Homework 2

CS 5785 Applied Machine Learning

Xialin Shen <xs293@cornell.edu>

An Le <aql6@cornell.edu>

26th September, 2017

PART A - PROGRAMMING EXERCISE

Question 1 - Eigenface for face recognition

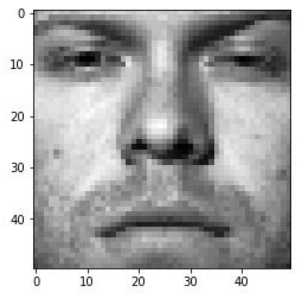
(a)(b) Download and load the training, testing dataset

<CODE>

Load training data set

```
train_labels, train_data = [], []
for line in open('./faces/train.txt'):
    im = misc.imread(line.strip().split()[0])
    train_data.append(im.reshape(2500,))
    train_labels.append(line.strip().split()[1])
train_data, train_labels = np.array(train_data, dtype=float), np.array(train_labels, dtype=int)

print (train_data.shape, train_labels.shape)
plt.imshow(train_data[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

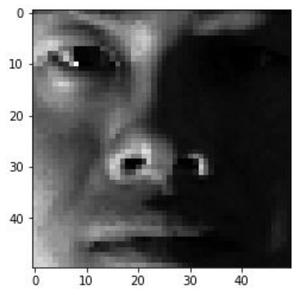
```
(540, 2500) (540,)
```



Load testing data set

```
test_labels, test_data = [], []
for line in open('./faces/test.txt'):
    im = misc.imread(line.strip().split()[0])
    test_data.append(im.reshape(2500,))
    test_labels.append(line.strip().split()[1])
test_data, test_labels = np.array(test_data, dtype=float), np.array(test_labels, dtype=int)

print (test_data.shape, test_labels.shape)
plt.imshow(test_data[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

```
(100, 2500) (100,)
```

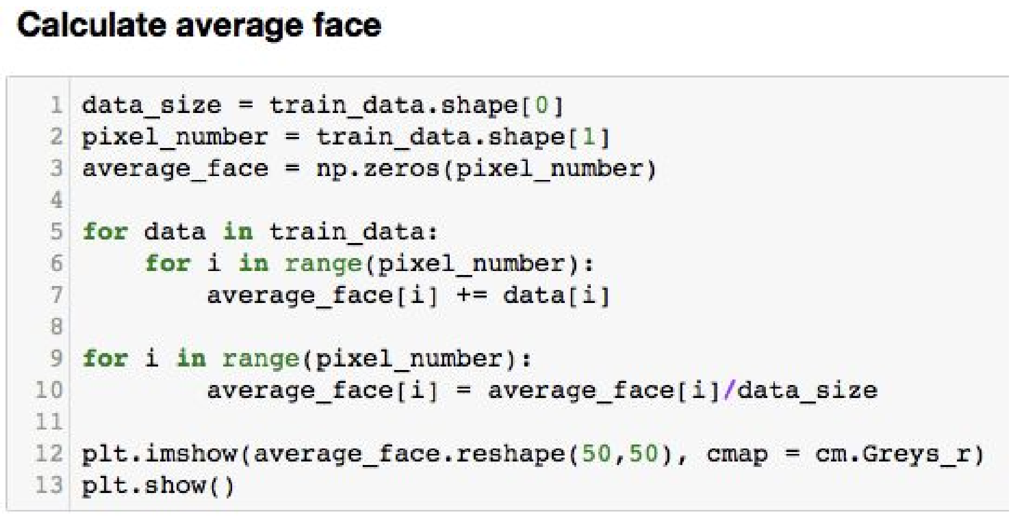


(c) Calculate Average Face

ANSWER:

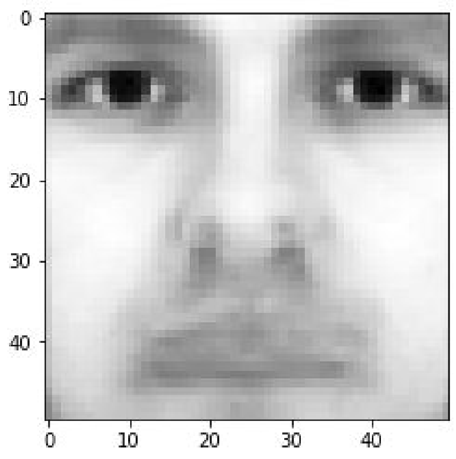
To compute the average face, we sum up every column in X and divide by the number of faces.

<CODE>



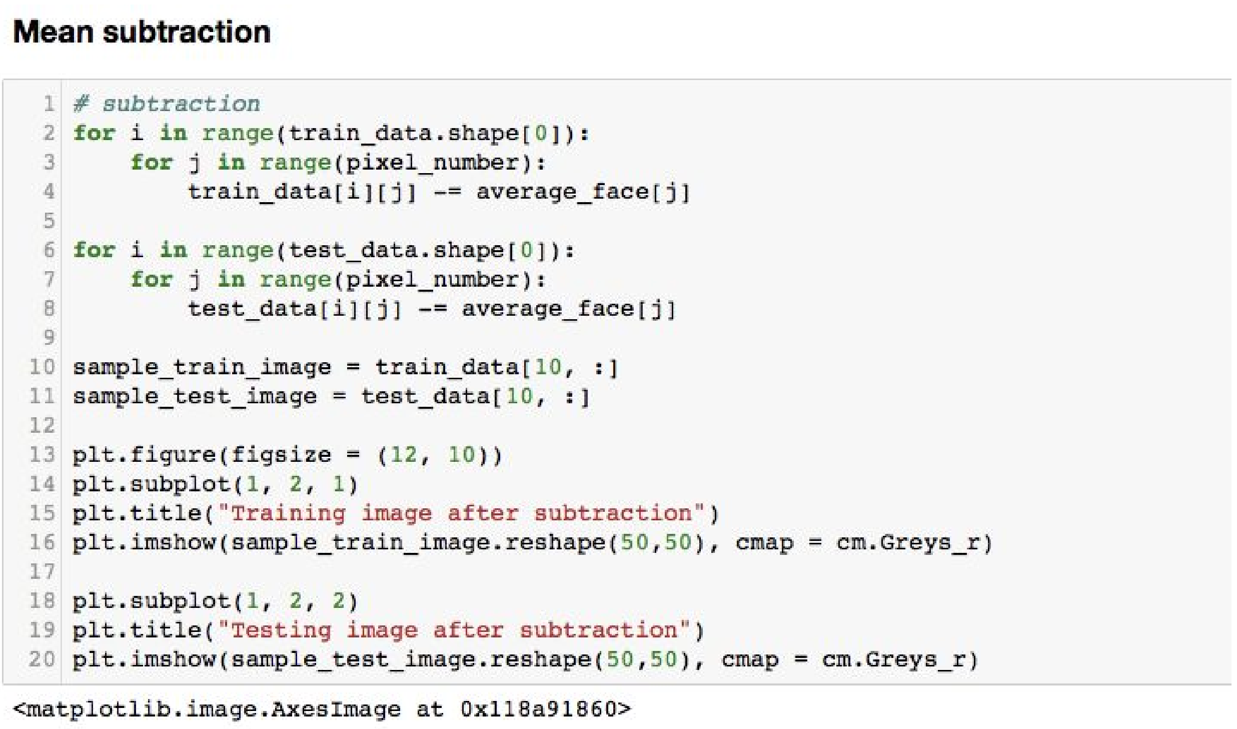
Calculate average face

```python
data_size = train_data.shape[0]
pixel_number = train_data.shape[1]
average_face = np.zeros(pixel_number)

for data in train_data:
    for i in range(pixel_number):
        average_face[i] += data[i]

for i in range(pixel_number):
        average_face[i] = average_face[i]/data_size

plt.imshow(average_face.reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



(d) Mean Subtraction

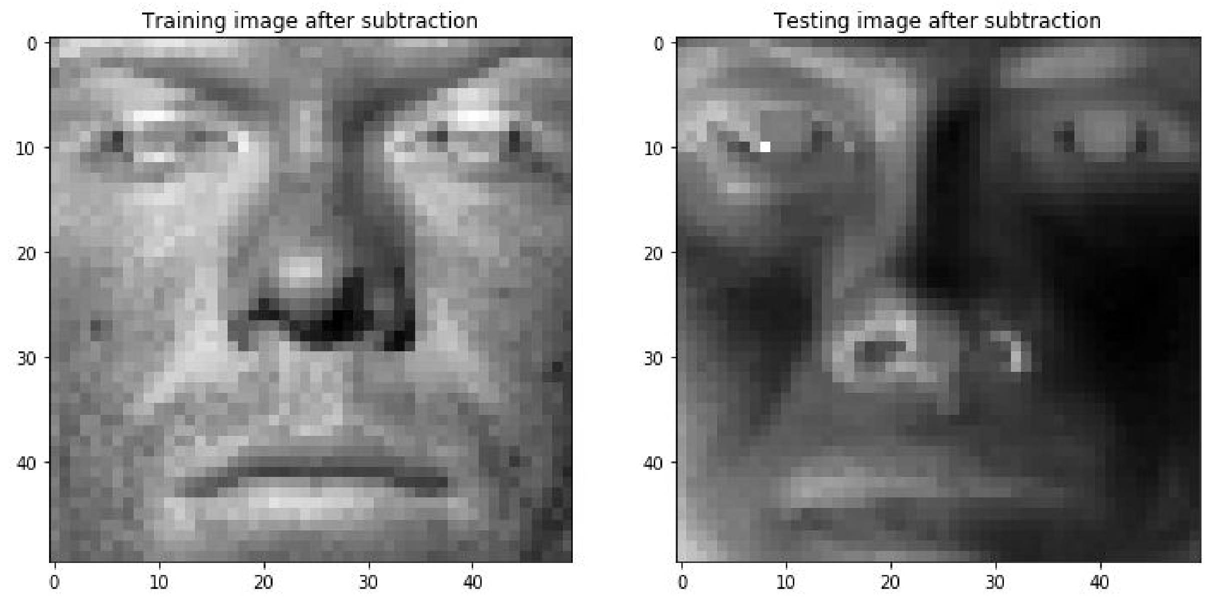
<CODE>



Mean subtraction

