Lab Report

Homework 2

Name **Jing Bi**
(jb2548)
Xuanrui Zhang
(xz572)

Programming Exercises

1. Eigenface for Face Recognition

(a) Download The Face Dataset. After you unzip `faces.zip`, you will find a folder called *images* which contains all the training and test images; *train.txt* and *test.txt* specifies the training set and test (validation) set split respectively, each line gives an image path and the corresponding label.

Variable name	Meaning
train_labels	Ndarray of the labels of the training data
train_data	Ndarray of the training data
test_labels	Ndarray of the labels of the test data
test_data	Ndarray of the test data
average_face_2d	2d array of the average of training data
train_mean_sub_2d	train_labels subtracte average_face_2d
test_mean_dub_2d	test_labels subtracte average_face_2d
U1	U of the SVD of train_mean_sub_2d
s1	list of the diagonal entries of the SVD of train_mean_sub_2d
Vt1	Vt of the SVD of the train_main_sub_2d
Sigma1	Diagonal matrix with diagonal entries being items in s1
errors	list of errors with increasing r values
F	r dimensional feature vector of the training data
F_test	r-dimensional feature vector of the test data
accuracies	List of accuracies of the logistic regression when r increases.

This problem is about face recognition problem. In order to answer the homework problems, we imported

```
In [1]: import numpy as np
        from numpy import linalg
        from scipy import misc
        from matplotlib import pylab as plt
        import matplotlib.cm as cm
        from sklearn.linear_model import LogisticRegression
        %matplotlib inline
```

the library listed down below:

After downloading the required datasets, we are ready to load the data and do some manipulations with it. Our data includes three parts:

- test.txt (contains the image names of the testing image, along with the labels)
- train.txt (contains the image names of the training image along with the labels)
- Images (contains the face imges)

(b) Load the training set into a matrix **X**: there are 540 training images in total, each has 50 × 50 pixels that need to be concatenated into a 2500-dimensional vector. So the size of **X** should be 540 × 2500, where each row is a flattened face image. Pick a face image from **X** and display that image in grayscale. Do the same thing for the test set. The size of matrix $\mathbf{X}_{\mathbf{test}}$ for the test set should be 100 × 2500.

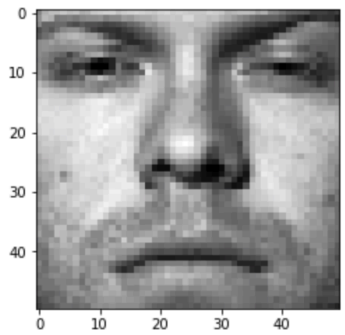
- To load data from the **train** data, we used the code below:

Loading data from training 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(tr
```

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

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

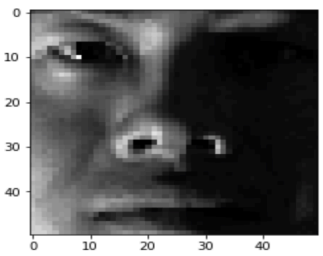


- Similarly, we load data from the **test** file:

```
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_
```

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

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

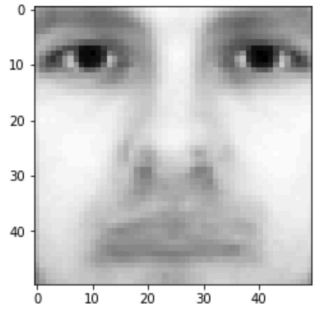


(c) Average Face.

Now, having the train data and test data loaded, we are able to calculate the average face from the whole training set. We use the sum(axis = 0) function to sum up the values of each column, and divide the value by the number of training data that we have (540.)

Therefore we get the average face value:

```
sum_col = train_data.sum(axis = 0)
average_face = [float(x/540) for x in sum_col]
average_face_2d = np.array(average_face)      #make it a 2d array
#plot the average face
plt.imshow(average_face_2d.reshape(50, 50), cmap = cm.Greys_r)
plt.show()
print(len(average_face))
```




```
2500
```

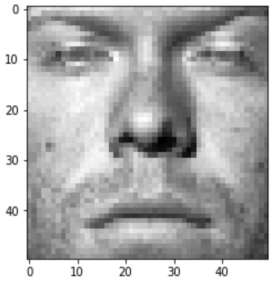
(d) Mean Subtraction.

After calculating the average face, we are able to calculate the matrix subtract the mean. We first do it to the train data.

```
train_mean_sub = []
for i in train_data:
    for j in range(len(i)):
        sub = i[j] - average_face[j]
        train_mean_sub.append(sub)
train_mean_sub = np.array(train_mean_sub)  #turning it into an np array
train_mean_sub_2d = train_mean_sub.reshape(540, 2500)  #reshape it
print(train_mean_sub_2d)
```

```
[[ -54.25185185  -49.10185185  -45.42222222 ...,  -54.22222222
   -44.61851852  -28.27592593]
 [   5.74814815   17.89814815   21.57777778 ...,   43.77777778
    41.38148148   36.72407407]
 [  41.74814815   59.89814815   77.57777778 ...,   -6.22222222
   -12.61851852  -18.27592593]
 ...,
 [  47.74814815   58.89814815   55.57777778 ...,  -41.22222222
   -41.61851852  -35.27592593]
 [ 195.74814815  198.89814815  202.57777778 ...,  -45.22222222
   -42.61851852  -37.27592593]
 [ 114.74814815  121.89814815  149.57777778 ...,  -39.22222222
   -38.61851852  -31.27592593]]
```

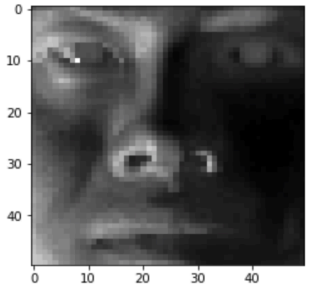
```
plt.imshow(train_mean_sub_2d[10, :].reshape(50, 50), cmap = cm.Greys_r)
plt.show()
```



Similarly, we subtract the test data by the mean that we got from 1(c):

```
test_mean_sub = []
for i in test_data:
    for j in range(len(i)):
        sub = i[j] - average_face[j]
        test_mean_sub.append(sub)
test_mean_sub = np.array(test_mean_sub)  #turning it into an np array
test_mean_sub_2d = test_mean_sub.reshape(100, 2500)  #reshape it
#print(test_mean_sub_2d)
#print(test_mean_sub_2d.shape)
```

