CS 5785 Applied Machine learning

Homework 2

Team Member: Xueqi Wei, Queenie Liu

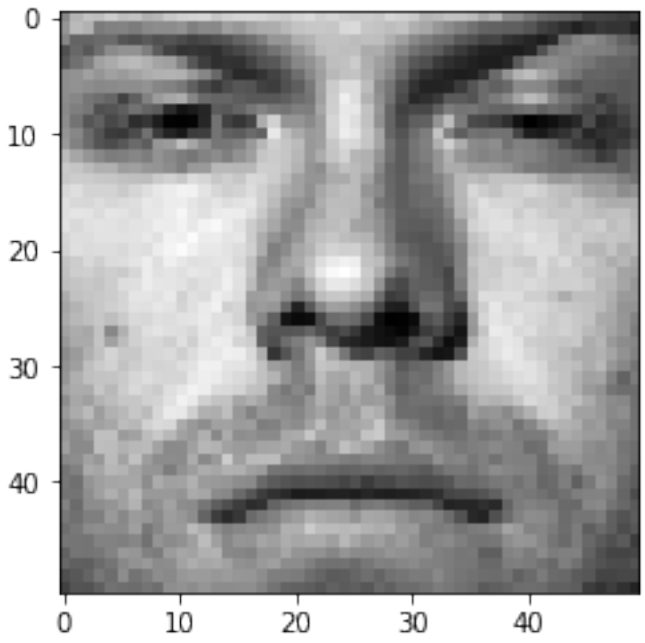
hw2-Q1

October 10, 2019

[2]:
```
import numpy as np
from scipy import misc
from matplotlib import pylab as plt
import matplotlib.cm as cm
import imageio
%matplotlib inline
```

[3]:
```
#(b)
#loading data from the training data set
train_labels, train_data = [], []
for line in open('./faces/train.txt'):
    im = imageio.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()
print(train_data)
```

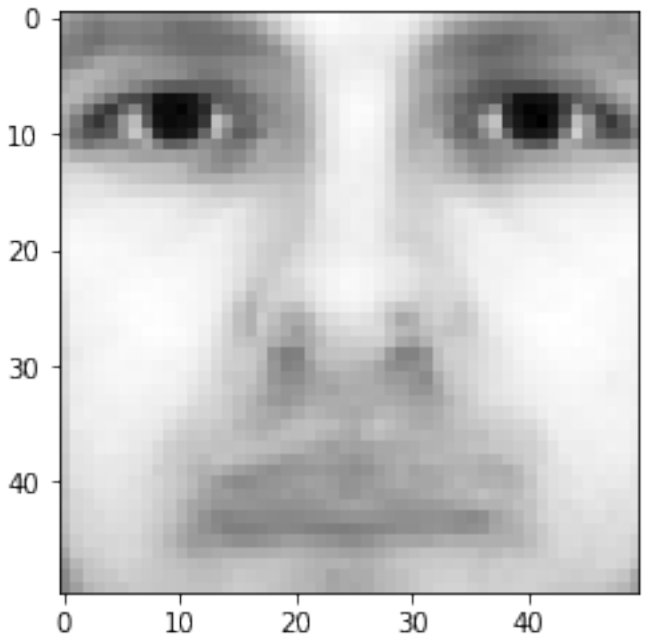
```
(540, 2500) (540,)
```



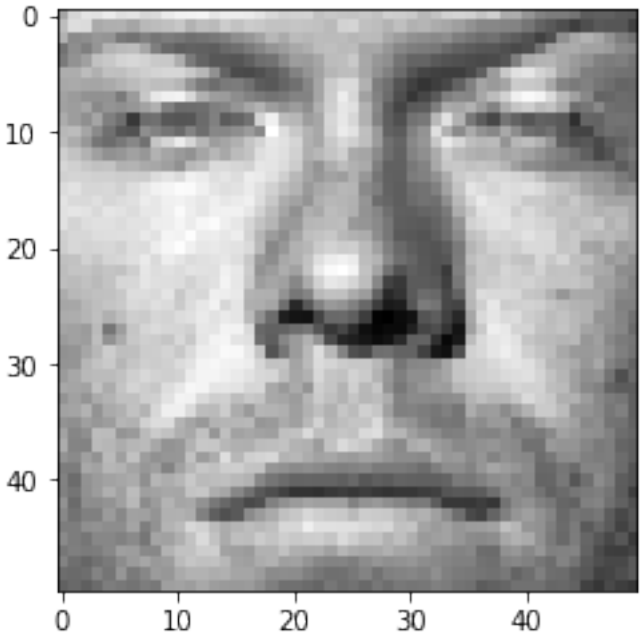

```
[[  5.   7.   7. …  13.  20.  31.]
 [ 65.  74.  74. … 111. 106.  96.]
 [101. 116. 130. …  61.  52.  41.]
 …
 [107. 115. 108. …  26.  23.  24.]
 [255. 255. 255. …  22.  22.  22.]
 [174. 178. 202. …  28.  26.  28.]]
```

