
cuisines = set(train['cuisine'])

for index, row in train.iterrows():
    ingredients |= set(row['ingredients'])
print ("There are ", len(ingredients), "ingredients in the training set")
```

```
There are  6714 ingredients in the training set
```

In [7]:

```
# c: Represent data in a n*d matrix
cuisines = list(cuisines)
ingredients = list(ingredients)

x_train = np.zeros([len(train), len(ingredients)])
y_train, name_map = pd.factorize(train['cuisine'])

for index, dish in train.iterrows():
    for j in dish['ingredients']:
        x_train[index][ingredients.index(j)] = 1
print (x_train.shape, y_train.shape)
```

```
(39774, 6714) (39774,)
```

In [8]:

```
# d: 3-fold crossvalidation using Bernoullis and Gausian NBC

gaussian_avg_acc = []
bernoulli_avg_acc = []

gaussian = GaussianNB()
bernoulli = BernoulliNB()

kf = KFold(n_splits=3)
kf.get_n_splits(x_train)

i = 1

for train_index, test_index in kf.split(x_train):
    X_train, X_test = x_train[train_index], x_train[test_index]
    Y_train, Y_test = y_train[train_index], y_train[test_index]
    print("GaussianNB iteration ", i)
    gaussian.fit(X_train, Y_train)
    score_gaussian = gaussian.score(X_test, Y_test)
    gaussian_avg_acc.append(score_gaussian)
    print(score_gaussian)
    print("BernoulliNB iteration ", i)
    bernoulli.fit(X_train, Y_train)
    score_bernoulli = bernoulli.score(X_test, Y_test)
    bernoulli_avg_acc.append(score_bernoulli)
    print(score_bernoulli)
    i += 1

print("Average accuracy of Gassian Naive Bayes: ", np.average(gaussian_avg_acc))
print("Average accuracy of Bernoulli Naive Bayes: ", np.average(bernoulli_avg_acc))
```

```
GaussianNB iteration  1
0.37901644290239855
BernoulliNB iteration  1
0.684190677326897
GaussianNB iteration  2
0.3829386031075577
BernoulliNB iteration  2
0.6795142555438226
GaussianNB iteration  3
0.37758334590435966
BernoulliNB iteration  3
0.6869060190073918
Average accuracy of Gassian Naive Bayes:  0.3798461306381053
Average accuracy of Bernoulli Naive Bayes:  0.6835369839593705
```

In [9]:

```
# e: Bernoulli performs better

"""The data is categorical, rather than numeric and is charecterised by presence or abs
ence. The Bernoulli distribution is sometimes used to model a single individual experie
ncing an event. Here, the random variable X which has a Bernoulli distribution can take
value 1 with the probability of presence, say p, and the value 0 with the probability o
f absence, say q or 1-p. As the distribution is Bernoulli, hence it performs better tha
n Gaussian."""
```

Out[9]:

```
'The data is categorical, rather than numeric and is charecterised by pres
ence or absence. The Bernoulli distribution is sometimes used to model a s
ingle individual experiencing an event. Here, the random variable X which
has a Bernoulli distribution can take value 1 with the probability of pres
ence, say p, and the value 0 with the probability of absence, say q or 1-
p. As the distribution is Bernoulli, hence it performs better than Gaussia
n.'
```

In [10]:

```
# f: 3-fold cross-validation using Logistic Regression

logistic_avg_acc = []
logistic = LogisticRegression(solver='lbfgs', multi_class='multinomial', random_state=0
)

i = 1

for train_index, test_index in kf.split(x_train):
    X_train, X_test = x_train[train_index], x_train[test_index]
    Y_train, Y_test = y_train[train_index], y_train[test_index]
    print("Logistic Regression iteration ", i)
    logistic.fit(X_train, Y_train)
    score_logistic = logistic.score(X_test, Y_test)
    logistic_avg_acc.append(score_logistic)
    print(score_logistic)
    i += 1

print("Average accuracy of Logistic Regression: ", np.average(logistic_avg_acc))
```