```python
# subtraction
for i in range(train_data.shape[0]):
    for j in range(pixel_number):
        train_data[i][j] -= average_face[j]

for i in range(test_data.shape[0]):
    for j in range(pixel_number):
        test_data[i][j] -= average_face[j]

sample_train_image = train_data[10, :]
sample_test_image = test_data[10, :]

plt.figure(figsize = (12, 10))
plt.subplot(1, 2, 1)
plt.title("Training image after subtraction")
plt.imshow(sample_train_image.reshape(50,50), cmap = cm.Greys_r)

plt.subplot(1, 2, 2)
plt.title("Testing image after subtraction")
plt.imshow(sample_test_image.reshape(50,50), cmap = cm.Greys_r)
```

```
<matplotlib.image.AxesImage at 0x118a91860>
```

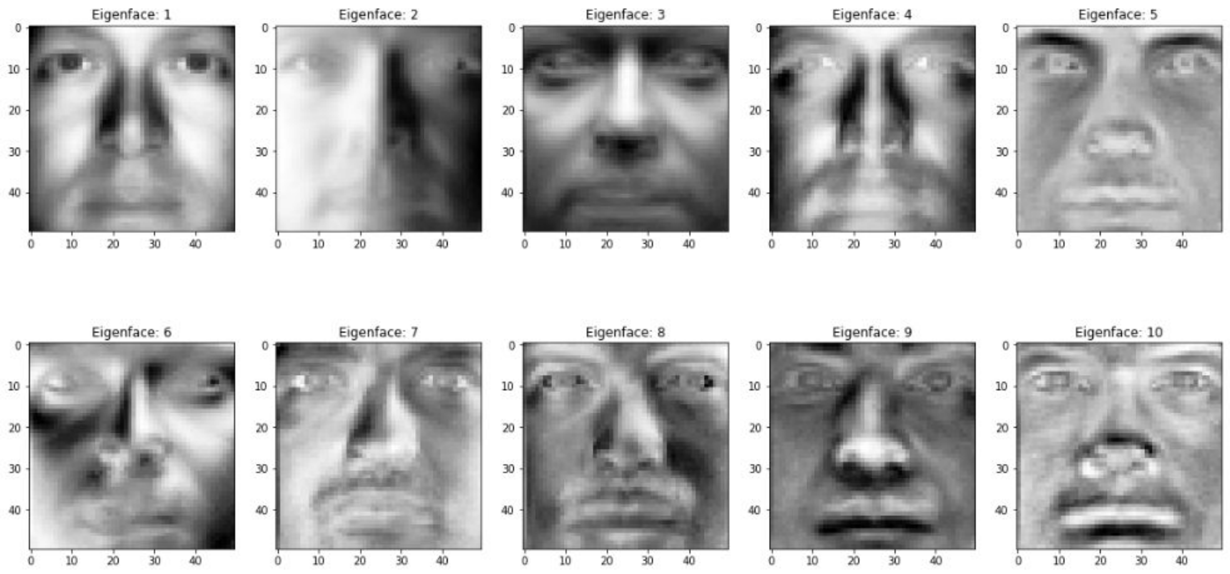

Training image after subtraction
Testing image after subtraction
0
10
20
30
40

(e) Eigenface

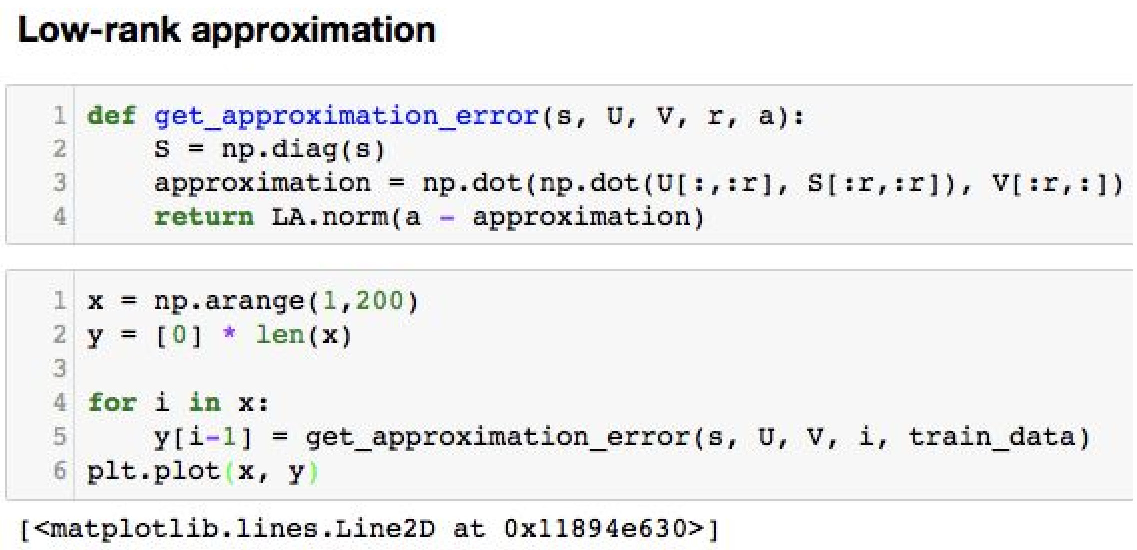
Eigenface

```
u, d, v = np.linalg.svd(train_data)
```

```
plt.figure(figsize=(20,10))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(v[i].reshape(50,50), cmap = cm.Greys_r)
    plt.title("Eigenface: " + str(i+1))
```



(f) Low-rank Approximation

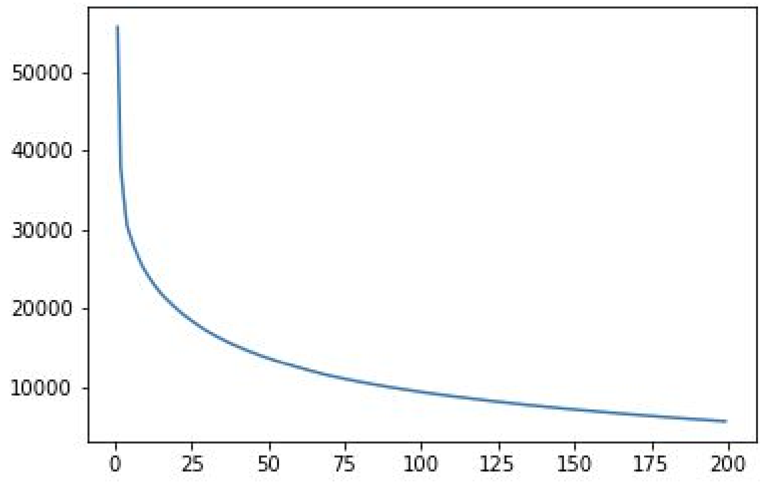


Low-rank approximation

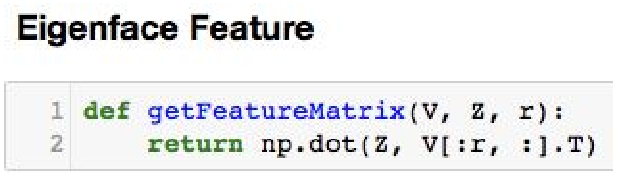
```python
def get_approximation_error(s, U, V, r, a):
    S = np.diag(s)
    approximation = np.dot(np.dot(U[:,:r], S[:r,:r]), V[:r,:])
    return LA.norm(a - approximation)
```

```python
x = np.arange(1,200)
y = [0] * len(x)