```
[[  47.74814815   56.89814815   74.57777778 ...,  -30.22222222
   -26.61851852  -37.27592593]
 [  51.74814815   81.89814815  100.57777778 ...,  -37.22222222
   -39.61851852  -37.27592593]
 [  51.74814815   49.89814815    5.57777778 ...,  -55.22222222
   -49.61851852  -48.27592593]
 ...,
 [ -49.25185185  -45.10185185  -41.42222222 ...,  109.77777778
   122.38148148  109.72407407]
 [  76.74814815   54.89814815   53.57777778 ...,  -50.22222222
   -46.61851852  -42.27592593]
 [  -8.25185185  -20.10185185  -20.42222222 ...,   93.77777778
    73.38148148   72.72407407]]
(100, 2500)
```



(e) Eigenface.

Now we perform SVD to the new mean subtracted train data, using the linalg from numpy library.

Perform SVD on the mean subtracted data

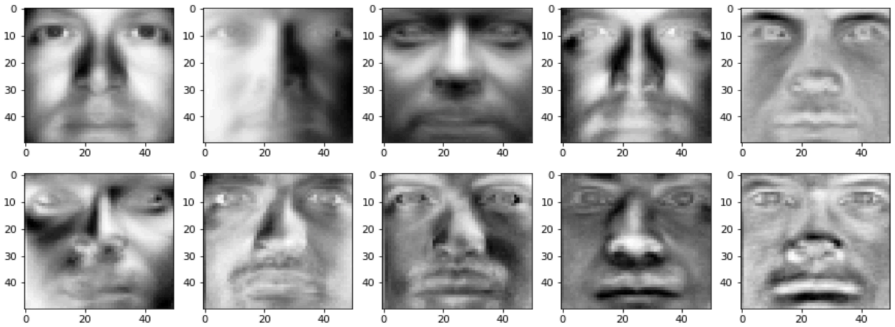
```
# Parsing the mean subtracted matrix into U, sigma, and Vt.
U1, s1, Vt1 = np.linalg.svd(train_mean_sub_2d, full_matrices = False)
```

```
#print(U)
print(U1)
```

```
[[-0.06944235 -0.01080895 -0.00795396 ..., -0.00042292  0.00054699
   0.04303315]
 [ 0.04966457 -0.01866124 -0.03511818 ..., -0.0011864   0.00136944
   0.04303315]
 [ 0.05590978  0.00400083  0.03797568 ...,  0.00248619 -0.00553981
   0.04303315]
 ...,
 [ 0.04161816  0.04679159 -0.01413834 ...,  0.00317775 -0.00373048
   0.04303315]
 [-0.05846347  0.00400296 -0.00947099 ...,  0.00307629 -0.00099639
   0.04303315]
 [ 0.03082678  0.07190205  0.0803649  ..., -0.00063514 -0.0022422
   0.04303315]]
```

```
# Converting a list s into a diagonal matrix
sigma1 = np.zeros((540, 2500), int)
np.fill_diagonal(sigma1, s1)
print(sigma1.shape)
```

```
(540, 2500)
```



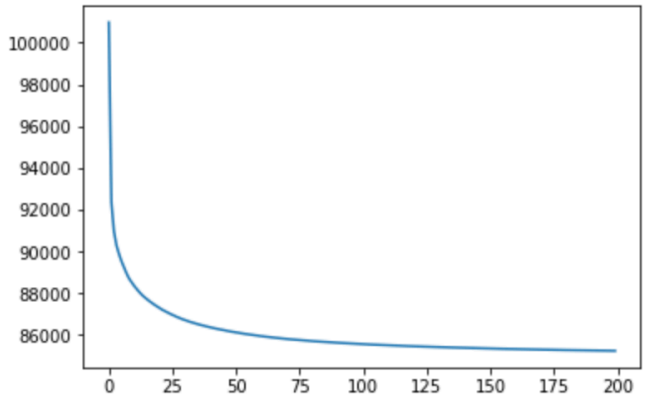
(f) Low-rank Approximation.

Now we do low rank approximation. From the plot down below, we can see that as the value of r increases, our error decreases.

```
def xr1(u, s, vt, r = 20):
    u = U1[:, :r]
    s = sigma1[:r, :r]
    vt1 = Vt1[:r, :]
    x = u.dot(s).dot(vt1)
    return x
```

```
errors = []
x = train_data
for i in range(1, 201):
    x_a = xr1(U1, sigma1, Vt1, r = i)
    e = np.linalg.norm(x - x_a)
    errors.append(e)
```

```
plt.plot(errors)
plt.show()
```



(g) Eigenface Feature.

Now write a function that calculates F and Ftest. To get F, multiply the mean subtracted training data to the transpose of first r rows of V_{t1}

```
def Ftrainrdim(x, vt, r):
    V = Vt1[:r, :]
    V = V.T
    return x.dot(V)
F = Ftrainrdim(train_mean_sub_2d, Vt1, r = 20)
F.shape
```

```
(540, 20)
```

(h) Face Recognition.

Logistic Regression, test on F test.
When r = 10, we get the accuracy of 0.79:

```
F = Ftrainrdim(train_mean_sub_2d, Vt1, r = 10)
F_test = Ftrainrdim(test_mean_sub_2d, Vt1, r = 10)
```

```
lr = LogisticRegression()
lr.fit(F, train_labels)
lr.score(F_test, test_labels)
```

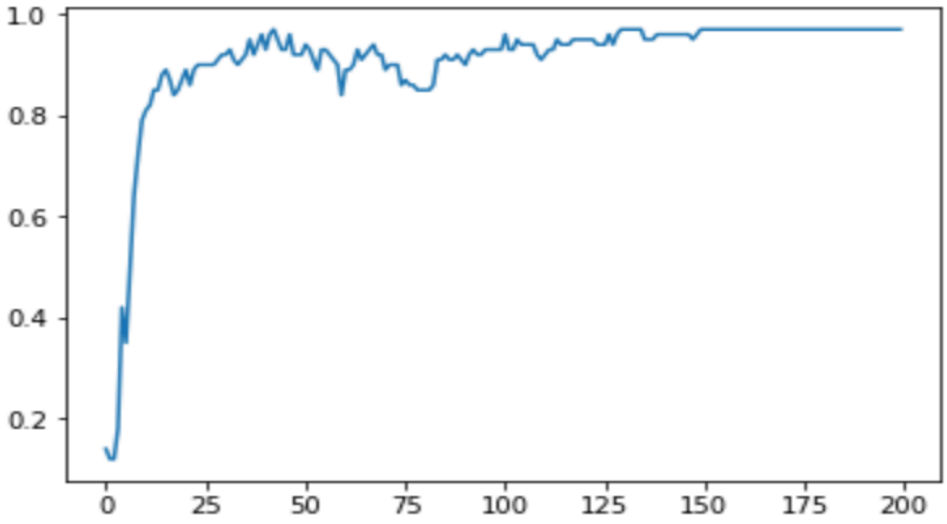
```
0.79000000000000004
```

Ranging r from 1 to 200, we get a list of accuracies. Hence we can observe the trend of the accuracy as r increases.