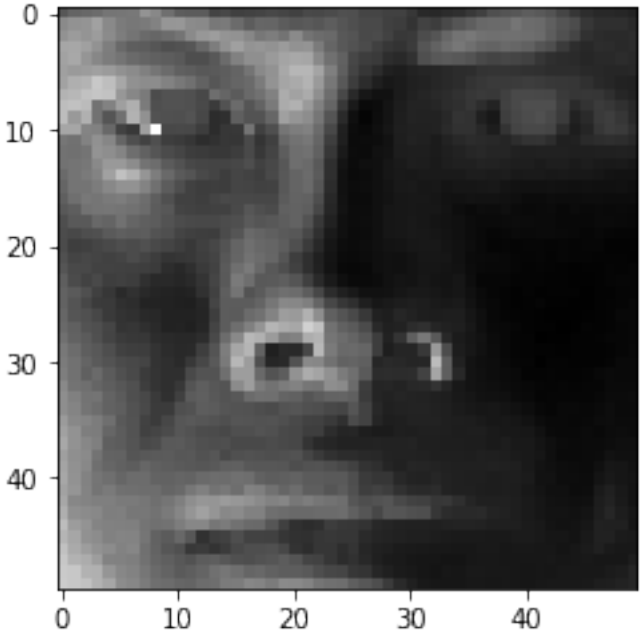
[4]:
```
# (c)
#Average Face
#Compute the average face from the whole training set
#by summing up every row in X then dividing by the number of faces
average_face = np.sum(train_data, axis = 0) /train_labels.shape[0]
print(average_face)
#Display the average face as a grayscale image.
plt.imshow(average_face.reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

```
[59.25185185 56.10185185 52.42222222 … 67.22222222 64.61851852
 59.27592593]
```



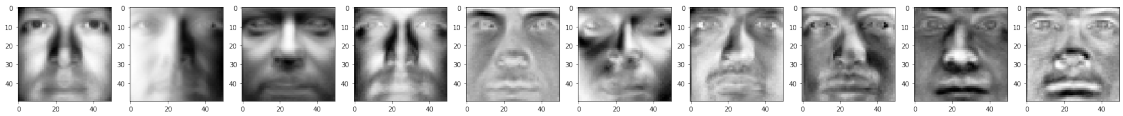
[5]:
```
#(d)
#Mean Subtraction.
#Subtract average face μ from every row in X.
sub_avg_face = train_data - average_face

#Pick a face image after mean subtraction from the new X and display that image␣
 ↪in grayscale.
plt.imshow(sub_avg_face[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()

#Do the same thing for the test set Xtest using the pre-computed average face μ␣
 ↪in (c).
#loading data from the testing data set
test_labels, test_data = [], []
for line in open('./faces/test.txt'):
    im = imageio.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)

#Subtract average face μ from every row in Xtest.
sub_avg_face_test = test_data - average_face
plt.imshow(sub_avg_face_test[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```





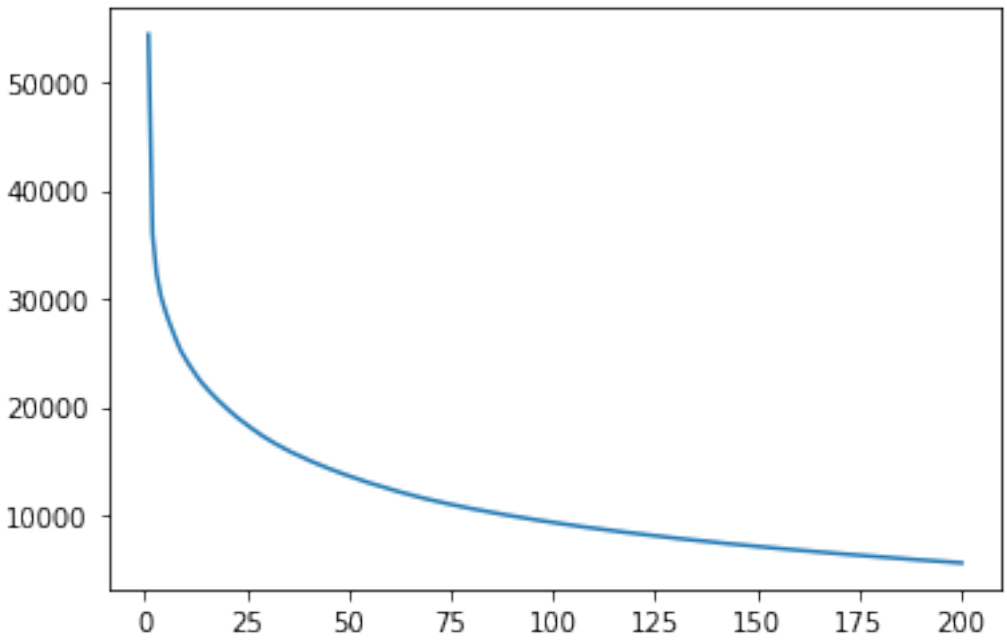
[6]:
```
#(e)
#Eigenface.
#Perform Singular Value Decomposition (SVD) on training set X (X = UΣVT) to get␣
 ↪matrix VT
```

```
u, s, vt = np.linalg.svd(sub_avg_face)
s = np.diag(s)
#create 10 subplots and set appropriate size
fig, ax = plt.subplots(nrows=1, ncols=10)
fig.set_size_inches(30,30)

#We refer to vi,the i-th row of V T, as i-th eigenface.
#Display the first 10 eigenfaces as 10 images in grayscale
for i in range(10):
    ax[i].imshow(vt[i, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



[7]:
```
#(f)
#Low-rank Approximation.
#we can approximate X by X^r = U[:,: r ] Σ[: r,: r ] VT[: r,:].
#The matrix X^r is called rank-r approximation of X.