for i in x:
    y[i-1] = get_approximation_error(s, U, V, i, train_data)
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x11894e630>]
```



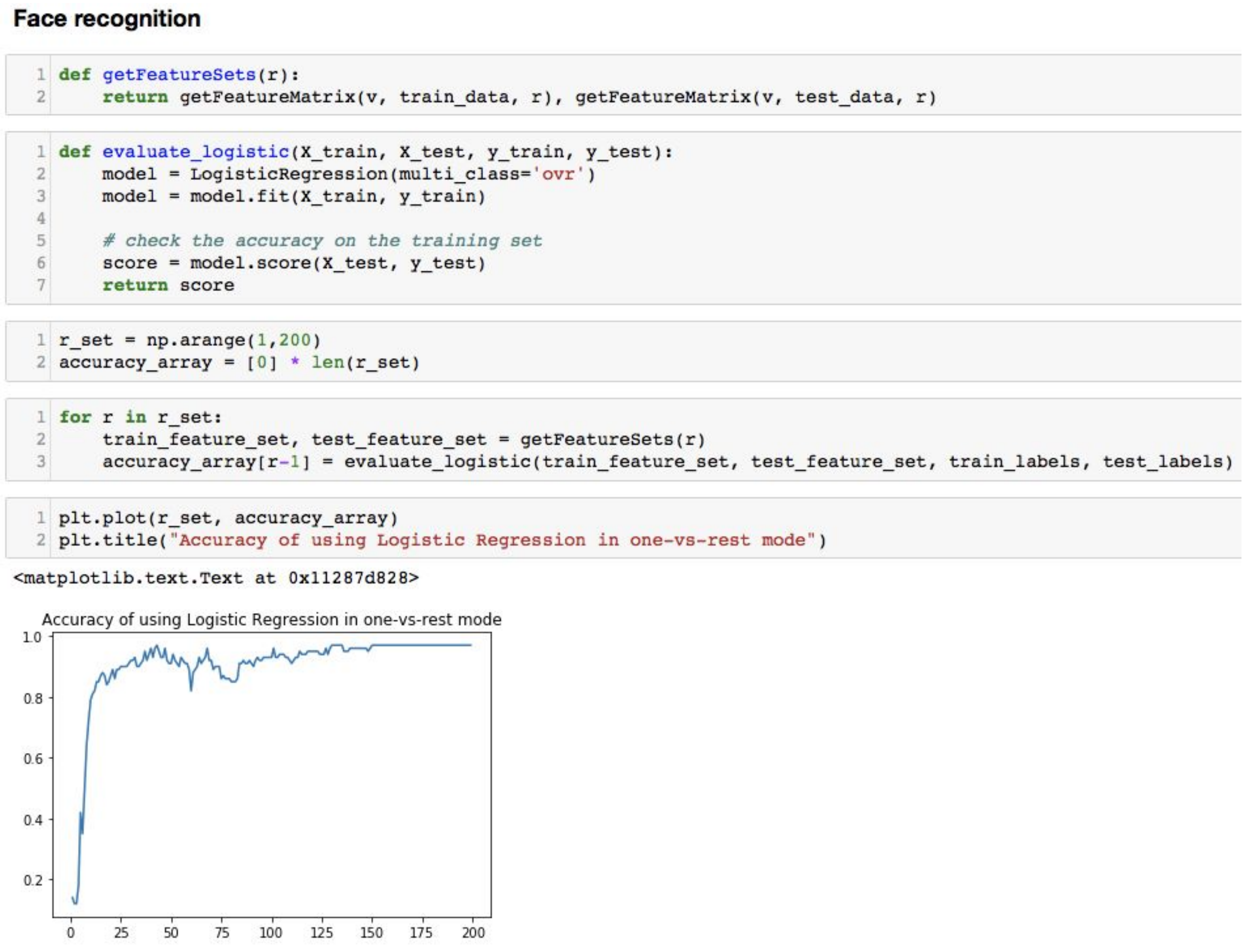
(g) Eigenface Feature



Eigenface Feature

```python
def getFeatureMatrix(V, Z, r):
    return np.dot(Z, V[:r, :].T)
```

(h) Face Recognition



Face recognition

```python
def getFeatureSets(r):
    return getFeatureMatrix(v, train_data, r), getFeatureMatrix(v, test_data, r)
```

```python
def evaluate_logistic(X_train, X_test, y_train, y_test):
    model = LogisticRegression(multi_class='ovr')
    model = model.fit(X_train, y_train)

    # check the accuracy on the training set
    score = model.score(X_test, y_test)
    return score
```

```python
r_set = np.arange(1,200)
accuracy_array = [0] * len(r_set)
```

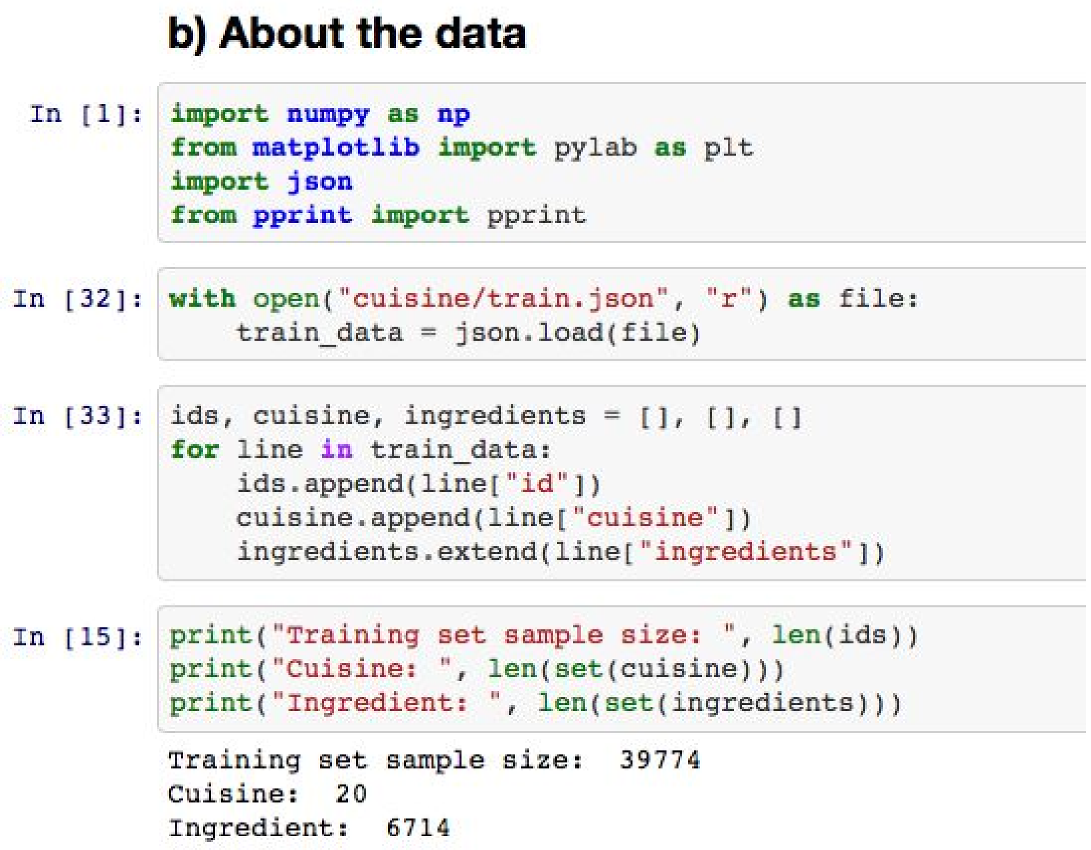
```python
for r in r_set:
    train_feature_set, test_feature_set = getFeatureSets(r)
    accuracy_array[r-1] = evaluate_logistic(train_feature_set, test_feature_set, train_labels, test_labels)
```

```python
plt.plot(r_set, accuracy_array)
plt.title("Accuracy of using Logistic Regression in one-vs-rest mode")
```

```
<matplotlib.text.Text at 0x11287d828>
```

Question 2 - What's Cooking?

(b) *Tell us about the data. How many samples (dishes) are there in the training set? How many categories (types of cuisine)? Use a list to keep all the unique ingredients appearing in the training set. How many unique ingredients are there?*



b) About the data

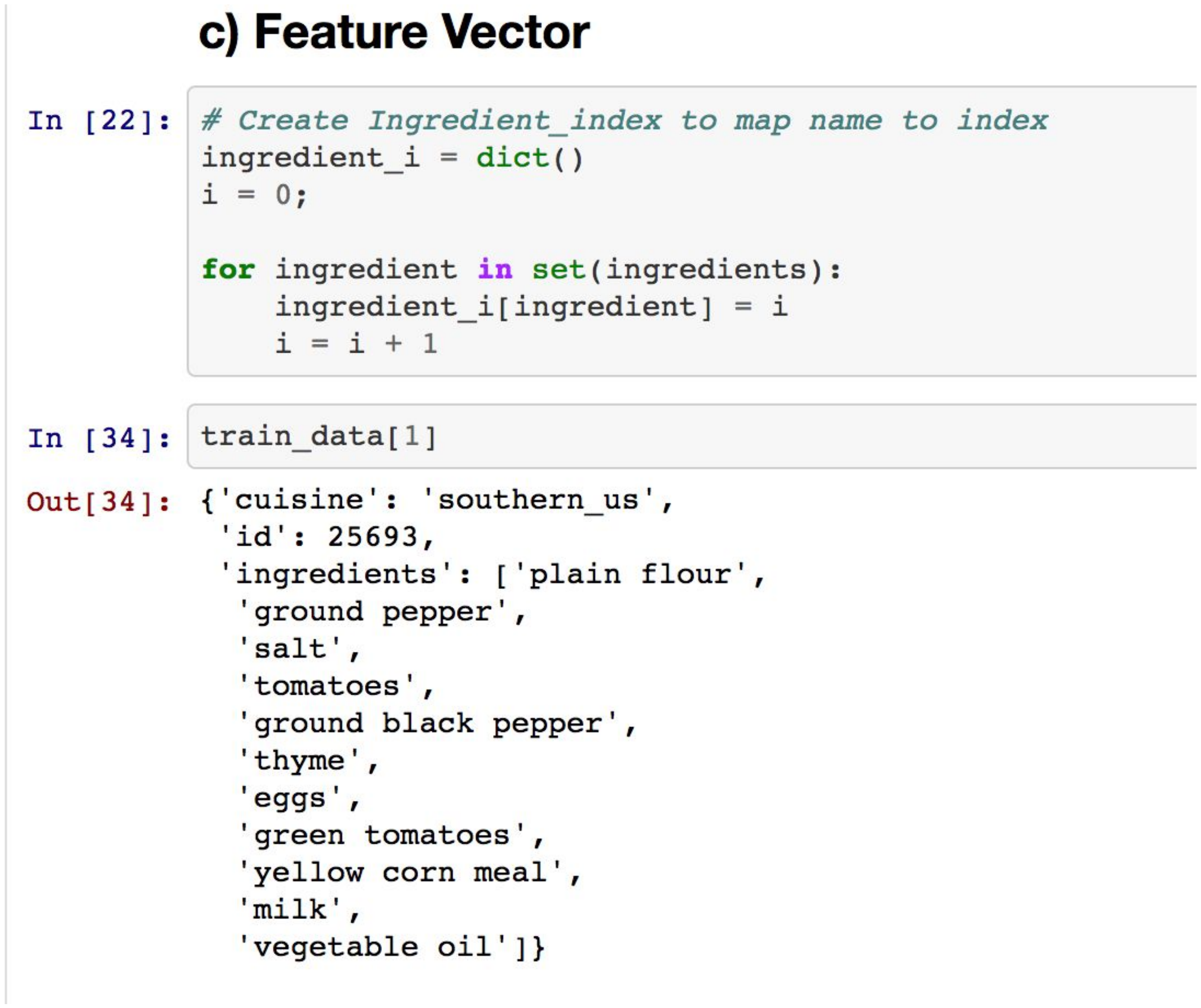
```
In [1]: import numpy as np
        from matplotlib import pylab as plt
        import json
        from pprint import pprint
```

```
In [32]: with open("cuisine/train.json", "r") as file:
             train_data = json.load(file)
```

```
In [33]: ids, cuisine, ingredients = [], [], []
         for line in train_data:
             ids.append(line["id"])
             cuisine.append(line["cuisine"])
             ingredients.extend(line["ingredients"])
```

```
In [15]: print("Training set sample size: ", len(ids))
         print("Cuisine: ", len(set(cuisine)))
         print("Ingredient: ", len(set(ingredients)))

         Training set sample size:  39774
         Cuisine:  20
         Ingredient:  6714
```

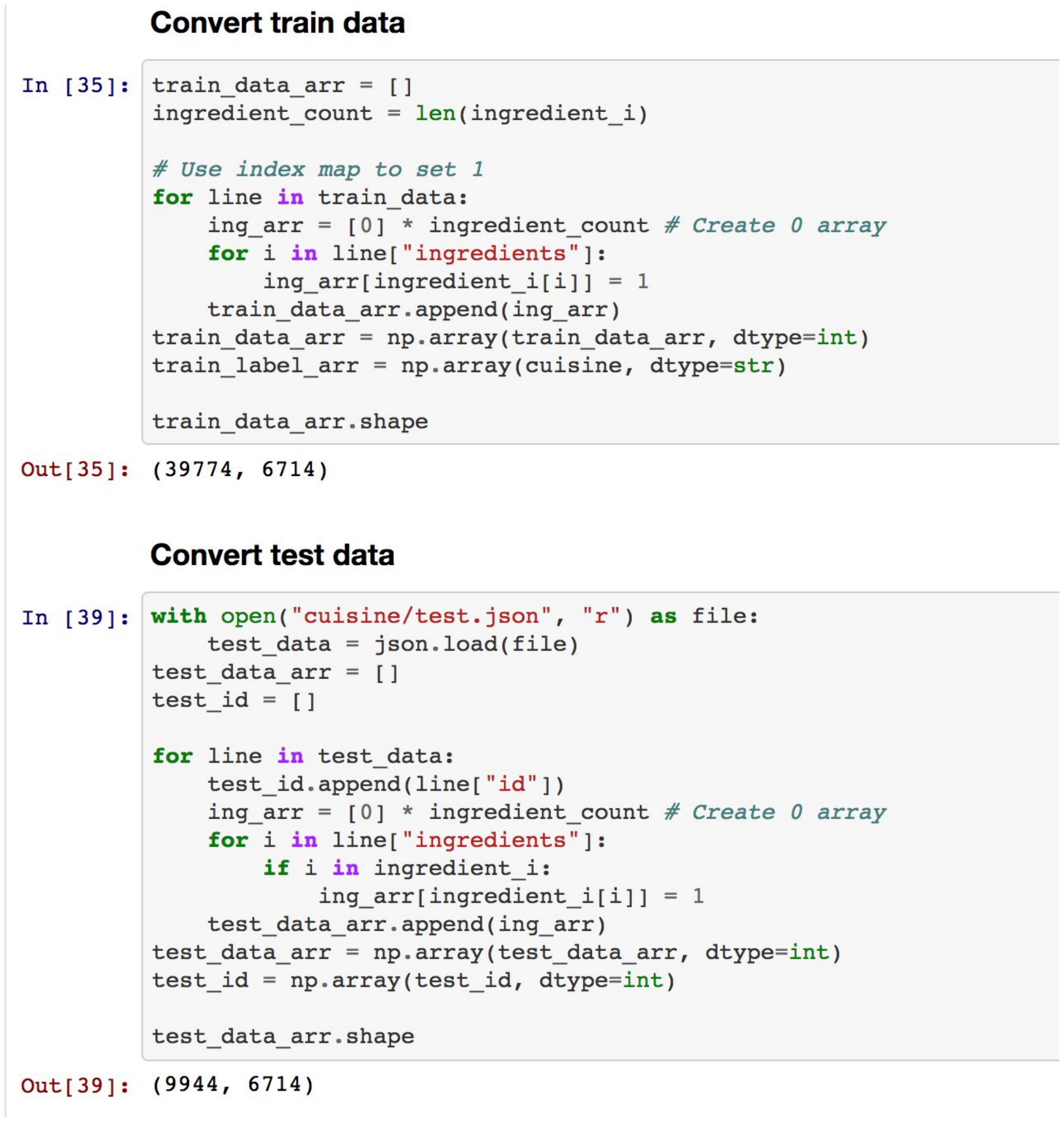
(c) *Represent each dish by a binary ingredient feature vector. Suppose there are d different ingredients in total from the training set, represent each dish by a 1×d binary ingredient vector x, where xi = 1 if the dish contains ingredient i and xi = 0 otherwise. For example, suppose all the ingredients we have in the training set are { beef, chicken, egg, lettuce, tomato, rice } and the dish is made by ingredients { chicken, lettuce, tomato, rice }, then the dish could be represented by a 6×1 binary vector [0, 1, 0, 1, 1, 1] as its feature or attribute. Use n ×d feature matrix to represent all the dishes in training set and test set, where n is the number of dishes.*



c) Feature Vector

```
In [22]: # Create Ingredient_index to map name to index
         ingredient_i = dict()
         i = 0;

         for ingredient in set(ingredients):
             ingredient_i[ingredient] = i
             i = i + 1
```

```
In [34]: train_data[1]

Out[34]: {'cuisine': 'southern_us',
          'id': 25693,
          'ingredients': ['plain flour',
           'ground pepper',
           'salt',
           'tomatoes',
           'ground black pepper',
           'thyme',
           'eggs',
           'green tomatoes',
           'yellow corn meal',
           'milk',
           'vegetable oil']}
```



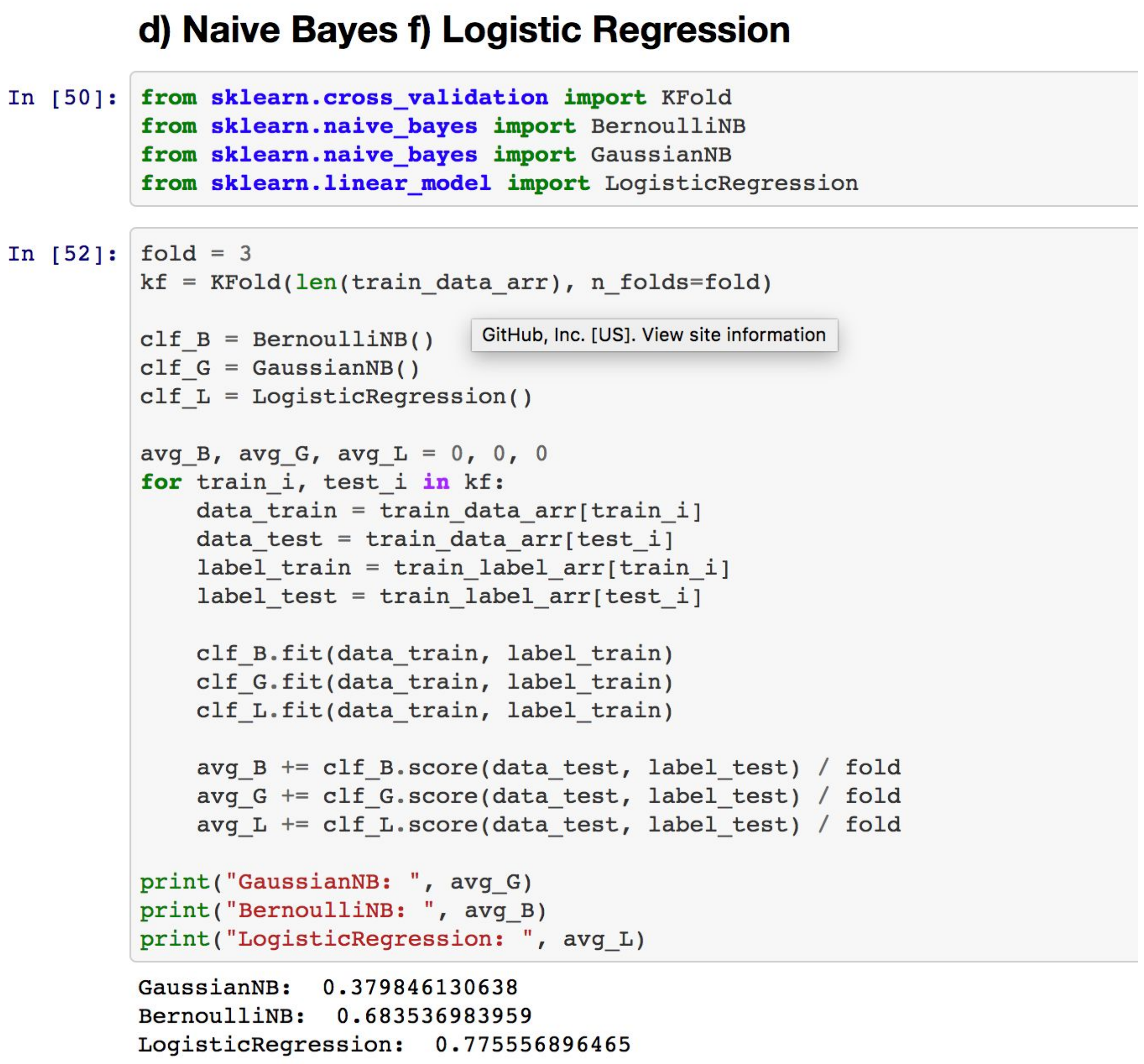
Convert train data