```
accuracies = []
for i in range(1,201):
    ftrain = Ftrainrdim(train_mean_sub_2d, Vt1, i)
    ftest = Ftrainrdim(test_mean_sub_2d, Vt1, i)
    model = LogisticRegression(multi_class = 'ovr')
    model = model.fit(ftrain, train_labels)
    m = model.score(ftest, test_labels)
    accuracies.append(m)
```

```
print(accuracies)
#plt.figure(figsize = (15,10))
plt.plot(accuracies)
```

The trend looks like this:



The plot of accuracies corresponds to the plot of errors. As r increases, the error decreases, and the accuracy increases.

2. What's Cooking?

(a) Traning and test data

```
#read datasets
with open('train.json') as json_data:
    train = json.load(json_data)

with open('test.json') as json_data:
    test = json.load(json_data)
```

- ***train.json*** - the training *set* containing type of **cuisine**, recipes **id**, and *list* of **ingredients**
- ***test.json*** - the test *set* containing recipes **id**, and *list* of **ingredients**

(b) Tell us about the data.

- How many *samples (dishes)* are there in the **training set**?

```
samples = len(train)
print('There are', samples, 'of samples in the training set.')
```

```
There are 39774 of samples in the training set.
```

- How many *categories (types of cuisine)*?

```
#unique types of cuisine
uni_cuis = set(cuisine)
print('There are',len(list(uni_cuis)),'types of cuisine in the training set, including:')
for i in list(uni_cuis):
       print (i, end = ', ')
```

```
There are 20 types of cuisine in the training set, including:
greek, jamaican, korean, brazilian, southern_us, spanish, filipino, british, french, indian, japanese, irish, italia
n, thai, vietnamese, chinese, cajun_creole, russian, moroccan, mexican,
```

- Use a list to keep all the unique *ingredients* appearing in the **training set**. How many unique ingredients are there?

```
#get the cuisine info out from the training set
ingred = []

for i in list(range(len(train))):
    temp = train[i]['ingredients']
    for j in temp:
        if j not in ingred:
            ingred.append(j)
```

```
print('There are',len(ingred),'types of unique ingredients in the training set.')
```

```
There are 6714 types of unique ingredients in the training set.
```

(c) Represent each dish by a binary ingredient feature vector.

Use *n*×*d* feature matrix to represent all the dishes in **training** set and **test** set, where *n* is the *number of dishes*.

- The *n*×*d* matrix of the **training set**

```
#create an data frame first
train_dish_ingred = pd.DataFrame(0, index=np.arange(samples), columns=ingred)

for d in list(range(samples)):
    vec = train_dish_ingred.iloc[d]
    temp = train[d]['ingredients']
    for item in temp:
        vec[item] = 1
        train_dish_ingred.iloc[d] = vec
```

```
train_dish_ingred.head()
```

	romaine lettuce	black olives	grape tomatoes	garlic	pepper	purple onion	seasoning	garbanzo beans	feta cheese crumbles	plain flour	...	Oscar Mayer Cotto Salami	Challenge Butter	orange glaze	cholesterol free egg substitute	ciabatta loaf	Lipton® Iced Tea Brew Family Size Tea Bags	H Va (Y Or Ra Di
0	1	1	1	1	1	1	1	1	1	0	...	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	
2	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

5 rows × 6714 columns

```
#transform the data frame into a nxd matrix
train_matrix = []
for d in list(range(samples)):
    vec = train_dish_ingred.values[d].tolist()
    train_matrix.append(vec)

train_matrix = np.matrix(train_matrix)
```

```
print(train_matrix)
```

```
[[1 1 1 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
```

The shape of the feature matrix of the training set is 39774 x 6714 .

- The *n*×*d* matrix of the **test set**

	baking powder	eggs	all-purpose flour	raisins	milk	white sugar	sugar	egg yolks	corn starch	cream of tartar	...	fraise	beef heart	lambs liver	soft cheese	sliced mango	pork strips	shark fillets	hash brown	porter	butter crackers
0	1	1	1	1	1	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 4484 columns

```
#transform the data frame into a nxd matrix
test_matrix = []
for d in list(range(len(test))):
    vec = test_dish_ingred.values[d].tolist()
    test_matrix.append(vec)

test_matrix = np.matrix(test_matrix)
```

```
print(test_matrix)
```

```
[[1 1 1 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
```

The shape of the feature matrix of the test set is 9944 x 4484 .

- The *n*×*d* matrix of the **training set** and **test set** combined

```
combine = pd.concat([train_dish_ingred,test_dish_ingred], axis=0, ignore_index=True)
```

```
features = combine.columns.values
print('The sequence of the unique ingredients is', features)
```

```
The sequence of the unique ingredients is ['(    oz.) tomato sauce' '(   oz.) tomato paste'
 '(10 oz.) frozen chopped spinach' ..., 'ziti' 'zucchini'
 'zucchini blossoms']
```

```
print('The shape of the feature matrix of the combined set of training and test data is', combine.shape[0],'x', combine
print('The first', train_dish_ingred.shape[0], 'dishes are of the training set.')
print('The last', test_dish_ingred.shape[0], 'dishes are of the test set.')
```

```
The shape of the feature matrix of the combined set of training and test data is 49718 x 7137 .
The first 39774 dishes are of the training set.
The last 9944 dishes are of the test set.
```

```
#transform the data frame into a nxd matrix
matrix = []
for d in list(range(combine.shape[0])):
    vec = np.nan_to_num(combine.iloc[0].values)
    vec = vec.tolist()
    matrix.append(vec)

matrix = np.matrix(matrix)
```

```
print(matrix)
print('The shape of the feature matrix of the test set is', matrix.shape[0],'x', matrix.shape[1],'.')
print('The first', train_dish_ingred.shape[0], 'dishes are of the training set.')
print('The last', test_dish_ingred.shape[0], 'dishes are of the test set.')
```

```
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
The shape of the feature matrix of the test set is 49718 x 7137 .
The first 39774 dishes are of the training set.
The last 9944 dishes are of the test set.
```

(d) Using Naïve Bayes Classifier to perform 3 fold cross-validation on the training set and report your average classification accuracy

```
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
```

- Dividing training set into 3 parts to deliver cross-validation
- **X**: a matrix of the *ingredients* of each dish
- **Y**: a list of type of *cuisine* corresponding to each dish

- Using **Gaussian** distribution prior assumption

```
X_train1 = np.concatenate((x1, x2))
X_test1 = x3
Y_train1 = np.concatenate((y1, y2))
Y_test1 = y3
```

```
g_clf1 = GaussianNB()
g_clf1.fit(X_train1, Y_train1)
g_pred1 = g_clf1.predict(X_test1)
```

```
g_acc1 = 0
for i in list(range(len(X_test1))):
    if g_pred1[i] == Y_test1[i]:
        g_acc1 += 1

print(g_acc1)
```

```
5006
```

Similarly, we get predicted result for group 2 and 3

```
print(g_acc2)
```

```
5077
```

```
print(g_acc3)
```

```
5025
```

The average classification accuracy of using Gaussian distribution prior assumption is 37.98 %.

- Using **Bernoulli** distribution prior assumption

```
clf1 = BernoulliNB()
clf1.fit(X_train1, Y_train1)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

```
pred1 = clf1.predict(X_test1)
```

```
acc1 = 0
for i in list(range(len(X_test1))):
    if pred1[i] == Y_test1[i]:
        acc1 += 1

print(acc1)
```

```
9107
```

Similarly, we get predicted result for group 2 and 3

```
print(acc2)
```

```
9009
```

```
print(acc3)
```

```
9071
```

The average classification accuracy of using Bernoulli distribution prior assumption is 68.35 %.

(e) For Gaussian prior and Bernoulli prior, which performs better in terms of cross-validation accuracy?

- Since 68.35 % > 37.98 %, the **Bernoulli** prior has *higher accuracy* in terms of cross-validation.
- The **Bernoulli** prior is more accurate in this case because we create an binary table of features that can fit **Bernoulli** better. Also, the **Gaussian** would require continuous variables but we only have discrete and its distribution does not fit the Gaussian's pattern.

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

```
from sklearn.linear_model import LogisticRegression
```

```
logreg1 = LogisticRegression()
logreg1.fit(X_train1, Y_train1)
Y_pred1 = logreg1.predict(X_test1)
```

```
l_acc1 = 0
for i in list(range(len(X_test1))):
    if Y_pred1[i] == Y_test1[i]:
        l_acc1 += 1

print(l_acc1)
```

```
10324
```

Similarly, we get predicted result for group 2 and 3

```
print(l_acc2)
```

```
10237
```

```
print(l_acc3)
```

```
10286
```

```
The average accuracy of using Logistic Regression is 77.56 %.
```

(g) Train your best-performed classifier with all of the training data, and generate test labels on test set.

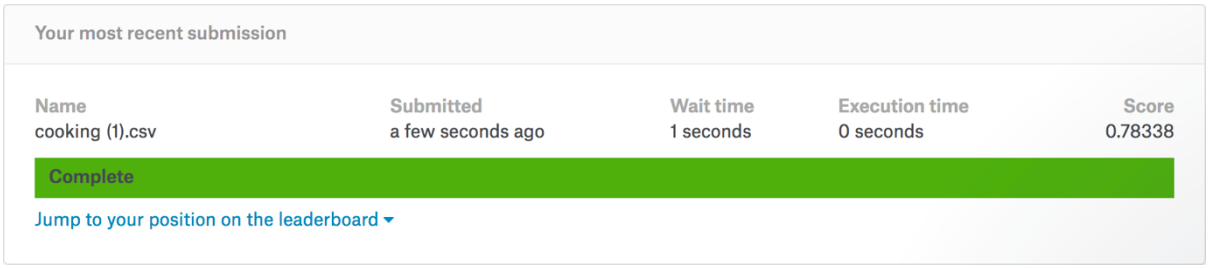
- Since the **Logistic Regression** has *highest accuracy* (77.56 %) among the three classifiers we tried, we choose to use the **Logistic Regression** to train our data and to make prediction.

- Throw the *ingredients* in the **test** set that ***never*** appeared in the **training** set to promote the accuracy of prediction.

```
#create the logistic regression model and predict
model = LogisticRegression()
model.fit(X,Y)
prediction = model.predict(test_mat)
```

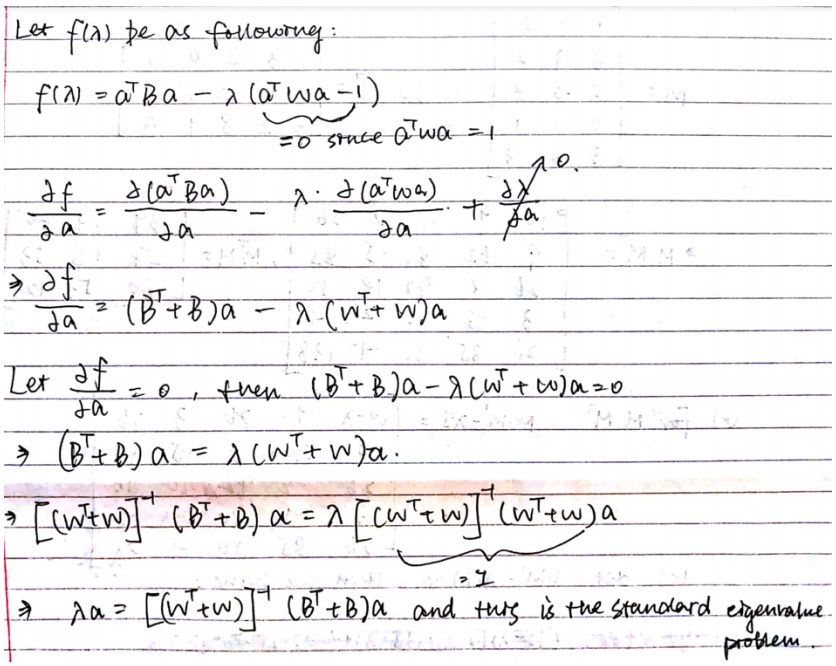
```
The length of the predicted cuisine is 9944
The prediction result of the test set is: ['british' 'southern_us' 'italian' ..., 'italian' 'southern_us' 'mexican']
```

- Write the predicted result into .csv file and submit it to the Kaggle Competition



Written Exercise

Ex. 4.1 Show how to solve the generalized eigenvalue problem $\max a^T \mathbf{B} a$ subject to $a^T \mathbf{W} a = 1$ by transforming to a standard eigenvalue problem.

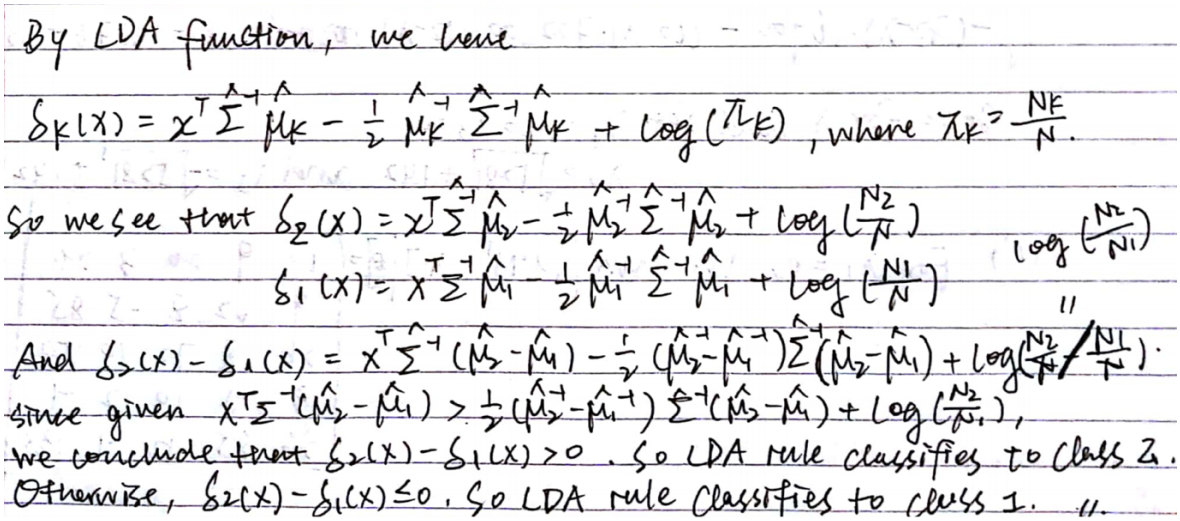


Ex. 4.2 Suppose we have features $x \in \mathbb{R}^p$, a two-class response, with class sizes N_1, N_2, and the target coded as $-N/N_1, N/N_2$.

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

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

and class 1 otherwise.



(b) Consider minimization of the least squares criterion

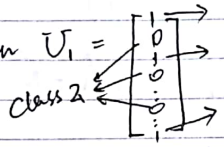
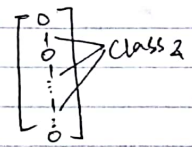
$$\sum_{i=1}^{N}(y_i - \beta_0 - x_i^T\beta)^2. \tag{4.55}$$

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

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

(after simplification), where $\hat{\mathbf{\Sigma}}_B = \frac{N_1 N_2}{N^2}(\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T$.

Let U_i := the n element vector with j^{th} element $\begin{cases} =1 \text{ if it is of class 1} \\ =0 \text{ otherwise} \end{cases}$

then U_1 =  class 1 and U_2 =  class 2

So $U_1 + U_2 = 1$

So we write $Y = t_1 U_1 + t_2 U_2$, where t_i := target label.

And $X^T U_i = N_i \hat{\mu}_i \Leftrightarrow X^T Y = t_1 U_1 \hat{\mu}_1 + t_2 U_2 \hat{\mu}_2$ (*)

In least squares criterion, we see that

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - X_i^T\beta)^2 = (Y - X\beta)^T(Y - X\beta)$$

$$\Updownarrow$$

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

Minimize it, we get: $2X^TX\beta - 2X^TY + 2\beta_0 X^T\cdot 1 = 0$.

$2N\beta_0 - 2\cdot 1^T(Y - X\beta) = 0$.

Solve for β and β_0 with $\hat{\beta}_0 = \frac{1}{N}\cdot 1^T\cdot(Y - X\beta)$

$$(X^TX - \frac{1}{N}X^T\cdot 1\cdot 1^TX)\hat{\beta} = X^TY - \frac{1}{N}X^T\cdot 1\cdot 1^T\cdot Y$$

$$X^TX\hat{\beta} - \frac{1}{N}X^T\cdot 1\cdot 1^T\cdot X\hat{\beta} = X^TY - \frac{1}{N}\cdot X^T\cdot 1\cdot 1^T\cdot Y$$

($X^TX\hat{\beta}$ = X^TY; $X^T\cdot 1$ = $U_1 + U_2$; $1^T\cdot X\hat{\beta}$ = $(U_1+U_2)^T$; $1^T\cdot Y$ = $(U_1+U_2)^T$)

(*) $= (t_1 U_1\hat{\mu}_1 + t_2 U_2\hat{\mu}_2) - \frac{1}{N}(N_1\hat{\mu}_1 + N_2\hat{\mu}_2)(t_1 N_1 + t_2 N_2) = \frac{N_1 N_2}{N}(t_1 - t_2)(\hat{\mu}_1 - \hat{\mu}_2)$

$\Rightarrow X^TX = (N-2)\hat{\Sigma} + N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T$ ①

By definition of $\hat{\Sigma}_B$, we have $X^TX - \frac{1}{N}X^T 1 1^T X = (N-2)\hat{\Sigma} + \frac{N_1 N_2}{N}\hat{\Sigma}_B$ ②

By ① and ② $\Rightarrow$ $\left((N-2)\hat{\Sigma} + \frac{N_1 N_2}{N}\hat{\Sigma}_B\right)\hat{\beta} = \frac{N_1 N_2}{N}(\hat{\mu}_2 - \hat{\mu}_1)(t_1 - t_2)$.

With $t_1 = -\frac{N}{N_1}$ and $t_2 = \frac{N}{N_2}$, we get

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

$$-N\cdot(\hat{\mu}_1 - \hat{\mu}_2) = N(\hat{\mu}_2 - \hat{\mu}_1)$$

Thus, we conclude that

$$\left((N-2)\hat{\Sigma} + \frac{N_1 N_2}{N}\hat{\Sigma}_B\right)\hat{\beta} = N(\hat{\mu}_2 - \hat{\mu}_1) \quad .//.$$

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

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

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

By definition of $\hat{\Sigma}_B$, we have $\hat{\Sigma}_B\hat{\beta} = (\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T\hat{\beta} = \lambda(\hat{\mu}_2 - \hat{\mu}_1)$

where $\lambda = (\hat{\mu}_2 - \hat{\mu}_1)^T\beta$

Then by part (b), we see that

$$\hat{\Sigma}\hat{\beta} = \frac{N - \lambda N_1 N_2}{N(N-2)}(\hat{\mu}_2 - \hat{\mu}_1)$$

so $\hat{\beta} = \frac{N - \lambda N_1 N_2}{N(N-2)}\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) \Rightarrow \hat{\beta} \propto \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$. //.

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

Suppose we have a new coding, denote it as y_i'.

Then y_i' can always be written as $y_i' = ay_i + b$

$$\Rightarrow \sum_{i=1}^{N}(y_i' - \beta_0 - \beta^T x_i)^2 = \sum_{i=1}^{N}\left((ay_i + b) - \beta_0 - \beta^T x_i\right)^2$$

$$= \sum_{i=1}^{N}\left[a\left(y_i + \frac{b}{a} - \frac{\beta_0}{a} - \frac{\beta^T}{a}x_i\right)\right]^2$$

$$= \sum_{i=1}^{N} a^2\left(y_i + \frac{1}{a}(b - \beta_0) - \frac{1}{a}\beta^T x_i\right)^2.$$

So we get $\hat{\beta}_0' = b - a\hat{\beta}_0$ and $\hat{\beta}^* = a\hat{\beta}$

Thus $\hat{\beta}_0'$ is proportional to $\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$.

(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.

By part (b), we can write

$$\hat{\beta}_0 = \frac{1}{N} \cdot \underbrace{1^T}_{(v_1+v_2)^T} \cdot (Y - X\hat{\beta}) = \frac{1}{N}(t_1 N_1 + t_2 N_2) - \frac{1}{N} \cdot 1^T X\hat{\beta}$$

$$= -\frac{1}{N}(N_1\hat{\mu}_1^T + N_2\hat{\mu}_2^T)\hat{\beta}$$

Then the predicted value $\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}^T x$ can be written as

$$\hat{f}(x) = -\frac{1}{N}(N_1\hat{\mu}_1^T + N_2\hat{\mu}_2^T)\hat{\beta} + x^T \cdot \hat{\beta}$$

$$= \frac{1}{N}(Nx^T - N_1\hat{\mu}_1^T - N_2\hat{\mu}_2^T)\hat{\beta}$$

By part (d), $\hat{\beta} = \lambda\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$, where $\lambda \in \mathbb{R}$, then

$$\hat{f}(x) = \frac{1}{N}(Nx^T - N_1\hat{\mu}_1^T - N_2\hat{\mu}_2^T)\lambda\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

If $\lambda > 0$, then

$$f(x) > 0 \Leftrightarrow x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) - \left(\frac{N_1}{N}\hat{\mu}_1 + \frac{N_2}{N}\hat{\mu}_2\right)^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > 0$$

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

And this is different from the LDA rule, (if $N_1 \neq N_2$)

Inspired by: http://statweb.stanford.edu/~tibs/stat315a/Homework/solution2.pdf

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.

$$M = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix}. \tag{1}$$

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

Since $M = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix}$, we have $M^T = \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix}$.

$$\mathbf{M^TM} = \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix} = \begin{bmatrix} \mathbf{39} & \mathbf{57} & \mathbf{60} \\ \mathbf{57} & \mathbf{118} & \mathbf{53} \\ \mathbf{60} & \mathbf{53} & \mathbf{127} \end{bmatrix};$$

$$\mathbf{MM^T} = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix} = \begin{bmatrix} \mathbf{10} & \mathbf{9} & \mathbf{26} & \mathbf{3} & \mathbf{26} \\ \mathbf{9} & \mathbf{62} & \mathbf{8} & \mathbf{-5} & \mathbf{85} \\ \mathbf{26} & \mathbf{8} & \mathbf{72} & \mathbf{10} & \mathbf{50} \\ \mathbf{3} & \mathbf{-5} & \mathbf{10} & \mathbf{2} & \mathbf{-1} \\ \mathbf{26} & \mathbf{85} & \mathbf{50} & \mathbf{-1} & \mathbf{138} \end{bmatrix}.$$

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

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

(A) Solve for M^TM in part (a)

Let eigenvalues be represented by λ and $\boldsymbol{A = M^TM}$, then λ need to satisfy $det(A - \lambda \cdot I) = 0$

We have $A - \lambda \cdot I = \begin{bmatrix} 39 - \lambda & 57 & 60 \\ 57 & 118 - \lambda & 53 \\ 60 & 53 & 127 - \lambda \end{bmatrix}$,

and $det(A - \lambda \cdot I) = \begin{vmatrix} 39-\lambda & 57 & 60 \\ 57 & 118-\lambda & 53 \\ 60 & 53 & 127-\lambda \end{vmatrix} = -\lambda^3 + 284 \times \lambda^2 - 14883 \times \lambda = (-1)(\lambda)(\lambda - (\sqrt{5281} + 142))(\lambda + (\sqrt{5281} - 142))$

Thus, the ***eigenvalue*** of $\boldsymbol{M^TM}$ are: $\boldsymbol{\lambda_1 = 0,\ \lambda_2 = \sqrt{5281} + 142,\ \lambda_3 = -\sqrt{5281} + 142.}$

For each λ, we find its own vectors:

(i) $\lambda_1 = 0$

$$[A - \lambda_1 \cdot I | 0] = \left(\begin{array}{ccc|c} 39 & 57 & 60 & 0 \\ 57 & 118 & 53 & 0 \\ 60 & 53 & 127 & 0 \end{array}\right) \xrightarrow{Row\ Operations} \left(\begin{array}{ccc|c} 1 & 0 & 3 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right).$$

We see that $x_1 = -3x_3$, and $x_2 = x_3$,

so the ***eigenvector*** for eigenvalue $\boldsymbol{\lambda_1 = 0}$ is $\boldsymbol{V_{\lambda 1}} = [-3, 1, 1]^T$

(ii) $\lambda_2 = \sqrt{5281} + 142$

$$[A - \lambda_2 \cdot I | 0] = \left(\begin{array}{ccc|c} -\sqrt{5281} - 103 & 57 & 60 & 0 \\ 57 & -\sqrt{5281} - 24 & 53 & 0 \\ 60 & 53 & -\sqrt{5281} - 15 & 0 \end{array}\right) \xrightarrow{Row\ Operations} \left(\begin{array}{ccc|c} 1 & 0 & \frac{-\sqrt{5281}-68}{219} & 0 \\ 0 & 1 & \frac{-\sqrt{5281}+5}{73} & 0 \\ 0 & 0 & 0 & 0 \end{array}\right).$$

We see that $x_1 = \frac{\sqrt{5281}+68}{219} x_3$, and $x_2 = \frac{\sqrt{5281}-5}{73} x_3$,

so the ***eigenvector*** for eigenvalue $\boldsymbol{\lambda_2 = \sqrt{5281} + 142}$ is $\boldsymbol{V_{\lambda 2}} = [\frac{\sqrt{5281}+68}{219}, \frac{\sqrt{5281}-5}{73}, 1]^T$

(iii) $\lambda_3 = -\sqrt{5281} + 142$

$$[A - \lambda_3 \cdot I|0] = \begin{pmatrix} \sqrt{5281} - 103 & 57 & 60 & 0 \\ 57 & \sqrt{5281} - 24 & 53 & 0 \\ 60 & 53 & \sqrt{5281} - 15 & 0 \end{pmatrix} \xrightarrow{Row\ Operations} \begin{pmatrix} 1 & 0 & \frac{\sqrt{5281}-68}{219} & 0 \\ 0 & 1 & \frac{\sqrt{5281}+5}{73} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We see that $x_1 = \frac{-\sqrt{5281}+68}{219} x_3$, and $x_2 = \frac{-\sqrt{5281}-5}{73} x_3$,

so the ***eigenvector*** for eigenvalue $\lambda_3 = -\sqrt{5281} + 142$ is $V_{\lambda 3} = \left[\frac{-\sqrt{5281}+68}{219}, \frac{-\sqrt{5281}-5}{73}, 1\right]^T$

(B) Solve for MM^T in part (a)

Let eigenvalues be represented by λ and $A = MM^T$, then λ need to satisfy $det(A - \lambda \cdot I) = 0$

We have $A - \lambda \cdot I = \begin{bmatrix} 10-\lambda & 9 & 26 & 3 & 26 \\ 9 & 62-\lambda & 8 & -5 & 85 \\ 26 & 8 & 72-\lambda & 10 & 50 \\ 3 & -5 & 10 & 2-\lambda & -1 \\ 26 & 85 & 50 & -1 & 138-\lambda \end{bmatrix}$,

and $det(A - \lambda \cdot I) =$

$$\begin{vmatrix} 10-\lambda & 9 & 26 & 3 & 26 \\ 9 & 62-\lambda & 8 & -5 & 85 \\ 26 & 8 & 72-\lambda & 10 & 50 \\ 3 & -5 & 10 & 2-\lambda & -1 \\ 26 & 85 & 50 & -1 & 138-\lambda \end{vmatrix} = -\lambda^5 + 284 \times \lambda^4 - 14883 \times \lambda^3 = (-1)(\lambda)(\lambda)(\lambda)(\lambda - (\sqrt{5281} + 142))(\lambda + (\sqrt{5281} - 142))$$

Thus, the ***eigenvalue*** of MM^T are:

$$\lambda_1 = \lambda_2 = \lambda_3 = 0,\ \lambda_4 = \sqrt{5281} + 142,\ \lambda_5 = -\sqrt{5281} + 142.$$

For each λ, we find its own vectors:

(i) $\lambda_1 = \lambda_2 = \lambda_3 = 0$

$$[A - \lambda_1 \cdot I|0] = \begin{pmatrix} 10 & 9 & 26 & 3 & 26 & 0 \\ 9 & 62 & 8 & -5 & 85 & 0 \\ 26 & 8 & 72 & 10 & 50 & 0 \\ 3 & -5 & 10 & 2 & -1 & 0 \\ 26 & 85 & 50 & -1 & 138 & 0 \end{pmatrix} \xrightarrow{Row\ Operations} \begin{pmatrix} 1 & 0 & \frac{20}{7} & \frac{3}{7} & \frac{11}{7} & 0 \\ 0 & 1 & -\frac{2}{7} & -\frac{1}{7} & \frac{8}{7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

We see that $x_1 = -\frac{20}{7}x_3 - \frac{3}{7}x_4 - \frac{11}{7}x_5$ and $x_2 = \frac{2}{7}x_3 + \frac{1}{7}x_4 - \frac{8}{7}x_5$

so the ***eigenvector*** for eigenvalue $\lambda_1 = 0$ is $V_{\lambda 1} = [-\frac{20}{7}, \frac{2}{7}, 1, 0, 0]^T$, $V_{\lambda 1}' = [-\frac{3}{7}, \frac{1}{7}, 0, 1, 0]^T$, and $V_{\lambda 1}'' = [-\frac{11}{7}, -\frac{8}{7}, 0, 0, 1]^T$

(ii) $\lambda_4 = \sqrt{5281} + 142$

$$[A - \lambda_4 \cdot I|0] = \begin{pmatrix} -\sqrt{5281} - 132 & 9 & 26 & 3 & 26 \\ 9 & -\sqrt{5281} - 80 & 8 & -5 & 85 \\ 26 & 8 & -\sqrt{5281} - 70 & 10 & 50 \\ 3 & -5 & 10 & -\sqrt{5281} - 140 & -1 \\ 26 & 85 & 50 & -1 & -\sqrt{5281} - 4 \end{pmatrix} \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\xrightarrow{Row\ Operations} \left(\begin{array}{cccc|c} & & & & \frac{-\sqrt{5281}+58}{71} \\ 1 & 0 & 0 & 0 & \frac{11\sqrt{5281}-1135}{568} \\ 0 & 1 & 0 & 0 & \frac{-13\sqrt{5281}+825}{284} \\ 0 & 0 & 1 & 0 & \frac{-5\sqrt{5281}+361}{568} \\ 0 & 0 & 0 & 1 & 0 \end{array}\right| \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Bigg).$$

We see that $x_1 = \frac{\sqrt{5281}-58}{71}x_5$, $x_2 = \frac{-11\sqrt{5281}+1135}{568}x_5$, $x_3 = \frac{13\sqrt{5281}-825}{284}x_5$, and $x_4 = \frac{5\sqrt{5281}-361}{568}x_5$

so the ***eigenvector*** for eigenvalue $\boldsymbol{\lambda_4 = \sqrt{5281} + 142}$ is $\boldsymbol{V_{\lambda 4}} = [\frac{\sqrt{5281}-58}{71}, \frac{-11\sqrt{5281}+1135}{568}, \frac{13\sqrt{5281}-825}{284}, \frac{5\sqrt{5281}-361}{568}, 1\ ]^T$

(iii) $\lambda_5 = -\sqrt{5281} + 142$

$$[A - \lambda_5 \cdot I|0] = \begin{pmatrix} \sqrt{5281} - 132 & 9 & 26 & 3 & 26 \\ 9 & \sqrt{5281} - 80 & 8 & -5 & 85 \\ 26 & 8 & \sqrt{5281} - 70 & 10 & 50 \\ 3 & -5 & 10 & \sqrt{5281} - 140 & -1 \\ 26 & 85 & 50 & -1 & \sqrt{5281} - 4 \end{pmatrix} \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\xrightarrow{Row\ Operations} \left(\begin{array}{cccc|c} & & & & \frac{\sqrt{5281}+58}{71} \\ 1 & 0 & 0 & 0 & \frac{-11\sqrt{5281}-1135}{568} \\ 0 & 1 & 0 & 0 & \frac{13\sqrt{5281}+825}{284} \\ 0 & 0 & 1 & 0 & \frac{5\sqrt{5281}+361}{568} \\ 0 & 0 & 0 & 1 & 0 \end{array}\right| \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Bigg)$$

