Applied Machine Learning HW2

1 Face Recognition

1. Read train data and test data. Display one training sample.

   We first load the data with the help of the code in homework 2. Several libraries are also imported for future use. One training example is displayed.

```
#1.(b) load the data
import numpy as np
from scipy import misc
from matplotlib import pylab as plt
import matplotlib.cm as cm
%matplotlib inline

# load train data
train_labels, train_data = [], []
for line in open('/Users/yingzhu/Desktop/faces/train.txt'):
    readline = line.strip().split()[0]
    im = misc.imread('/Users/yingzhu/Desktop'+readline[1:])
    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)

# load test data
test_labels, test_data = [], []
for line in open('/Users/yingzhu/Desktop/faces/test.txt'):
    readline = line.strip().split()[0]
    im = misc.imread('/Users/yingzhu/Desktop'+readline[1:])
    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)

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

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

Figure 1:

2. Average Face

   We calculated the mean of the whole training set by using the mean function with axis equals 0. The following is the plot of average face.

```
# 1.(c) AVERAG Face
# Average face for train
Average_face_train = np.mean(train_data,axis=0)
Average_reshape_train=Average_face_train.reshape(50,50)
plt.imshow(Average_reshape_train,cmap=cm.Greys_r)
plt.show()
```

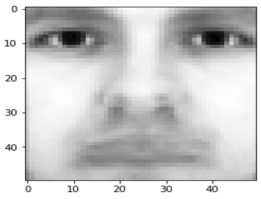

Figure 2:

3. Mean Subtraction

   We subtract the mean from every column in X. And also subtract the training set mean from the test set. The following two graphs are the 10th sample from both training set and test set.

```
# 1.(d) Mean Substraction
# mean subtraction for train data
X_subtract_train= train_data-Average_face_train
plt.figure(figsize=(10,6),dpi=50)
plt.subplot(1,2,1)
plt.imshow(X_subtract_train[10,:].reshape(50,50),cmap=cm.Greys_r)

# mean subtraction for test data
X_subtract_test= test_data-Average_face_train
plt.subplot(1,2,2)
plt.imshow(X_subtract_test[10,:].reshape(50,50),cmap=cm.Greys_r)
plt.show()
```

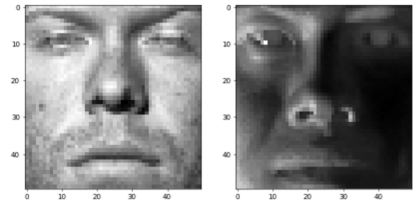

Figure 3:

4. Eigenface

   We apply singular value decomposition by using function *np.linalg.svd*. The following are the 10 eigenfaces of mean subtracted training set.

```
# 1.(e) Eigenface
U, s, V = np.linalg.svd(X_subtract_train, full_matrices=True)
plt.figure(figsize=(10, 6), dpi=80)
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.imshow(V[i].reshape(50,50),cmap=cm.Greys_r)
```

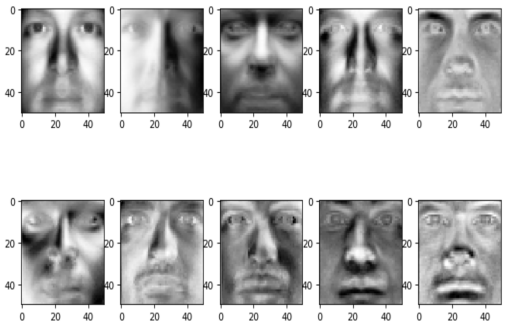

Figure 4:

5. Low Rank Approximation

We defined function called rankapprox calculate $U[:, : r] * \sum[: r, : r] * V_T[: , : r]$. Function approx error is used to calculate the rank-r approximation error of absolute value of $X - \hat{X}_r$. After we run r from 1,2,3....200, we plot the approximation error with r.In the following graph, the x axis is the value of r and the y axis is the approximation error.

```
# 1.(f) Low_rank Approximation
S=np.diag(s)
```

```
def rank_approx(U,S,V,r):
    X_test = np.dot(np.dot(U[:,:r],S[:r,:r]),V[:r,:])
    return X_test
```

```
def approx_error(index,index_array,X):
    error=[]
    U, s, V = np.linalg.svd(X, full_matrices=True)
    S=np.diag(s)
    for i in range(1,index):
        index_array.append(i)
        X_test = rank_approx(U,S,V,i)
        error.append(np.linalg.norm(X_test-X))
    return error
```

```
Index = []
Error = approx_error(201,Index,X_subtract_train)
```

```
plt.plot(Index,Error)
plt.show()
```

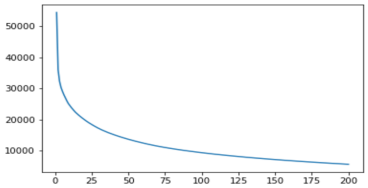

Figure 5:

From the graph, we could observe that with the increase of r, the approximation error drops with a decreasing speed.With smaller r, the approximation error drops faster.

6. Eigenface Feature
   We write the function F to generate the r-dimensional feature matrix F and F_{test} for both training images X and test images X_{test}. Both F and F_{test} have the same number of rows as X and X_{test} and r columns.(we test r=10 in this following code)