```
In [35]: train_data_arr = []
         ingredient_count = len(ingredient_i)

         # Use index map to set 1
         for line in train_data:
             ing_arr = [0] * ingredient_count # Create 0 array
             for i in line["ingredients"]:
                 ing_arr[ingredient_i[i]] = 1
             train_data_arr.append(ing_arr)
         train_data_arr = np.array(train_data_arr, dtype=int)
         train_label_arr = np.array(cuisine, dtype=str)

         train_data_arr.shape

Out[35]: (39774, 6714)
```

Convert test data

```
In [39]: with open("cuisine/test.json", "r") as file:
             test_data = json.load(file)
         test_data_arr = []
         test_id = []

         for line in test_data:
             test_id.append(line["id"])
             ing_arr = [0] * ingredient_count # Create 0 array
             for i in line["ingredients"]:
                 if i in ingredient_i:
                     ing_arr[ingredient_i[i]] = 1
             test_data_arr.append(ing_arr)
         test_data_arr = np.array(test_data_arr, dtype=int)
         test_id = np.array(test_id, dtype=int)

         test_data_arr.shape

Out[39]: (9944, 6714)
```

(d) *Using Naïve Bayes Classifier to perform 3 fold cross-validation on the training set and report your average classification accuracy. Try both Gaussian distribution prior assumption and Bernoulli distribution prior assumption.*



d) Naive Bayes f) Logistic Regression

In [50]:
```
from sklearn.cross_validation import KFold
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
```

In [52]:
```
fold = 3
kf = KFold(len(train_data_arr), n_folds=fold)

clf_B = BernoulliNB()
clf_G = GaussianNB()
clf_L = LogisticRegression()

avg_B, avg_G, avg_L = 0, 0, 0
for train_i, test_i in kf:
    data_train = train_data_arr[train_i]
    data_test = train_data_arr[test_i]
    label_train = train_label_arr[train_i]
    label_test = train_label_arr[test_i]

    clf_B.fit(data_train, label_train)
    clf_G.fit(data_train, label_train)
    clf_L.fit(data_train, label_train)

    avg_B += clf_B.score(data_test, label_test) / fold
    avg_G += clf_G.score(data_test, label_test) / fold
    avg_L += clf_L.score(data_test, label_test) / fold

print("GaussianNB: ", avg_G)
print("BernoulliNB: ", avg_B)
print("LogisticRegression: ", avg_L)
```

```
GaussianNB:  0.379846130638
BernoulliNB:  0.683536983959
LogisticRegression:  0.775556896465
```

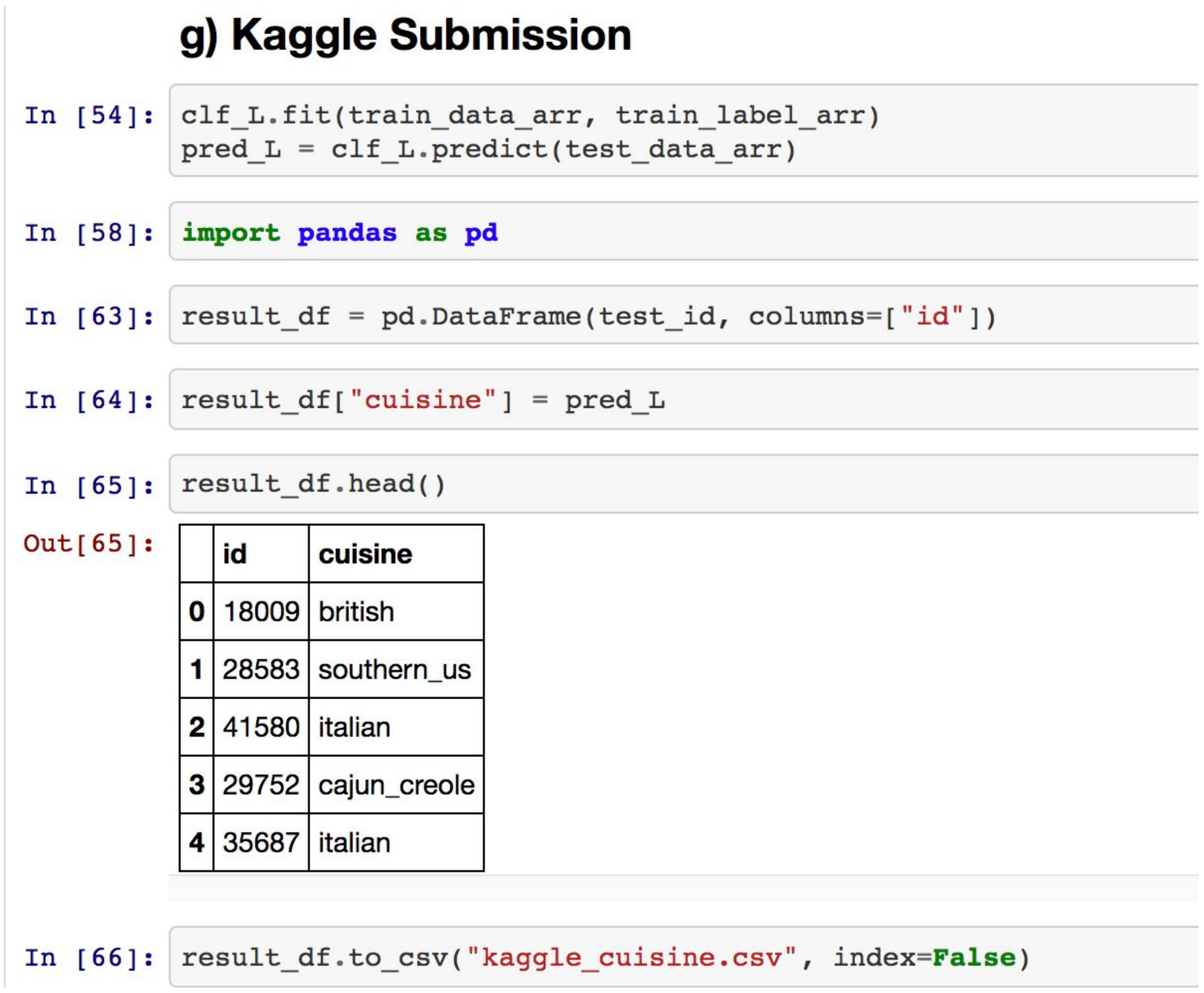
(e) *For Gaussian prior and Bernoulli prior, which performs better in terms of cross-validation accuracy? Why? Please give specific arguments.*

e) BernoulliNB scores 0.68 meanwhile GaussianNB scores 0.38. BernoulliNB performs better since the data (ingredients) describes whether an ingredient is present or not.

(f) *Using Logistic Regression Model to perform 3 fold cross-validation on the training set and report your average classification accuracy.*

See (d)

(g) *Train your best-performed classifier with all of the training data, and generate test labels on test set. Submit your results to Kaggle and report the accuracy.*



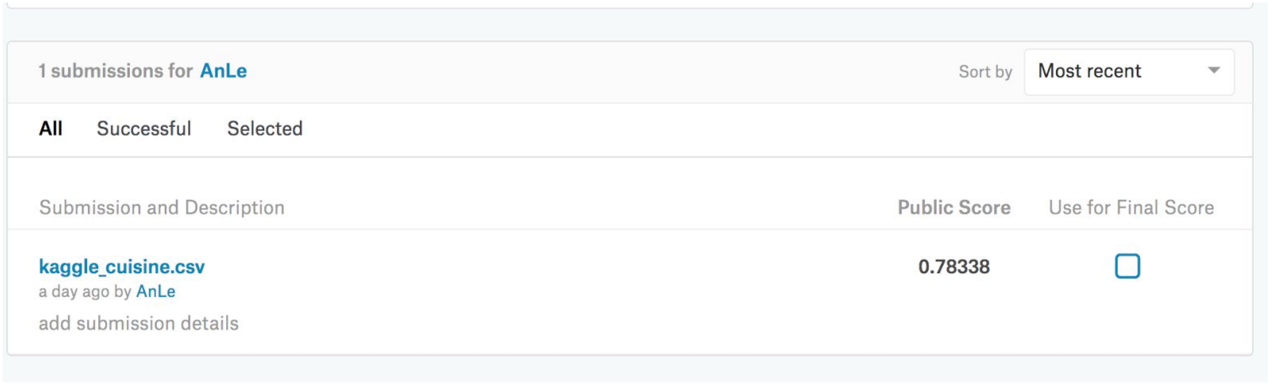
g) Kaggle Submission

```
In [54]: clf_L.fit(train_data_arr, train_label_arr)
         pred_L = clf_L.predict(test_data_arr)

In [58]: import pandas as pd

In [63]: result_df = pd.DataFrame(test_id, columns=["id"])

In [64]: result_df["cuisine"] = pred_L

In [65]: result_df.head()
```

Out[65]:

	id	cuisine
0	18009	british
1	28583	southern_us
2	41580	italian
3	29752	cajun_creole
4	35687	italian

```
In [66]: result_df.to_csv("kaggle_cuisine.csv", index=False)
```



1 submissions for AnLe

Sort by Most recent

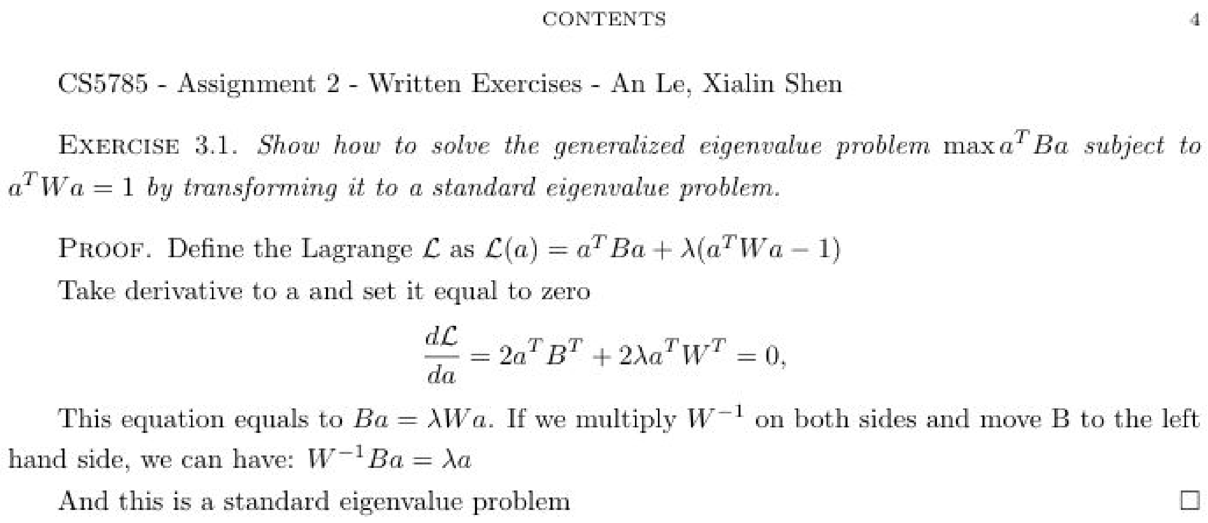
All Successful Selected

Submission and Description	Public Score	Use for Final Score
kaggle_cuisine.csv a day ago by AnLe add submission details	0.78338	☐

PART B - WRITTEN EXERCISE

Question 1

Show how to solve the generalized eigenvalue problem max a^T Ba subject to a^T Wa = 1 by transforming to a standard eigenvalue problem.



CONTENTS 4

CS5785 - Assignment 2 - Written Exercises - An Le, Xialin Shen

EXERCISE 3.1. *Show how to solve the generalized eigenvalue problem* $\max a^T Ba$ *subject to* $a^T Wa = 1$ *by transforming it to a standard eigenvalue problem.*

PROOF. Define the Lagrange $\mathcal{L}$ as $\mathcal{L}(a) = a^T Ba + \lambda(a^T Wa - 1)$

Take derivative to a and set it equal to zero

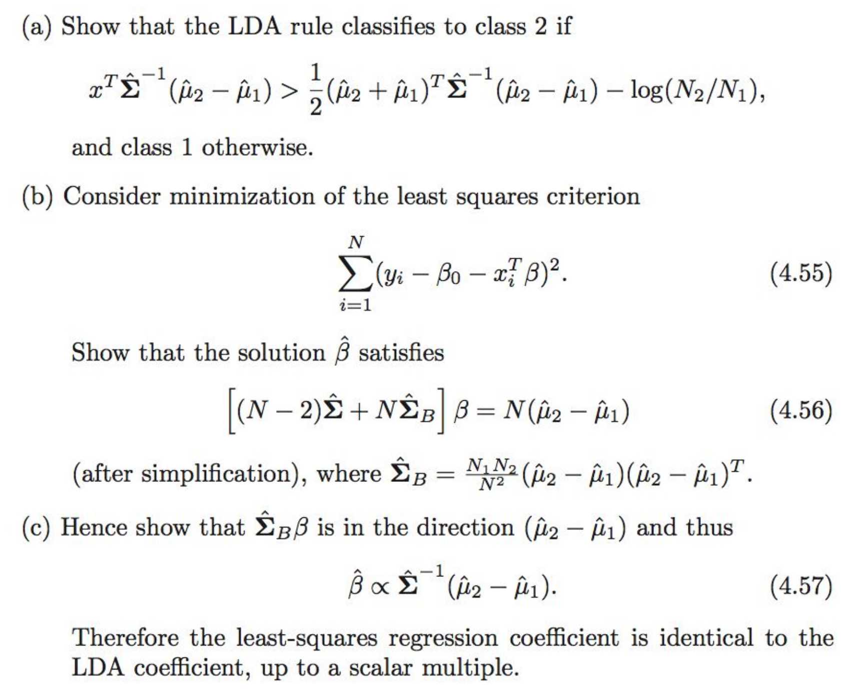
$$\frac{d\mathcal{L}}{da} = 2a^T B^T + 2\lambda a^T W^T = 0,$$

This equation equals to $Ba = \lambda Wa$. If we multiply W^{-1} on both sides and move B to the left hand side, we can have: $W^{-1}Ba = \lambda a$

And this is a standard eigenvalue problem □

Question 2

Suppose we have features x ∈ IRp, a two-class response, with class sizes N1, N2, and the target coded as −N/N1, N/N2.



(a) Show that the LDA rule classifies to class 2 if

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

and class 1 otherwise.

(b) Consider minimization of the least squares criterion

$$\sum_{i=1}^{N}(y_i-\beta_0-x_i^T\beta)^2. \tag{4.55}$$

Show that the solution $\hat{\beta}$ satisfies

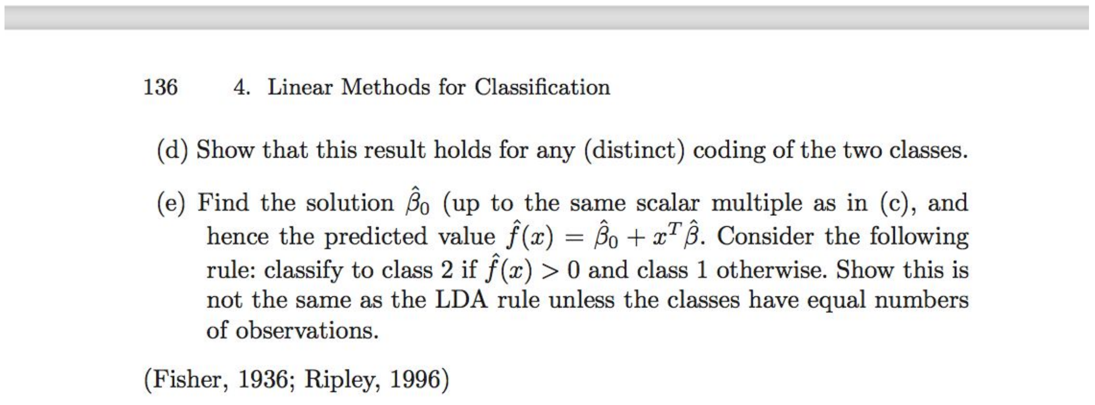
$$\left[(N-2)\hat{\boldsymbol{\Sigma}}+N\hat{\boldsymbol{\Sigma}}_B\right]\beta = N(\hat{\mu}_2-\hat{\mu}_1) \tag{4.56}$$

(after simplification), where $\hat{\boldsymbol{\Sigma}}_B = \frac{N_1N_2}{N^2}(\hat{\mu}_2-\hat{\mu}_1)(\hat{\mu}_2-\hat{\mu}_1)^T$.

(c) Hence show that $\hat{\boldsymbol{\Sigma}}_B\beta$ is in the direction $(\hat{\mu}_2-\hat{\mu}_1)$ and thus

$$\hat{\beta} \propto \hat{\boldsymbol{\Sigma}}^{-1}(\hat{\mu}_2-\hat{\mu}_1). \tag{4.57}$$

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



(d) Show that this result holds for any (distinct) coding of the two classes.

(e) Find the solution $\hat{\beta}_0$ (up to the same scalar multiple as in (c), and hence the predicted value $\hat{f}(x) = \hat{\beta}_0 + x^T\hat{\beta}$. Consider the following rule: classify to class 2 if $\hat{f}(x) > 0$ and class 1 otherwise. Show this is not the same as the LDA rule unless the classes have equal numbers of observations.

(Fisher, 1936; Ripley, 1996)

We found the solution online spent hours but still not able to digest it. The original source is here:
http://www.waxworksmath.com/Authors/G_M/Hastie/WriteUp/Weatherwax_Epstein_Hastie_Solution_Manual.pdf

Ex. 4.2 (two-class classification)

Part (a): Under zero-one classification loss, for each class ω_k the Bayes' discriminant functions $\delta_k(x)$ take the following form

$$\delta_k(x) = \ln(p(x|\omega_k)) + \ln(\pi_k)\,. \tag{91}$$

If our conditional density $p(x|\omega_k)$ is given by a multidimensional normal then its function form is given by

$$p(x|\omega_k) = \mathcal{N}(x;\mu_k,\Sigma_k) \equiv \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right\}\,. \tag{92}$$

Taking the logarithm of this expression as required by Equation 91 we find

$$\ln(p(x|\omega_k)) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{p}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma_k|)\,,$$

so that our discriminant function in the case when $p(x|\omega_k)$ is a multidimensional Gaussian is given by

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{p}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma_k|) + \ln(\pi_k)\,. \tag{93}$$

We will now consider some specializations of this expression for various possible values of Σ_k and how these assumptions modify the expressions for $\delta_k(x)$. Since linear discriminant analysis (LDA) corresponds to the case of equal covariance matrices our decision boundaries (given by Equation 93), but with equal covariances ($\Sigma_k = \Sigma$). For decision purposes we can drop the two terms $-\frac{p}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma|)$ and use a discriminant $\delta_k(x)$ given by

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) + \ln(\pi_k)\,.$$

Expanding the quadratic in the above expression we get

$$\delta_k(x) = -\frac{1}{2}\left(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_k - \mu_k^T\Sigma^{-1}x + \mu_k^T\Sigma^{-1}\mu_k\right) + \ln(\pi_k)\,.$$

Since $x^T\Sigma^{-1}x$ is a common term with the same value in all discriminant functions we can drop it and just consider the discriminant given by

$$\delta_k(x) = \frac{1}{2}x^T\Sigma^{-1}\mu_k + \frac{1}{2}\mu_k^T\Sigma^{-1}x - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln(\pi_k)\,.$$

Since $x^T\Sigma^{-1}\mu_k$ is a scalar, its value is equal to the value of its transpose so

$$x^T\Sigma^{-1}\mu_k = \left(x^T\Sigma^{-1}\mu_k\right)^T = \mu_k^T(\Sigma^{-1})^T x = \mu_k^T\Sigma^{-1}x\,,$$

since Σ^{-1} is symmetric. Thus the two linear terms in the above combine and we are left with

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln(\pi_k)\,. \tag{94}$$

Next we can estimate π_k from data using $\pi_i = \frac{N_i}{N}$ for $i = 1, 2$ and we pick class 2 as the classification outcome if $\delta_2(x) > \delta_1(x)$ (and class 1 otherwise). This inequality can be written as

$$x^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln(\frac{N_2}{N}) > x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \ln(\frac{N_1}{N})\,.$$

or moving all the x terms to one side

$$x^T\Sigma^{-1}(\mu_2 - \mu_1) > \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \ln(\frac{N_1}{N}) - \ln(\frac{N_2}{N})\,,$$

as we were to show.

Part (b): To minimize the expression $\sum_{i=1}^N (y_i - \beta_0 - \beta^T x_i)^2$ over $(\beta_0, \beta)'$ we know that the solution $(\hat{\beta}_0, \hat{\beta})'$ must satisfy the normal equations which in this case is given by