We see that $x_1 = \frac{-\sqrt{5281}-58}{71}x_5$, $x_2 = \frac{11\sqrt{5281}+1135}{568}x_5$, $x_3 = \frac{-13\sqrt{5281}-825}{284}x_5$, and $x_4 = \frac{-5\sqrt{5281}-361}{568}x_5$

so the ***eigenvector*** for eigenvalue $\boldsymbol{\lambda_5 = \sqrt{5281} + 142}$ is $\boldsymbol{V_{\lambda 5}} = [\frac{-\sqrt{5281}-58}{71}, \frac{11\sqrt{5281}+1135}{568}, \frac{-13\sqrt{5281}-825}{284}, \frac{-5\sqrt{5281}-361}{568}, 1\ ]^T$

(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.

The **non-zero eigenvalues** are $\lambda_1 = \sqrt{5281} + 142$ and $\lambda_2 = -\sqrt{5281} + 142$, where $\lambda_1 > \lambda_2$.

So $\Sigma = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix} = \begin{bmatrix} \sqrt{\sqrt{5281} + 142} & 0 \\ 0 & \sqrt{-\sqrt{5281} + 142} \end{bmatrix} \approx \begin{bmatrix} 14.6516 & 0 \\ 0 & 8.3264 \end{bmatrix}$.

The eigenvectors corresponding to the non-zero eigenvalues of $\mathbf{MM^T}$ is $V_{\lambda 4} = [\frac{\sqrt{5281}-58}{71}, \frac{-11\sqrt{5281}+1135}{568}, \frac{13\sqrt{5281}-825}{284}, \frac{5\sqrt{5281}-361}{568}, 1\ ]^T$ and $V_{\lambda 5} = [\frac{-\sqrt{5281}-58}{71}, \frac{11\sqrt{5281}+1135}{568}, \frac{-13\sqrt{5281}-825}{284}, \frac{-5\sqrt{5281}-361}{568}, 1\ ]^T$.

Now, we **normalize** the eigenvectors we have and get:

$$V'_{\lambda 4} = \frac{V_{\lambda 4}}{|V_{\lambda 4}|} = [\ \frac{\sqrt{5281}-58}{71}, \frac{-11\sqrt{5281}+1135}{568}, \frac{13\sqrt{5281}-825}{284}, \frac{5\sqrt{5281}-361}{568}, 1\ ]^T \ / \ 1.2528183536480515$$

$$= [0.16492942, 0.47164732, 0.33647055, 0.00330585, 0.79820031]^T$$

$$V'_{\lambda 5} = \frac{V_{\lambda 5}}{|V_{\lambda 5}|} = \ [\ \frac{-\sqrt{5281}-58}{71}, \frac{11\sqrt{5281}+1135}{568}, \frac{-13\sqrt{5281}-825}{284}, \frac{-5\sqrt{5281}-361}{568}, 1\ ]^T \ / \ 7.5127776383349785$$

$$= [-0.24497323, 0.45330644, -0.82943965, -0.16974659, 0.13310656]^T$$

$$\text{so U} = [\ V'_{\lambda 4}, V'_{\lambda 5}] \approx \begin{bmatrix} 0.16492942 & -0.24497323 \\ 0.47164732 & 0.45330644 \\ 0.33647055 & -0.82943965 \\ 0.00330585 & -0.16974659 \\ 0.79820031 & 0.13310656 \end{bmatrix}$$

The eigenvectors corresponding to the non-zero eigenvalues of $\mathbf{M^TM}$ is $\boldsymbol{V_{\lambda 2}} = [\ \frac{\sqrt{5281}+68}{219}, \frac{\sqrt{5281}-5}{73}, 1\ ]^T$ *and* $\boldsymbol{V_{\lambda 3}} = [\ \frac{-\sqrt{5281}+68}{219}, \frac{-\sqrt{5281}-5}{73}, 1\ ]^T$.

Similarly, we **normalize** the eigenvectors and get:

$$\boldsymbol{V'_{\lambda 2}} = \frac{V_{\lambda 2}}{|V_{\lambda 2}|} = [\ \frac{\sqrt{5281}+68}{219}, \frac{\sqrt{5281}-5}{73}, 1\ ]^T \ / \ 1.5072840282244717 = [0.42615127, 0.61500884, 0.66344497]^T$$

$$\boldsymbol{V'_{\lambda 3}} = \frac{V_{\lambda 3}}{|V_{\lambda 3}|} = [\ \frac{-\sqrt{5281}+68}{219}, \frac{-\sqrt{5281}-5}{73}, 1\ ]^T \ / \ 1.46031050680193 = [-0.01460404, -0.72859799, 0.68478587]^T$$

$$\text{so V} = [\ V'_{\lambda 2}, V'_{\lambda 3}] \approx \begin{bmatrix} 0.42615127 & -0.01460404 \\ 0.61500884 & -0.72859799 \\ 0.66344497 & 0.68478587 \end{bmatrix}.$$

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

We have $\boldsymbol{\Sigma} = \begin{bmatrix} 14.6516 & 0 \\ 0 & 8.3264 \end{bmatrix}$, and set the smaller singular value to 0 and get $\boldsymbol{\Sigma_1} = \begin{bmatrix} 14.6516 & 0 \\ 0 & 0 \end{bmatrix}$
Let M' represent the one-dimensional approximation of matrix M, then we have :

$$\text{M'} = \text{U}\boldsymbol{\Sigma_1}\mathbf{V^T}$$

$$= \begin{bmatrix} 0.16492942 & -0.24497323 \\ 0.47164732 & 0.45330644 \\ 0.33647055 & -0.82943965 \\ 0.00330585 & -0.16974659 \\ 0.79820031 & 0.13310656 \end{bmatrix} \begin{bmatrix} 14.6516 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.42615127 & 0.61500884 & 0.66344497 \\ -0.01460404 & -0.72859799 & 0.68478587 \end{bmatrix}$$

$$= \begin{bmatrix} 1.0298 & 1.4862 & 1.6032 \\ 2.9449 & 4.2500 & 4.5847 \\ 2.1009 & 3.0319 & 3.2707 \\ 0.0206 & 0.0298 & 0.0321 \\ 4.9838 & 7.1925 & 7.7590 \end{bmatrix}$$