err = []
for r in range(1,201):
    temp = u[:,:r].dot(s[:r,:r])
    Xr = temp.dot(vt[:r,:])
    err_fnorm = np.linalg.norm(sub_avg_face - Xr, ord='fro')
    err.append(err_fnorm)
#Plot the rank-r approximation error ||X-X^r||F as a function of r when r = 1,␣
 ↪2,..., 200.
plt.plot(range(1,201),err)
```

[7]: [<matplotlib.lines.Line2D at 0x111425e10>]



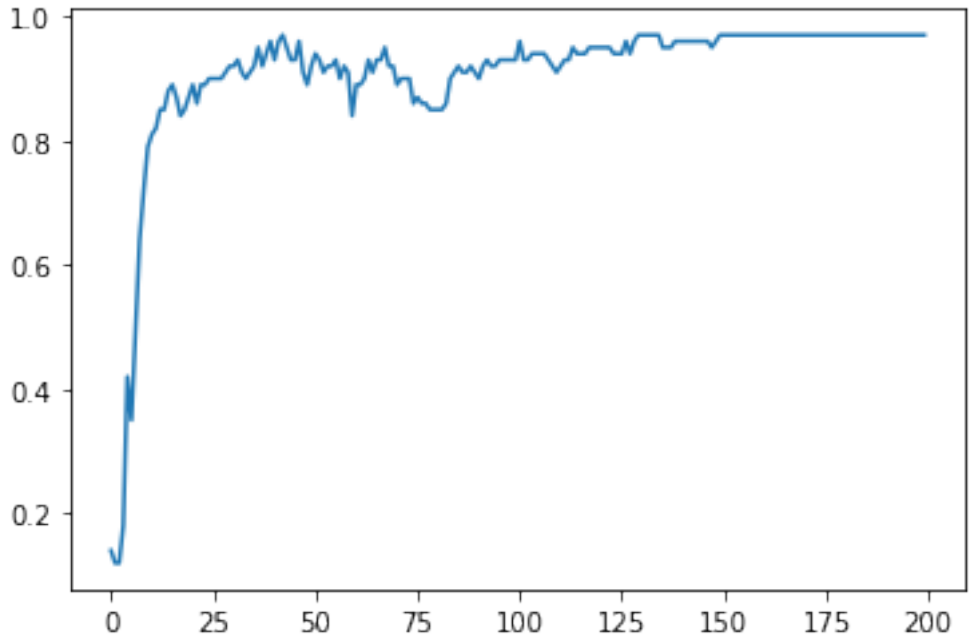
[8]:
```
#(g)
#Eigenface Feature.
#Write a function to generate r-dimensional feature matrix F and Ftest for␣
 ↪training images X and test images Xtest, respectively
#(to get F, multiply X to the transpose of first r rows of VT
#F should have same number of rows as X and r columns

def train_featurematrix_generator(r):
#multiply X to the transpose of first r rows of VT
    F = np.dot(sub_avg_face, vt[:r,:].T)
    return F

def test_featurematrix_generator(r):
#multiply X to the transpose of first r rows of VT
    Ftest = np.dot(sub_avg_face_test, vt[:r,:].T)
    return Ftest

F = train_featurematrix_generator(10)
Ftest = test_featurematrix_generator(10)
print(F.shape)
print(Ftest.shape)
```

```
(540, 10)
(100, 10)
```

[9]:
```
from sklearn.linear_model import LogisticRegression
```

[10]:
```
# import warnings filter
import warnings
# ignore all future warnings
warnings.simplefilter('ignore')
```

[11]:
```
#(h)
#Face Recognition.
#Extract training and test features for r = 10.
#Train a Logistic Regression model using F and test on Ftest.
#Report the classification accuracy on the test set.
#Plot the classification accuracy on the test set as a function of r when r =␣
 ↪1, 2,..., 200.
accuracy = []
logreg = LogisticRegression(multi_class='ovr')

for r in range(1, 201):
    F = train_featurematrix_generator(r)
    Ftest = test_featurematrix_generator(r)
    model = logreg.fit(F, train_labels)
    pred = model.predict(Ftest)
    score = model.score(Ftest, test_labels)
    accuracy.append(score)

plt.plot(accuracy)
plt.show()
```



[13]:
```
F = train_featurematrix_generator(10)
Ftest = test_featurematrix_generator(10)
clf = LogisticRegression().fit(F, train_labels)
pred = clf.predict(Ftest)
score = clf.score(Ftest, test_labels)
print(score)
```

```
0.79
```

[ ]:
```

```

In [2]:
```
import numpy as np
import json
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
```

In [3]:
```
# loading data from training data
f = open('./whats-cooking/train.json','r')
train_data = json.loads(f.read())
f.close()
```

In [57]:
```
# loading data from testing data
f = open('./whats-cooking/test.json','r')
test_data = json.loads(f.read())
f.close()
```

In [17]:
```
#(b) count
dish_count = len(train_data)
cuisine = []
uniq_ingredients = []

for d in train_data:
    cuisine.append(d['cuisine'])
    uniq_ingredients += d['ingredients']

uniq_ingredients = np.unique(uniq_ingredients)
cuisine = np.unique(cuisine)
cuisine_count = len(cuisine)
uniq_ingredients_count = len(uniq_ingredients)

print("dish count: {}".format(dish_count))
print("unique cuisines: {}".format(cuisine_count))
print("unique ingredients: {}".format(uniq_ingredients_count))
```

```
dish count: 39774
unique cuisines: 20
unique ingredients: 6714
```

In [38]:
```
#(c) Represent each dish by a binary ingredient feature vector.
#https://stackoverflow.com/questions/45312377/how-to-one-hot-encode-from
-a-pandas-column-containing-a-list
from sklearn.preprocessing import MultiLabelBinarizer
train_data_df = pd.DataFrame(train_data)
mlb = MultiLabelBinarizer()
binary_train_data_df = train_data_df.join(pd.DataFrame(mlb.fit_transform
(train_data_df.pop('ingredients')),
                          columns=mlb.classes_,
                          index=train_data_df.index))
binary_train_data_df
```

Out[38]:

	id	cuisine	(oz.) tomato sauce	(oz.) tomato paste	(10 oz.) frozen chopped spinach	(10 oz.) frozen chopped spinach, thawed and squeezed dry	(14 oz.) sweetened condensed milk	(14.5 oz.) diced tomatoes	(15 oz.) refried beans
0	10259	greek	0	0	0	0	0	0	0
1	25693	southern_us	0	0	0	0	0	0	0
2	20130	filipino	0	0	0	0	0	0	0
3	22213	indian	0	0	0	0	0	0	0
4	13162	indian	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
39769	29109	irish	0	0	0	0	0	0	0
39770	11462	italian	0	0	0	0	0	0	0
39771	2238	irish	0	0	0	0	0	0	0
39772	41882	chinese	0	0	0	0	0	0	0
39773	2362	mexican	0	0	0	0	0	0	0

39774 rows × 6716 columns

In [39]:
```
# convert dataframe to nparray
train_labels = df['cuisine'].to_numpy()
binary_train_data_df.drop(['id','cuisine'], axis=1, inplace=True)
binary_train_data = binary_train_data_df.to_numpy()
binary_train_data
```

Out[39]:
```
array([[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]])
```

In [40]:
```
#print(binary_train_data.shape)
```

```
(39774, 6714)
```

In [46]:
```
#(d) Using Naïve Bayes Classifier to perform 3 fold cross-validation on
 the training set
# Report your average classification accuracy.
# Try both Gaussian distribution prior assumption and Bernoulli distribu
tion prior assumption.
gaussian = GaussianNB()
bernoulli = BernoulliNB()
Gaussian_accuracy = []
Bernoulli_accuracy = []
kf = KFold(n_splits=3)
for train_index, test_index in kf.split(binary_train_data):
    trainx = binary_train_data[train_index]
    testx = binary_train_data[test_index]
    trainy = train_labels[train_index]
    testy = train_labels[test_index]
    gaussian.fit(trainx, trainy)
    score_gaussian = gaussian.score(testx, testy)
    Gaussian_accuracy.append(score_gaussian)
    avg_score_g = np.average(Gaussian_accuracy)
    bernoulli.fit(trainx, trainy)
    score_bernoulli = bernoulli.score(testx, testy)
    Bernoulli_accuracy.append(score_bernoulli)
    avg_score_b = np.average(Bernoulli_accuracy)
    print("Average accuracy of Gaussian:", avg_score_g)
    print("Average accuracy of Bernoulli:", avg_score_b)
```