$$X^T X \left[\begin{array}{c} \beta_0 \\ \beta \end{array}\right] = X^T \mathbf{y}\,.$$

Our normal equations have a block matrix $X^T X$ on the left-hand-side given by

$$\left[\begin{array}{cccccccc} 1 & 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{N_1} & x_{N_1+1} & x_{N_1+2} & \cdots & x_{N_1+N_2} \end{array}\right] \left[\begin{array}{cc} 1 & x_1^T \\ 1 & x_2^T \\ \vdots & \\ 1 & x_{N_1}^T \\ 1 & x_{N_1+1}^T \\ 1 & x_{N_1+2}^T \\ \vdots & \\ 1 & x_{N_1+N_2}^T \end{array}\right]\,.$$

When we take the product of these two matrices we find

$$\left[\begin{array}{cc} N & \sum_{i=1}^N x_i^T \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i x_i^T \end{array}\right]\,. \tag{95}$$

For the case where we code our response as $-\frac{N}{N_1}$ for the first class and $+\frac{N}{N_2}$ for the second class (where $N = N_1 + N_2$), the right-hand-side or $X^T y$ of the normal equations becomes

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{N_1} & x_{N_1+1} & x_{N_1+2} & \cdots & x_{N_1+N_2} \end{bmatrix} \begin{bmatrix} -N/N_1 \\ -N/N_1 \\ \vdots \\ -N/N_1 \\ N/N_2 \\ N/N_2 \\ \vdots \\ N/N_2 \end{bmatrix} .$$

When we take the product of these two matrices we get

$$\begin{bmatrix} N_1\left(-\frac{N}{N_1}\right) + N_2\left(\frac{N}{N_2}\right) \\ \left(\sum_{i=1}^{N_1} x_i\right)\left(-\frac{N}{N_1}\right) + \left(\sum_{i=N_1+1}^{N} x_i\right)\left(\frac{N}{N_2}\right) \end{bmatrix} = \begin{bmatrix} 0 \\ -N\mu_1 + N\mu_2 \end{bmatrix} .$$

Note that we can simplify the $(1,2)$ and the $(2,1)$ elements in the block coefficient matrix $X^T X$ in Equation 95 by introducing the class specific means (denoted by μ_1 and μ_2) as

$$\sum_{i=1}^{N} x_i = \sum_{i=1}^{N_1} x_i + \sum_{i=N_1+1}^{N} x_i = N_1\mu_1 + N_2\mu_2 ,$$

Also if we pool all of the samples for this two class problem ($K = 2$) together we can estimate the pooled covariance matrix $\hat{\Sigma}$ (see the section in the book on linear discriminant analysis) as

$$\hat{\Sigma} = \frac{1}{N-K} \sum_{k=1}^{K} \sum_{i:g_i=k} (x_i - \mu_k)(x_i - \mu_k)^T .$$

When $K = 2$ this is

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{N-2}\left[\sum_{i:g_i=1} (x_i - \mu_1)(x_i - \mu_1)^T + \sum_{i:g_i=2} (x_i - \mu_2)(x_i - \mu_2)^T\right] \\ &= \frac{1}{N-2}\left[\sum_{i:g_i=1} x_i x_i^T - N_1\mu_1\mu_1^T + \sum_{i:g_i=1} x_i x_i^T - N_2\mu_2\mu_2^T\right] . \end{aligned}$$

From which we see that the sum $\sum_{i=1}^{N} x_i x_i^T$ found in the $(2,2)$ element in the matrix from Equation 95 can be written as

$$\sum_{i=1}^{N} x_i x_i^T = (N-2)\hat{\Sigma} + N_1\mu_1\mu_1^T + N_2\mu_2\mu_2^T .$$

Now that we have evaluated both sides of the normal equations we can write them down again as a linear system. We get

$$\begin{bmatrix} N & N_1\mu_1^T + N_2\mu_2^T \\ N_1\mu_1 + N_2\mu_2 & (N-2)\hat{\Sigma} + N_1\mu_1\mu_1^T + N_2\mu_2\mu_2^T \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ -N\mu_1 + N\mu_2 \end{bmatrix} . \quad (96)$$

In more detail we can write out the first equation in the above system as

$$N\beta_0 + (N_1\mu_1^T + N_2\mu_2^T)\beta = 0\,,$$

or solving for β_0, in terms of β, we get

$$\beta_0 = \left(-\frac{N_1}{N}\mu_1^T - \frac{N_2}{N}\mu_2^T\right)\beta\,. \tag{97}$$

When we put this value of β_0 into the second equation in Equation 96 we find the total equation for β then looks like

$$(N_1\mu_1 + N_2\mu_2)\left(-\frac{N_1}{N}\mu_1^T - \frac{N_2}{N}\mu_2^T\right)\beta + \left((N-2)\hat{\Sigma} + N_1\mu_1\mu_1^T + N_2\mu_2\mu_2^T\right)\beta = N(\mu_2 - \mu_1)\,.$$

Consider the terms that are outer products of the vectors μ_i (namely terms like $\mu_i\mu_j^T$) we see that taken together they look like

$$\begin{aligned}
\text{Outer Product Terms} &= -\frac{N_1^2}{N}\mu_1\mu_1^T - \frac{2N_1N_2}{N}\mu_1\mu_2^T - \frac{N_2^2}{N}\mu_2\mu_2^T + N_1\mu_1\mu_2^T + N_2\mu_2\mu_2^T \\
&= \left(-\frac{N_1^2}{N} + N_1\right)\mu_1\mu_1^T - \frac{2N_1N_2}{N}\mu_1\mu_2^T + \left(-\frac{N_2^2}{N} + N_2\right)\mu_2\mu_2^T \\
&= \frac{N_1}{N}(-N_1 + N)\,\mu_1\mu_1^T - \frac{2N_1N_2}{N}\mu_1\mu_2^T + \frac{N_2}{N}(-N_2 + N)\,\mu_2\mu_2^T \\
&= \frac{N_1N_2}{N}\mu_1\mu_1^T - \frac{2N_1N_2}{N}\mu_1\mu_2^T + \frac{N_2N_1}{N}\mu_2\mu_2^T \\
&= \frac{N_1N_2}{N}(\mu_1\mu_1^T - 2\mu_1\mu_2 - \mu_2\mu_2) = \frac{N_1N_2}{N}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T\,.
\end{aligned}$$

Here we have used the fact that $N_1 + N_2 = N$. If we introduce the matrix $\hat{\Sigma}_B$ as

$$\hat{\Sigma}_B \equiv (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T\,, \tag{98}$$

we get that the equation for β looks like

$$\left[(N-2)\hat{\Sigma} + \frac{N_1N_2}{N}\hat{\Sigma}_B\right]\beta = N(\mu_2 - \mu_1)\,, \tag{99}$$

as we were to show.

Part (c): Note that $\hat{\Sigma}_B\beta$ is $(\mu_2 - \mu_1)(\mu_2 - \mu_1)^T\beta$, and the product $(\mu_2 - \mu_1)^T\beta$ is a scalar. Therefore the vector *direction* of $\hat{\Sigma}_B\beta$ is given by $\mu_2 - \mu_1$. Thus in Equation 99 as both the right-hand-side and the term $\frac{N_1N_2}{N}\hat{\Sigma}_B$ are in the direction of $\mu_2 - \mu_1$ the solution β must be in the direction (i.e. proportional to) $\hat{\Sigma}^{-1}(\mu_2 - \mu_1)$.

Question 3

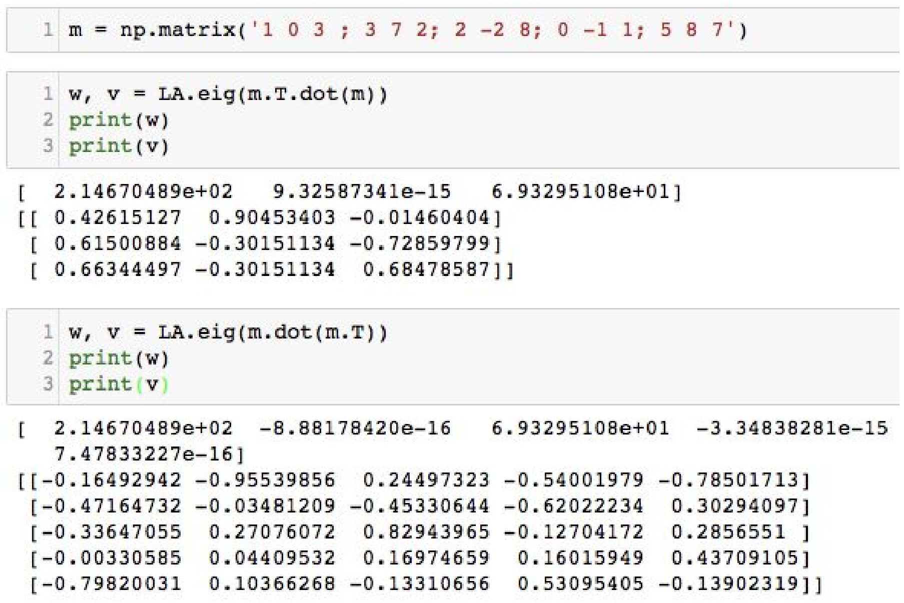
SVD of Rank Deficient Matrix. Consider matrix M. It has rank 2, as you can see by observing that there times the first column minus the other two columns is 0.

(a) Compute the matrices M^T M and MM^T

$$M = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix} \qquad M^T = \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix}$$

$$M^T M = \begin{bmatrix} 39 & 57 & 60 \\ 57 & 118 & 53 \\ 60 & 53 & 127 \end{bmatrix} \qquad M M^T = \begin{bmatrix} 10 & 9 & 26 & 3 & 26 \\ 9 & 62 & 8 & -5 & 85 \\ 26 & 8 & 72 & 10 & 50 \\ 3 & -5 & 10 & 2 & -1 \\ 26 & 85 & 50 & -1 & 138 \end{bmatrix}$$

(b) Find the eigenvalues for your matrices of part (a).



```
m = np.matrix('1 0 3 ; 3 7 2; 2 -2 8; 0 -1 1; 5 8 7')
```

```
w, v = LA.eig(m.T.dot(m))
print(w)
print(v)
```

```
[  2.14670489e+02   9.32587341e-15   6.93295108e+01]
[[ 0.42615127  0.90453403 -0.01460404]
 [ 0.61500884 -0.30151134 -0.72859799]
 [ 0.66344497 -0.30151134  0.68478587]]
```

```
w, v = LA.eig(m.dot(m.T))
print(w)
print(v)
```

```
[  2.14670489e+02  -8.88178420e-16   6.93295108e+01  -3.34838281e-15
   7.47833227e-16]
[[-0.16492942 -0.95539856  0.24497323 -0.54001979 -0.78501713]
 [-0.47164732 -0.03481209 -0.45330644 -0.62022234  0.30294097]
 [-0.33647055  0.27076072  0.82943965 -0.12704172  0.2856551 ]
 [-0.00330585  0.04409532  0.16974659  0.16015949  0.43709105]
 [-0.79820031  0.10366268 -0.13310656  0.53095405 -0.13902319]]
```

Eigenvalues for M^TM : $\lambda1 = 214.67,\ \lambda2 = 69.33,\ \lambda3 = 0$

Eigenvalues for MM^T : $\lambda1 = 214.67,\ \lambda2 = 69.33,\ \lambda3 = 0,\ \lambda4 = 0,\ \lambda5 = 0$

(c) Find the eigenvectors for the matrices of part (a).

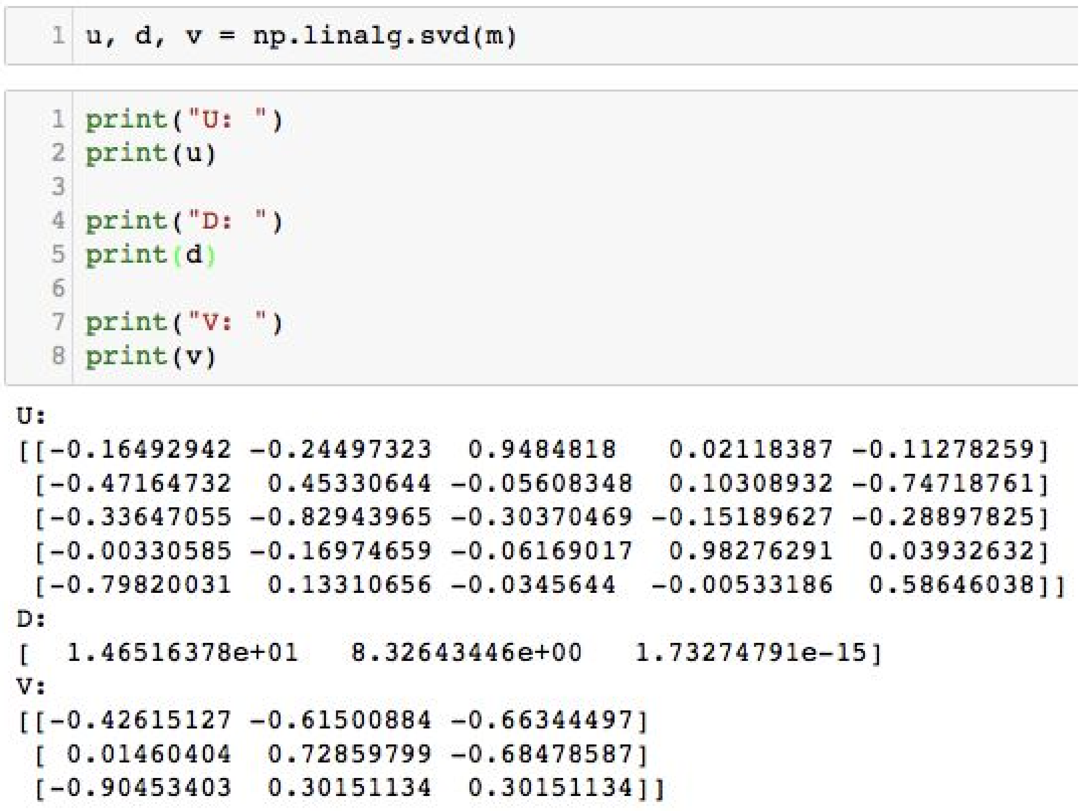
Eigenvectors for $M^T M$:

V1 = [0.426, 0.615, 0.663]   V2 = [-0.015, 0.729, 0.684]

Eigenvectors for $M M^T$:

V1 = [0.165, 0.472, 0.336, 0.003, 0.798]   V2 = [0.245, -0.453, 0.829, 0.170, -0.133]

(d) Find the SVD for the original matrix M from parts (b) and (c). Note that there are only two nonzero eigenvalues, so your matrix Σ should have only two singular values, while U and V have only two columns.

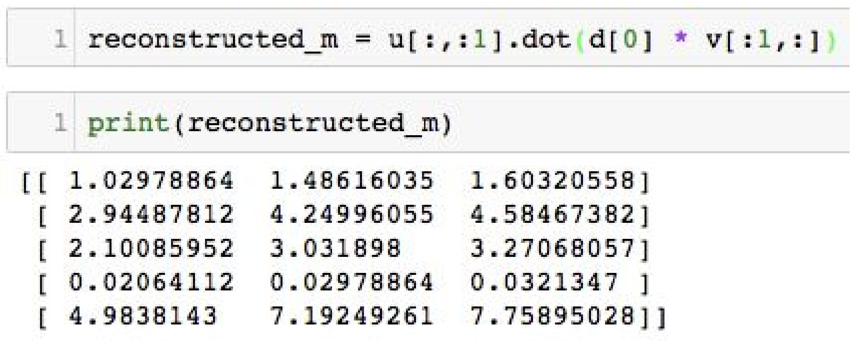


```
u, d, v = np.linalg.svd(m)
```

```
print("U: ")
print(u)

print("D: ")
print(d)

print("V: ")
print(v)
```

```
U:
[[-0.16492942 -0.24497323  0.9484818   0.02118387 -0.11278259]
 [-0.47164732  0.45330644 -0.05608348  0.10308932 -0.74718761]
 [-0.33647055 -0.82943965 -0.30370469 -0.15189627 -0.28897825]
 [-0.00330585 -0.16974659 -0.06169017  0.98276291  0.03932632]
 [-0.79820031  0.13310656 -0.0345644  -0.00533186  0.58646038]]
D:
[  1.46516378e+01   8.32643446e+00   1.73274791e-15]
V:
[[-0.42615127 -0.61500884 -0.66344497]
 [ 0.01460404  0.72859799 -0.68478587]
 [-0.90453403  0.30151134  0.30151134]]
```

(e) Set your smaller singular value to 0 and compute the one-dimensional approximation to the matrix M.



```
reconstructed_m = u[:,:1].dot(d[0] * v[:1,:])
```

```
print(reconstructed_m)
```

```
[[ 1.02978864  1.48616035  1.60320558]
 [ 2.94487812  4.24996055  4.58467382]
 [ 2.10085952  3.031898    3.27068057]
 [ 0.02064112  0.02978864  0.0321347 ]
 [ 4.9838143   7.19249261  7.75895028]]
```

PART C - APPENDIX

CODE

An Le

Xialin Shen

REFERENCES

A Solution Manual and Notes for: The Elements of Statistical Learning

- END -