```
# 1.(g) Eigenface Feature
def F(X,r):
    U, s, V = np.linalg.svd(X_subtract_train, full_matrices=True)
    f=np.dot(X,np.transpose(V[:r,:]))
    return f
```

```
F_train = F(X_subtract_train,10)
print(F_train.shape,X_subtract_train.shape)
F_test =F(X_subtract_test,10)
print(F_test.shape,X_subtract_test.shape)
```

```
(540, 10) (540, 2500)
(100, 10) (100, 2500)
```

Figure 6:

7. Face recognition

   In this section, we extract training and test features for r=10 and train the logistic regression model using F and F_{test}. The classification accuracy on the test set is stable around 0.8. The following plot is the classification accuracy on the test set as a function of r when r=1,2,3.....,200. We found that after r=25, we already have a pretty low classification error.The approximation accuracy is stable around 0.9 after r = 25. This means that we could reduce the 2500 dimension feature into 25 dimensional feature with a acceptable classification error.

```
# 1.(h) Face Recognition
from sklearn.linear_model import LogisticRegression
ls = LogisticRegression(multi_class='ovr')
ls.fit(F_train,train_labels)
print(ls.score(F_test,test_labels))
```

```
0.79
```

```
r=[]
ls_score=[]
U, s, V = np.linalg.svd(X_subtract_train, full_matrices=True)
for i in range(1,201):
    r.append(i)
    F_train = np.dot(X_subtract_train,np.transpose(V[:i,:]))
    F_test =np.dot(X_subtract_test,np.transpose(V[:i,:]))
    ls.fit(F_train,train_labels)
    ls_score.append(ls.score(F_test,test_labels))
plt.plot(r,ls_score)
plt.show()
```

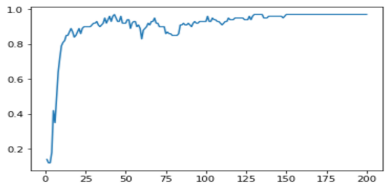

Figure 7:

2 Cooking

1. Load the data

```
# 1.(a) load the data
import json
import pandas as pd
with open('/Users/yingzhu/Desktop/train.json') as json_train:
    train_data = json.load(json_train)
with open('/Users/yingzhu/Desktop/test.json') as json_test:
    test_data = json.load(json_test)
```

```
import numpy as np
train = pd.DataFrame(train_data)
test = pd.DataFrame(test_data)
print(train.shape)
print(test.shape)
```

```
(39774, 3)
(9944, 2)
```

Figure 8:

2. Description of the data.
   There are 39774 samples in the training set. There are total 20 categories. As what we displayed in the (c) part, there are 6714 unique ingredients. And list of unique ingredients called ingredientslist is created and listed in the (c) part.

```
# 1.(b) How many categories(types of cuisine)?
type_cuisine = train['cuisine'].value_counts()
print(type_cuisine)
print(len(type_cuisine))
```

```
italian          7838
mexican          6438
southern_us      4320
indian           3003
chinese          2673
french           2646
cajun_creole     1546
thai             1539
japanese         1423
greek            1175
spanish           989
korean            830
vietnamese        825
moroccan          821
british           804
filipino          755
irish             667
jamaican          526
russian           489
brazilian         467
Name: cuisine, dtype: int64
20
```

Figure 9:

3. Represent each dish by a binary ingredients 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 and d is the total number of unique ingredients.[1]

   The following is the display of the training data binary ingredients feature vectors. d equals 6714 in this problem. We also fit the test data into the n* d binary ingredients feature vector using for loop.

```
# 1.(c) Represent each dish by a binary ingredient feature vector
ingredient_matrix = train['ingredients'].str.join(sep='*').str.get_dummies(sep='*')
```

```
# citation for ingredient_matrix:
# https://stackoverflow.com/questions/18889588/create-dummies-from-column-with-multiple-values-in-pandas

ingredient_list = list(ingredient_matrix.columns.values)
print(ingredient_matrix.shape)
ingredient_matrix.head(n=10)
```

```
(39774, 6714)
```

	(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	1% low-fat buttermilk	1% low-fat chocolate milk	1% low-fat cottage cheese	...	yukon gold potatoes	yuzu	yuzu juice	za'atar	zest	zesty italian dressing	zinfa
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	0	0	0	0	0	0	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	
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	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

10 rows × 6714 columns

Figure 10:

[1] https://stackoverflow.com/questions/18889588/create-dummies-from-column-with-multiple-values-in-pandas

```
ingredient_matrix_test = np.zeros((len(test),len(ingredient_list)),dtype=np.int)

for i in range(0,len(test)):
    for j in range(0,len(test.values[i][1])):
        if test.values[i][1][j] in ingredient_list:
            pos = ingredient_list.index(test.values[i][1][j])
            ingredient_matrix_test[i][pos]=1
```

```
print(ingredient_matrix_test.shape)
ingredient_matrix_test[:10]
```

```
(9944, 6714)

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]])
```

Figure 11:

4. Using Naive Bayes Classification 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.

   First, we tried the Gaussian distribution prior assumption. The accuracy by 3 fold cross validation is around 0.38.