```
Logistic Regression iteration  1
0.7725147081007694
Logistic Regression iteration  2
0.7704782018403983
Logistic Regression iteration  3
0.7757580328858048
Average accuracy of Logistic Regression:  0.7729169809423242
```

In [11]:

```
# g: Train your best-classifier [Logistic Regression]

# Converting test file to matrix
x_test = np.zeros([len(test), len(ingredients)])

for index, dish in test.iterrows():
    for j in dish['ingredients']:
        if j in ingredients:
            x_test[index][ingredients.index(j)] = 1

print (x_test.shape)
```

```
(9944, 6714)
```

In [12]:

```
# Prediction using Model
logistic.fit(x_train, y_train)
predictions = logistic.predict(x_test)
```

In [13]:

```
final_out = pd.DataFrame(name_map[predictions], columns=['cuisine'])
final_out['id'] =test['id']
final_out = final_out.set_index('id')
final_out.to_csv('F:\Annie\CornellMS\Semester 4\Machine Learning\Homework\HW2\Whats Coo
king\WhatsCooking_Solutions.csv')
```

In [15]:

```
"""
Reference:
https://github.com/wrymax/CS5785-AML"""
```

Out[15]:

```
'\nReference:\nhttps://github.com/wrymax/CS5785-AML'
```

In [17]:

```
"""Accuracy: 77.956%"""
```

Out[17]:

```
'Accuracy: 77.956%'
```

Applied Machine Laerning

Homework 2

Zeyu Wang, Chandrima Bhattacharya

10/9/2019

Question 1

To show

$$max\ a^T Ba \quad s.t.\ a^T Wa = 1$$

According to Lagrange multiplier, we can define

$$L(a, \lambda) = a^T Ba - \lambda(a^T Wa - 1)$$

Take derivative with respect to a, since W and B are covariance matrices which should be symmetric:

$$\frac{\partial L}{\partial a} = 2Ba - 2\lambda Wa$$

Let it equal to 0, we have

$$Ba = \lambda Wa \Rightarrow W^{-1}Ba = \lambda a$$

Which now is a standard eigenvalue problem.

Question 2

2.a

From the log-ratio equation (4.9) on textbook we have

$$log\frac{(G=2|X=x)}{(G=1|X=x)} = log\frac{\pi_2}{\pi_1} - \frac{1}{2}(\mu_2+\mu_1)^T\Sigma^{-1}(\mu_2-\mu_1) + x^T\Sigma^{-1}(\mu_2-\mu_1)$$

Here we have class size N_1, N_2, so we get

$$\pi_1 = \frac{N_1}{N} \qquad \pi_2 = \frac{N_2}{N}$$

Then we have

$$log\frac{(G=2|X=x)}{(G=1|X=x)} = log\frac{N_2}{N_1} - \frac{1}{2}(\hat{\mu}_2+\hat{\mu}_1)^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) + x^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1)$$

When it > 0, the LDA rule classifies it to class 2, that is the same as the equation below:

$$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) > \frac{1}{2}(\hat{\mu}_2+\hat{\mu}_1)^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) - log\frac{N_2}{N_1}$$

2.b

For the least squares criterion

$$\sum_{i=1}^{N}(y_i-\beta_0-x_i^T\beta)^2$$

Take derivative respect to β_0 and β separately, we have

$$\sum_{i=1}^{N}(y_i-\beta_0-x_i^T\beta) = 0 \qquad (1)$$

$$\sum_{i=1}^{N}(y_i-\beta_0-x_i^T\beta)x_i = 0 \qquad (2)$$

From equation (1) we have

$$\left[N_1(-\frac{N}{N_1}) + N_2\frac{N}{N_2}\right] - N\beta_0 - N\hat{\mu}\beta = 0$$

Where

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N}x_i = \frac{N_1\hat{\mu}_1 + N_2\hat{\mu}_2}{N}$$

Then we get

$$\beta_0 = -\hat{\mu}^T\beta$$

And for equation (2), we have

$$\sum_{i=1}^{N}(y_i+\hat{\mu}^T\beta-x_i^T\beta)x_i = 0$$

Expand the equation, we have

$$\left(\sum_{i=1}^{N} x_i x_i^T - N\hat{\mu}\hat{\mu}^T\right)\beta = \sum_{i=1}^{N} y_i x_i \qquad (3)$$

For the left part, since

$$\hat{\Sigma} = \frac{\sum_{k=1}^{2}\sum_{G_i=k}(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{N-2}$$

We have

$$\begin{aligned}
\hat{\Sigma} &= \frac{\sum_{i=1}^{N_1}(x_i - \hat{\mu}_1)(x_i - \hat{\mu}_1)^T + \sum_{i=1}^{N_2}(x_i - \hat{\mu}_2)(x_i - \hat{\mu}_2)^T}{N-2} \\
&= \frac{\sum_{i=1}^{N_1}(x_i x_i^T - \hat{\mu}_1 x_i^T - x_i\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_1^T) + \sum_{i=1}^{N_2}(x_i x_i^T - \hat{\mu}_2 x_i^T - x_i\hat{\mu}_2^T + \hat{\mu}_2\hat{\mu}_2^T)}{N-2} \\
&= \frac{(\sum_{i=1}^{N_1} x_i x_i^T - N_1\hat{\mu}_1\hat{\mu}_1^T) + (\sum_{i=1}^{N_2} x_i x_i^T - N_2\hat{\mu}_2\hat{\mu}_2^T)}{N-2} \\
&= \frac{\sum_{i=1}^{N} x_i x_i^T - N_1\hat{\mu}_1\hat{\mu}_1^T - N_2\hat{\mu}_2\hat{\mu}_2^T}{N-2}
\end{aligned}$$

Then equation (3) becomes

$$\left[(N-2)\hat{\Sigma} + N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T - N\hat{\mu}\hat{\mu}^T\right]\beta = \sum_{i=1}^{N} y_i x_i \qquad (4)$$

For $N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T - N\hat{\mu}\hat{\mu}^T$, we have

$$\begin{aligned}
N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T - N\hat{\mu}\hat{\mu}^T &= N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T - N\frac{N_1\hat{\mu}_1 + N_2\hat{\mu}_2}{N}\frac{N_1\hat{\mu}_1^T + N_2\hat{\mu}_2^T}{N} \\
&= \frac{N_1 N_2}{N}\left[\frac{N}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \frac{N}{N_1}\hat{\mu}_2\hat{\mu}_2^T - \left(\frac{N_1}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_2^T + \hat{\mu}_2\hat{\mu}_1^T + \frac{N_2}{N_1}\hat{\mu}_2\hat{\mu}_2^T\right)\right] \\
&= \frac{N_1 N_2}{N}\left(\hat{\mu}_1\hat{\mu}_1^T + \hat{\mu}_2\hat{\mu}_2^T - \hat{\mu}_1\hat{\mu}_2^T - \hat{\mu}_2\hat{\mu}_1^T\right) \\
&= \frac{N_1 N_2}{N}\left(\hat{\mu}_2 - \hat{\mu}_1\right)\left(\hat{\mu}_2 - \hat{\mu}_1\right)^T
\end{aligned}$$

Because

$$\hat{\Sigma}_B = \frac{N_1 N_2}{N^2}\left(\hat{\mu}_2 - \hat{\mu}_1\right)\left(\hat{\mu}_2 - \hat{\mu}_1\right)^T$$

We have

$$N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T - N\hat{\mu}\hat{\mu}^T = N\hat{\Sigma}_B$$

So the left part becomes

$$\left[(N-2)\hat{\Sigma}+N\hat{\Sigma}_B\right]\beta$$

For the right part, we have

$$\begin{aligned}\sum_{i=1}^{N} y_i x_i &= \sum_{i=1}^{N_1} y_i x_i + \sum_{j=1}^{N_2} y_i x_i \\ &= -\frac{N}{N_1} N_1 \hat{\mu}_1 + \frac{N}{N_2} N_2 \hat{\mu}_2 \\ &= N(\hat{\mu}_2 - \hat{\mu}_1)\end{aligned}$$

As a result, we have

$$\left[(N-2)\hat{\Sigma}+N\hat{\Sigma}_B\right]\beta = N(\hat{\mu}_2 - \hat{\mu}_1)$$

2.c

According to 2.b, we have

$$\begin{aligned}\hat{\Sigma}_B\beta &= \frac{N_1 N_2}{N^2}\left(\hat{\mu}_2 - \hat{\mu}_1\right)\left(\hat{\mu}_2 - \hat{\mu}_1\right)^T \beta \\ &= C\left(\hat{\mu}_2 - \hat{\mu}_1\right)\end{aligned}$$

where C is a constant, and

$$C = \frac{N_1 N_2}{N^2}\left(\hat{\mu}_2 - \hat{\mu}_1\right)^T \beta$$

hence $\hat{\Sigma}_B$ is in the direction $(\hat{\mu}_2 - \hat{\mu}_1)$

Thus we have

$$(N-2)\hat{\Sigma}\hat{\beta} + NC\left(\hat{\mu}_2 - \hat{\mu}_1\right) = N(\hat{\mu}_2 - \hat{\mu}_1) \Rightarrow \quad \hat{\beta} = \frac{N(1-C)}{N-2}\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

As a result

$$\hat{\beta} \propto \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

Therefore the least-squares regression coefficient is identical to the LDA coefficient, up to a scalar multiple.

2.d

Let the codes be C_1 and C_2, where $C_1 \neq C_2$

According to equation (1) in 2.b, now we have

$$N_1C_1 + N_2C_2 - N\beta_0 - N\hat{\mu}\beta = 0\beta_0 = \frac{N_1C_1 + N_2C_2}{N} - \hat{\mu}\beta$$

Similarly for equation (3) in 2.b, we now have

$$\left(\sum_{i=1}^{N} x_i x_i^T - N\hat{\mu}\hat{\mu}^T\right)\beta = \sum_{i=1}^{N} y_i x_i - \frac{N_1C_1 + N_2C_2}{N} N\hat{\mu}$$

Hence we have

$$\begin{aligned}\left[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B\right]\beta &= C_1N_1\hat{\mu}_1 + C_2N_2\hat{\mu}_2 - \frac{N_1C_1 + N_2C_2}{N}(N_1\hat{\mu}_1 + N_2\hat{\mu}_2) \\ &= \hat{\mu}_1\left(C_1N_1 - N_1\frac{N_1C_1 + N_2C_2}{N}\right) + \hat{\mu}_2\left(C_2N_2 - N_2\frac{N_1C_1 + N_2C_2}{N}\right) \\ &= \frac{N_1N_2}{N}(C_2 - C_1)\left(\hat{\mu}_2 - \hat{\mu}_1\right)\end{aligned}$$

As a result, we have

$$\left[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B\right]\beta = \frac{N_1N_2}{N}(C_2 - C_1)\left(\hat{\mu}_2 - \hat{\mu}_1\right)$$

Difference from 2.b result only on scalar part, which has no effect on the direction of $\hat{\Sigma}_B$

Similarly we still have $\hat{\beta} \propto \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$

2.e

According to 2.b, we have

$$\hat{\beta}_0 = -\hat{\mu}^T\hat{\beta}$$

Thus

$$\begin{aligned}\hat{f}(x) &= (x - \hat{\mu})^T\hat{\beta} \\ &= m(x - \hat{\mu})^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)\end{aligned}$$

where

$$m = \frac{N(1-C)}{N-2}$$

According to 2.a, for LDA, when $N_1 = N_2$, we have

$$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) > \frac{1}{2}(\hat{\mu}_2+\hat{\mu}_1)^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) \Rightarrow x^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) > \hat{\mu}^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) \Rightarrow (x-\hat{\mu})^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) > 0$$

Since

$$m = \frac{N(1-C)}{N-2}$$

And

$$\begin{aligned} C &= \frac{N_1N_2}{N^2}\left(\hat{\mu}_2-\hat{\mu}_1\right)^T\beta \\ &= \frac{N_1N_2}{N^2}\left(\hat{\mu}_2-\hat{\mu}_1\right)^T\frac{N(\hat{\mu}_2-\hat{\mu}_1)}{(N-2)\hat{\Sigma}+N\hat{\Sigma}_B} \\ &= \frac{\frac{N_1N_2}{N}(\hat{\mu}_2-\hat{\mu}_1)\left(\hat{\mu}_2-\hat{\mu}_1\right)^T}{(N-2)\hat{\Sigma}+\frac{N_1N_2}{N}\left(\hat{\mu}_2-\hat{\mu}_1\right)\left(\hat{\mu}_2-\hat{\mu}_1\right)^T} < 1 \end{aligned}$$

So we have m > 0, and in this case, LDA is the same with $\hat{f}(x) > 0$

And when $N_1 \neq N_2$, $\hat{f}(x)$ is not the same as LDA rule.

Question 3

3.a

Two matrices we get are:

M^TM **:**

```
##      [,1] [,2] [,3]
## [1,]   39   57   60
## [2,]   57  118   53
## [3,]   60   53  127
```

MM^T **:**

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   10    9   26    3   26
## [2,]    9   62    8   -5   85
## [3,]   26    8   72   10   50
## [4,]    3   -5   10    2   -1
## [5,]   26   85   50   -1  138
```

3.b & 3.c

Eigenvalues of M^TM:

```
## [1] 214.67049  69.32951   0.00000
```

Eigenvectors of M^TM:

```
## [1] -0.4261513 -0.6150088 -0.6634450
```

```
## [1] -0.01460404 -0.72859799  0.68478587
```

```
## [1]  0.9045340 -0.3015113 -0.3015113
```

Eigenvalues of MM^T:

```
## [1] 214.67049  69.32951   0.00000   0.00000   0.00000
```

Eigenvectors of MM^T:

```
## [1] -0.164929423 -0.471647316 -0.336470547 -0.003305851 -0.798200311
```

```
## [1] -0.2449732  0.4533064 -0.8294396 -0.1697466  0.1331066
```

```
## [1]  0.95539856  0.03481209 -0.27076072 -0.04409532 -0.10366268
```

```
## [1]  0.0000000  0.7329548  0.3368074 -0.1388977 -0.5744958
```

```
## [1]  0.0000000 -0.1833779  0.1098493 -0.9746477  0.0660870
```

3.d

SVD of matrix M:

Matrx U(first two eigenvectors of MM^T):

```
##              [,1]       [,2]
## [1,] -0.164929423 -0.2449732
## [2,] -0.471647316  0.4533064
## [3,] -0.336470547 -0.8294396
## [4,] -0.003305851 -0.1697466
## [5,] -0.798200311  0.1331066
```

Matrix V(first two eigenvectors of M^TM):

```
##            [,1]        [,2]
## [1,] -0.4261513  0.01460404
## [2,] -0.6150088  0.72859799
## [3,] -0.6634450 -0.68478587
```

Matrix D(two eigenvalues on diagonal):

```
##          [,1]     [,2]
## [1,] 14.65164 0.000000
## [2,]  0.00000 8.326434
```

3.e

Approximation is $M' = UD'V^T$, result is

```
##            [,1]       [,2]      [,3]
## [1,] 1.02978864 1.48616035 1.6032056
## [2,] 2.94487812 4.24996055 4.5846738
## [3,] 2.10085952 3.03189800 3.2706806
## [4,] 0.02064112 0.02978864 0.0321347
## [5,] 4.98381430 7.19249261 7.7589503
```

Reference:

```
## [1] "http://members.cbio.mines-paristech.fr/~jvert/svn/tutorials/course/1102Senegal/solution.pdf"
```

```
## [1] "http://www2.stat.duke.edu/~banks/cam-lectures.dir/Examples2-solns.pdf"
```