```
Average accuracy of Gaussian: 0.37901644290239855
Average accuracy of Bernoulli: 0.684190677326897
Average accuracy of Gaussian: 0.3809775230049781
Average accuracy of Bernoulli: 0.6818524664353598
Average accuracy of Gaussian: 0.3798461306381053
Average accuracy of Bernoulli: 0.6835369839593705
```

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

Average accuracy of Gaussian prior is aroung 0.38 and average accuracy of Bernoulli is around 0.68. Bernoulli performs better in terms of cross-validation accuracy. The ingredients have two features which are represented by 0 or 1 (i.e. existing or non-existing), which is better described by a Bernoulli model. Thus, Bernoulli distribution prior assumption is better.

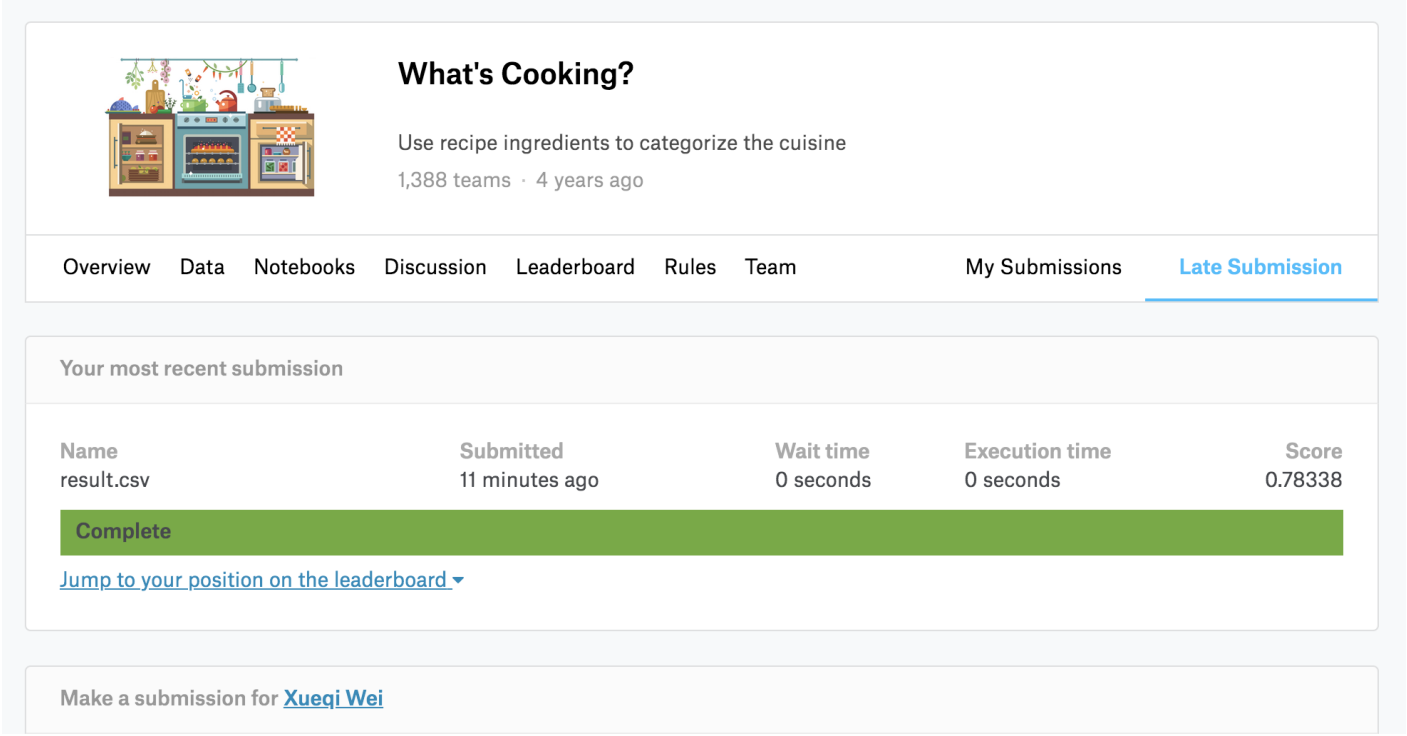
In [48]:
```
# (f) Using Logistic Regression Model to perform 3 fold cross-validation
on the training set
# Report your average classification accuracy

# import warnings filter
import warnings
# ignore all future warnings
warnings.simplefilter('ignore')

logreg = LogisticRegression()
logreg_accuracy = []
for train_index, test_index in kf.split(binary_train_data):
    trainx = binary_train_data[train_index]
    testx = binary_train_data[test_index]
    trainy = train_labels[train_index]
    testy = train_labels[test_index]
    logreg.fit(trainx, trainy)
    score_logreg = logreg.score(testx, testy)
    logreg_accuracy.append(score_logreg)
    avg_score_l = np.average(logreg_accuracy)
    print("Average accuracy of LogisticRegression:", avg_score_l)
```

```
Average accuracy of LogisticRegression: 0.7758334590435964
Average accuracy of LogisticRegression: 0.773985518177704
Average accuracy of LogisticRegression: 0.7755568964650275
```

In [67]:
```
# (g) Train your best-performed classifier with all of the training dat
a, and generate test labels on
# test set. Submit your results to Kaggle and report the accuracy.

# Average accuracy of LogisticRegression is about 0.77, which is the bes
t-performed classifier
# train LogisticRegression with all of the training data

# prepare test data
binary_test_data = np.zeros([len(test_data), uniq_ingredients_count])

for i, dish in enumerate(test_data):
    for j in dish['ingredients']:
        if j in uniq_ingredients:
            binary_test_data[i][np.where(uniq_ingredients == j)] = 1
```

In [74]:
```
#train model
logreg.fit(binary_train_data, train_labels)
pred = logreg.predict(binary_test_data)
```

In [75]:
```
#output
df = pd.DataFrame(data = {"id" : test_id, "cuisine" : pred})
df.to_csv(path_or_buf="result.csv", index=False)
```



The accuracy on Kaggle after submission is 0.78338.

HW2 write up

Q1. $\max a^T B a$

s.t. $a^T W a = 1$ by transforming to a standard eigenvalue problem

Lagrange function:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x)$$

in order to find the stationary points of of a function $f(x)$ subject to the equality constraint $g(x) = 0$,

Since $a^T W a = 1$. Let $g(x) = a^T W a - 1 = 0$.

$$\mathcal{L}(\lambda) = a^T B a + \lambda (a^T W a - 1)$$

$$\Rightarrow \frac{d\mathcal{L}(\lambda)}{da} = (B + B^T) a - \lambda (W + W^T) a = 0$$

$$(B + B^T) a = \lambda (W + W^T) a$$

$$(W + W^T)^{-1} (B + B^T) a = \lambda a$$

$\Rightarrow$ the problem is transfered to a standard eigenvalue problem

Q2. $X \in \mathbb{R}^p$, two-class response, class sizes N_1, N_2

targeted coded as $-\frac{N}{N_1}, \frac{N}{N_2}$

(a) $\pi_1 = -\frac{N}{N_1}$ $\pi_2 = \frac{N}{N_2}$

LDA: $\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$

class 1: $\log \frac{Pr(G=1|X=x)}{Pr(G=2|X=x)} = \log \frac{f_1(x)}{f_2(x)} + \log \frac{\pi_1}{\pi_2}$

class 2: $\log \frac{Pr(G=2|X=x)}{Pr(G=1|X=x)} = -\log \frac{f_1(x)}{f_2(x)} - \log \frac{\pi_1}{\pi_2}$

$f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}$

plug in

for class 2: If LDA rules classfies to class 2

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

should be greater than 0

$$\Rightarrow x^T \Sigma^{-1}(\mu_2-\mu_1) > \log \frac{\pi_1}{\pi_2} + \frac{1}{2}(\mu_1+\mu_2)^T \Sigma^{-1}(\mu_2-\mu_1)$$

$$= -\log \frac{N_2}{N_1} + \frac{1}{2}(\mu_1+\mu_2)^T \Sigma^{-1}(\mu_2-\mu_1)$$

Same for class 1:

$$x^T \Sigma^{-1}(\mu_2-\mu_1) < -\log \frac{N_2}{N_1} + \frac{1}{2}(\mu_1+\mu_2)^T \Sigma^{-1}(\mu_2-\mu_1)$$

(b) minimization of the least square criterion

$$RSS(\beta') = \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2$$

Let $U_i \in \mathbb{R}^n$ be the class indicator vector of class i

if $x_j \in$ class i. $j \in [0, N]$, $\Rightarrow U_{ij} = 1$ $U_{ij} = 0$ o.w.

$U = U_1 + U_2$ will be a vector of ones.

$a_1 = -\frac{N}{N_1}$ $a_2 = \frac{N}{N_2}$ $Y = a_1 u_1 + a_2 u_2$ vector of labels

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

$$= (Y - \beta_0 u - X\beta)^T (Y - \beta_0 u - X\beta)$$

$$\frac{\partial RSS}{\partial \beta} = 2X^T X\beta - 2X^T Y + 2\beta_0 X^T u = 0 \quad (1)$$

$$\frac{\partial RSS}{\partial \beta_0} = 2u^T u \beta_0 - 2u^T (Y - X\beta) = 2N\beta_0 - 2u^T (Y - X\beta) = 0 \quad (2)$$

$$\beta_0 = \frac{1}{N} u^T (Y - X\beta)$$

plug in β_0 to (1)

$$X^T X\beta - X^T Y + \frac{1}{N} u^T u (Y - X\beta) X^T = 0$$

$$X^T X\beta - X^T Y + \frac{1}{N} u^T u X^T Y - \frac{1}{N} u^T u X X^T \beta = 0$$

$$\boxed{(X^T X - \frac{1}{N} X^T X u^T u)\beta} = X^T Y - \frac{1}{N} X^T u^T u Y \quad (3)$$

LHS

$$X^T u = X^T (u_1 + u_2) = N_1 \hat{\mu}_1 + N_2 \hat{\mu}_2$$

$$(X^T X - \frac{1}{N} (N_1^2 \hat{\mu}_1 \hat{\mu}_1^T + N_2^2 \hat{\mu}_2 \hat{\mu}_2^T + N_1 N_2 \hat{\mu}_1 \hat{\mu}_2^T + N_1 N_2 \hat{\mu}_2 \hat{\mu}_1^T))\beta$$

Def of LDA

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

$$\sum_{g_i = k} x_i = N_k \hat{\mu}_k$$

↙ Sum of all variables ← count × mean ↘

$$\sum_{k=1}^{2} \sum_{g_i = k} (x_i^2) = X^T X$$

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

$$= \sum_{k=1}^{2} \sum_{i:g_i=k} (x_i^2 - 2x_i\mu_k^T + \mu_k\mu_k^T)$$

$$= XX^T + \sum_{i:g_i=a_1} (-2x_i\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_1^T)$$

add $$+ \sum_{i:g_i=a_2} (-2x_i\hat{\mu}_2^T + \hat{\mu}_2\hat{\mu}_2^T)$$

$$= XX^T - 2N_1\hat{\mu}_1\hat{\mu}_1^T + N_1\hat{\mu}_1\hat{\mu}_1^T - 2N_2\hat{\mu}_2\hat{\mu}_2^T + N_2\hat{\mu}_2\hat{\mu}_2^T$$

$$= XX^T - N_1\hat{\mu}_1\hat{\mu}_1^T - N_2\hat{\mu}_2\hat{\mu}_2^T$$

$$\hat{\Sigma}_B = \frac{N_1N_2}{N^2}(\hat{\mu}_2-\hat{\mu}_1)(\hat{\mu}_2-\hat{\mu}_1)^T$$

$$N\hat{\Sigma}_B = \frac{N_1N_2}{N}(\hat{\mu}_2-\hat{\mu}_1)(\hat{\mu}_2-\hat{\mu}_1)^T$$

$$\beta((N-2)\hat{\Sigma} + N\hat{\Sigma}_B) = \beta\Big(X^TX + \Big(\frac{N_1N_2}{N} - N_1\Big)\hat{\mu}_1\hat{\mu}_1^T + \Big(\frac{N_1N_2}{N} - N_2\Big)\hat{\mu}_2\hat{\mu}_2^T - \frac{N_1N_2}{N}(\hat{\mu}_2\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_2^T)\Big)$$

$$N = N_1 + N_2 \rightarrow = \Big(XX^T - \frac{1}{N}(N_1^2\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T + N_1N_2\hat{\mu}_2\hat{\mu}_1^T + N_1N_2\hat{\mu}_1\hat{\mu}_2^T)\Big)$$

- RHS $U^TY = a_1N_1 + a_2N_2$

$$X^TU = N_1\hat{\mu}_1 + N_2\hat{\mu}_2$$

$$X^TY - \frac{1}{N}(U^TYX^TU) = X^T(a_1U_1 + a_2U_2) - \frac{1}{N}(a_1N_1 + a_2N_2)(N_1\hat{\mu}_1 + N_2\hat{\mu}_2)$$

$$= a_1N_1\hat{\mu}_1 + a_2N_2\hat{\mu}_2 - \frac{\hat{\mu}_1(a_1N_1^2 + a_2N_1N_2)}{N} - \frac{\hat{\mu}_2(a_1N_1N_2 + a_2N_2^2)}{N}$$

$$a_1N_1^2 = a_1N_1(N-N_2) = a_1N_1N - a_1N_1N_2$$

$$a_2N_2^2 = a_2N_2N - a_2N_2N_1$$

$$a_1N_1\hat{\mu}_1 + a_2N_2\hat{\mu}_2 - \frac{\hat{\mu}_1(a_1N_1^2 + a_2N_1N_2)}{N} - \frac{\hat{\mu}_2(a_1N_1N_2 + a_2N_2^2)}{N}$$

$$= \frac{\hat{\mu}_1(a_1N_1N - a_1N_1N + a_1N_1N_2 - a_2N_1N_2)}{N}$$

$$+ \frac{\hat{\mu}_2(a_2N_2N - a_1N_1N - a_2N_2N + a_2N_1N_2)}{N}$$

$$= \frac{N_1N_2}{N}(\hat{\mu}_1 - \hat{\mu}_2)(a_1 - a_2)$$

$a_1 = -\frac{N}{N_1}$ $a_2 = \frac{N}{N_2}$ $a_1 - a_2 = \frac{-N^2}{N_1N_2}$

$$= \frac{N_1N_2}{N}(\hat{\mu}_1 - \hat{\mu}_2)\left(-\frac{N^2}{N_1N_2}\right)$$

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

Thus. $(N-2)\hat{\Sigma} + N\hat{\Sigma}_B)\hat{\beta} = X^TY - \frac{1}{N}(U^TUX^TY)$

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

$$\hat{\Sigma}_B\beta = (\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T\beta$$

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

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

$\Rightarrow \hat{\beta} \propto (\hat{\mu}_2 - \hat{\mu}_1)$

(d) $\frac{N_1N_2}{N}(\hat{\mu}_1 - \hat{\mu}_2)(a_1 - a_2)$ hold for any encoding a_1, a_2, this result in c) hold for any coding of the two classes

(e) $a_1 = -\frac{N}{N_1}$ $a_2 = \frac{N}{N_2}$ $Y = a_1U_1 + a_2U_2$

$$\beta_0 = \frac{U^T(Y - X\beta)}{N}$$

$$= \frac{1}{N}U^T(a_1U_1 + a_2U_2 - X\beta)$$

plug in a_1, a_2 $= \frac{1}{N}U^T(-\frac{N}{N_1}U_1 + \frac{N}{N_2}U_2 - X\beta)$

$$= -\frac{U^TU_1}{N_1} + \frac{1}{N_2}U^TU_2 - \frac{1}{N}U^TX\beta$$

$$= -\frac{N_1}{N_1} + \frac{N_2}{N_2} - \frac{1}{N} u^T X \beta$$

$$= -\frac{1}{N} u^T X \beta$$

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

$$\hat{f}(x) = \hat{\beta}_0 + x^T \hat{\beta} = -\frac{1}{N}(N_1 \hat{\mu}_1^T + N_2 \hat{\mu}_2^T)\hat{\beta} + x^T \hat{\beta}$$

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

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

$$\hat{f}(x) > 0$$

$$x^T c \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) - \frac{N_1 \hat{\mu}_1^T}{N} c \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) - \frac{N_2 \hat{\mu}_2^T}{N} c \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > 0$$

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

$\Rightarrow$

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

if $\hat{f}(x) > 0$ classify to class 2 or class 1 o.w.

when $N_1 = N_2$.

this simplifies to the LDA decision f in (a)

Q3. (a)

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

$$= \begin{bmatrix} 39 & 57 & 60 \\ 57 & 118 & 53 \\ 60 & 53 & 127 \end{bmatrix}$$

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

$$= \begin{bmatrix} 10 & 9 & 26 & 3 & 26 \\ 9 & 62 & 8 & -5 & 85 \\ 26 & 8 & 72 & 10 & 50 \\ 3 & -5 & 10 & 2 & -1 \\ 26 & 85 & 50 & -1 & 138 \end{bmatrix}$$

(b) Eigenvalue of M^TM

$$\det(M - \lambda I) = \det \begin{bmatrix} 39-\lambda & 57 & 60 \\ 57 & 118-\lambda & 53 \\ 60 & 53 & 127-\lambda \end{bmatrix} = 0$$

$$(39-\lambda)\left[(118-\lambda)(127-\lambda) - 53^2\right] - 57\left((57 \times (127-\lambda)) - 60 \times 53\right)$$
$$+ 60\left[57 \times 53 - 60 \times (118-\lambda)\right] = 0$$

$(39-\lambda)(14986-245\lambda+\lambda^2-2809)-57(7239-57\lambda-3180)$
$+60(3021-7080+60\lambda)=0$
$\lambda = 214.6705,\ 69.3295$

Eigenvalue of MM^T

$$\det(M-\lambda I)=\det\begin{bmatrix}10-\lambda & 9 & 26 & 3 & 26\\ 9 & 62-\lambda & 8 & -5 & 85\\ 26 & 8 & 72-\lambda & 10 & 50\\ 3 & -5 & 10 & 2-\lambda & -1\\ 26 & 85 & 50 & -1 & 138-\lambda\end{bmatrix}=0$$

$\lambda = 214.6705,\ 69.3295$

(c) Eigenvector of M^TM:

$$\begin{bmatrix}0.4261\\ 0.6150\\ 0.6634\end{bmatrix}\qquad\begin{bmatrix}-0.0146\\ -0.7285\\ 0.6847\end{bmatrix}$$

Eigenvector of MM^T

$$\begin{bmatrix}-0.1649\\ -0.4716\\ -0.3364\\ -0.0033\\ -0.7982\end{bmatrix}\qquad\begin{bmatrix}0.2449\\ -0.4533\\ 0.8294\\ 0.1697\\ -0.1331\end{bmatrix}$$

(d) Find SVD $\quad A = U\Sigma V^T$

Singular values: $\sqrt{69.3295} = 8.3264\quad\sqrt{214.6705} = 14.6516$

$$\Sigma = \begin{bmatrix}14.6516 & 0\\ 0 & 8.3264\end{bmatrix}$$

$MM^T \Rightarrow$ eigenvector $\Rightarrow V$

$$V = \begin{bmatrix}0.4261 & -0.0146\\ \end{bmatrix}$$

$$V = \begin{bmatrix} 0.6150 & -0.1285 \\ 0.6634 & 0.8847 \end{bmatrix}$$

$M^TM \Rightarrow$ eigenvector $\Rightarrow U$

$$U = \begin{bmatrix} -0.1649 & 0.2449 \\ -0.4716 & -0.4533 \\ -0.3364 & 0.8294 \\ -0.0033 & 0.1697 \\ -0.7982 & -0.1331 \end{bmatrix}$$

(2) $\Sigma' = \begin{bmatrix} 14.6516 & 0 \\ 0 & 0 \end{bmatrix}$

$$U\Sigma' V^T = \begin{bmatrix} -0.1649 & 0.2449 \\ -0.4716 & -0.4533 \\ -0.3364 & 0.8294 \\ -0.0033 & 0.1697 \\ -0.7982 & -0.1331 \end{bmatrix} \begin{bmatrix} 14.6516 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.6150 & 0.6634 \\ -0.1285 & 0.8847 \end{bmatrix}$$

$$= \begin{bmatrix} -1.0294 & -1.4858 & -1.6028 \\ -2.9442 & -4.2494 & -4.5838 \\ -2.1001 & -3.0312 & -3.2697 \\ -0.0206 & -0.0297 & -0.0320 \\ -4.9832 & -7.1933 & -7.7584 \end{bmatrix}$$