```
# 1.(d) Using Naive Bayes Classifier to perfroma 3 fold cross-validation on the training set
# report average classification accuracy
from sklearn import cross_validation
from sklearn.naive_bayes import GaussianNB,BernoulliNB
clf_G = GaussianNB()
clf_B = BernoulliNB()
#(a) Gaussian distribution\\

from sklearn.cross_validation import KFold
def evaluate(X,Y):
    kf =KFold(len(Y), n_folds=3)
    accu=0.0
    sub_accu_list=[]
    for train_indices, test_indices in kf:
        X_train = X[train_indices,:]
        Y_train = Y[train_indices]
        X_test = X[test_indices,:]
        Y_test = Y[test_indices]
        subaccu = clf_G.fit(X_train,Y_train).score(X_test,Y_test)
        sub_accu_list.append(subaccu)
        accu+=subaccu
    print(sub_accu_list)
    return accu/3.0
evaluate(ingredient_matrix.values,train['cuisine'])
```

```
[0.37901644290239855, 0.38293860310755767, 0.37758334590435966]

0.3798461306381052 9
```

Figure 12:

Secondly, we tries the Bernoulli distribution prior assumption.The accuracy by 3 fold cross validation is around 0.68.

```
#(b) Bernoulli distribution
def evaluateB(X,Y):
    kf =KFold(len(Y), n_folds=3)
    accu=0.0
    sub_accu_list=[]
    for train_indices, test_indices in kf:
        X_train = X[train_indices,:]
        Y_train = Y[train_indices]
        X_test = X[test_indices,:]
        Y_test = Y[test_indices]
        subaccu = clf_B.fit(X_train,Y_train).score(X_test,Y_test)
        sub_accu_list.append(subaccu)
        accu+=subaccu
    print(sub_accu_list)
    return accu/3.0
evaluateB(ingredient_matrix.values,train['cuisine'])
```

```
[0.68419067732689698, 0.67951425554382261, 0.68690601900739179]

0.68353698395937046
```

Figure 13:

5. For Gaussian prior and Bernoulli prior, which performs better in terms of cross-validation accuracy? Why? Please give specific accuracy.

   Compared Gaussian prior with Bernoulli prior, we found that Bernoulli prior performs better with accuracy of around 0.68. The reason of this outcome is that we represent each dish by a binary ingredients feature vector. There are only two possible value for each ingredient, either 0 or 1. Apparently assuming the feature follows a Bernoulli distribution is more reasonable than assuming they follows Gaussian distribution. So compared with Gaussian distribution, Bernoulli distribution fits better.

6. Using logistic regression model to perform 3 fold cross validation on the training set and report you average classification accuracy.

   After we run the logistic regression, we get the average classification accuracy as around 0.78. The accuracy of logistic regression is higher than Naive Bayes' result. This is reasonable. Because in Naive Bayes, we assume that each feature is independent to each other. But for each dish, the correlation of each ingredient should be very high. This high correlation is omitted by the Naive Bayes.

```
# 1. (f) Using logistic regression performa 3 fold cross validation on training
# average classification accuracy
from sklearn.linear_model import LogisticRegression
ls = LogisticRegression()
def evaluateLogistic(X,Y):
    kf =KFold(len(Y), n_folds=3)
    accu=0.0
    sub_accu_list=[]
    for train_indices, test_indices in kf:
        X_train = X[train_indices,:]
        Y_train = Y[train_indices]
        X_test = X[test_indices,:]
        Y_test = Y[test_indices]
        subaccu = ls.fit(X_train,Y_train).score(X_test,Y_test)
        sub_accu_list.append(subaccu)
        accu+=subaccu
    print(sub_accu_list)
    return accu/3.0
evaluateLogistic(ingredient_matrix.values,train['cuisine'])
```

```
[0.77583345904359635, 0.77213757731181176, 0.77869965303967414]

0.77555689646502746
```

Figure 14:

7. Train your best-performed classification with all the training data and generate test labels on test set. Submit your results to Kaggle and report the accuracy.

   After we train Logistic regression with all the training data and generate test labels on test set. We got accuracy of around 0.78 in Kaggle.

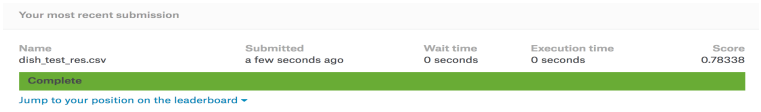

Figure 15: