**“CS5785 Homework 2”**

**Submitted as part of Homework**

**required for**

**Applied Machine Learning (CS5785)**

**SUBMITTED BY**

TANUJ AHUJA – ta364

JAMIE YU – jky32

**SUBMITTED TO**

***Cornell Tech, Cornell University***

**2 W Loop Road,**

**New York, NY 10044**

**PROGRAMMING EXCERCISES**

**Q1.** Eigenface for face recognition

**Q1(a).** Download The Face Dataset

**Ans1(a).** Downloaded

**Q1(b).** Load the training and testing set into a matrix X. Pick a face image from X and display the image in grayscale

**Ans1(b). CODE**

#Training Data

import numpy as np

from scipy import misc

from matplotlib import pylab as plt

import matplotlib.cm as cm

%matplotlib inline

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.title('Training Data: 10th image')

plt.show()

#Testing Data

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.title('Testing Data: 10th image')

plt.show()

**OUTPUT**



**Q1(c).** Average Face. Compute the *average face* μ from the whole training set by summing up every column in **X** then dividing by the number of faces. Display the *average face* as a grayscale image.

**Ans1(c). CODE**

#sum every column in training data and divide it by the number of faces

average\_values = np.sum(train\_data, axis=0)/train\_data.shape[0]

# display average face as a grayscale image

plt.imshow(average\_values.reshape(50,50), cmap=cm.Greys\_r)

plt.title('Training Data: Average Face')

plt.show()

**OUTPUT**



**Q1(d).** Mean Subtraction. Subtract average face μ from every column in **X**. That is, **xi** := **xi** − μ, where **xi** is the *i*-th column of **X**. Pick a face image after mean subtraction from the new **X** and display that image in grayscale. Do the same thing for the test set **Xtest** using the pre- computed average face μ in (c).

**Ans1(d). CODE**

mean\_subtracted\_train\_data = train\_data - average\_values

mean\_subtracted\_test\_data = test\_data - average\_values

# display average face as a grayscale image

plt.imshow(mean\_subtracted\_train\_data[26].reshape(50,50), cmap=cm.Greys\_r)

plt.title('Training Data: 26th image - Mean Subtracted Face')

plt.show()

plt.imshow(mean\_subtracted\_test\_data[26].reshape(50,50), cmap=cm.Greys\_r)

plt.title('Testing Data: 26th image - Mean Subtracted Face')

plt.show()

**OUTPUT**





**Q1(e).** Eigenface. Perform Singular Value Decomposition (SVD) on training set **X** (**X** = **U**Σ**V***T* ) to get matrix **V***T* , where each row of **V***T* has the same dimension as the face image. 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.

**Ans1(e). CODE**

U, s, V = np.linalg.svd(mean\_subtracted\_train\_data)

for i in range(10):

plt.figure()

plt.imshow(V[i].reshape(50,50), cmap=cm.Greys\_r)

plt.title('Eigenface: %s'%i)

**OUTPUT**

















**Q1(f).** Low-rank Approximation. Since Σ is a diagonal matrix with non-negative real numbers on the diagonal in non-ascending order, we can use the first *r* elements in Σ together with first *r* columns in **U** and first *r* rows in **V***T* to approximate **X**. That is, we can approximate **X** by **X**ˆ**r** = **U**[:,: *r*] Σ[: *r*,: *r*] **V***T* [: *r*,:]. The matrix **X**ˆ**r** is called rank-*r* approximation of **X**. Plot the rank-*r* approximationerror∥**X**−**X**ˆ**r**∥*F* 2 asafunctionof*r* when*r* =1,2,...,200.

**Ans1(f). CODE**

**rankR\_approximation\_error = []**

for r in range(1,201):

XHatR = np.dot(np.dot(U[:,:r] , np.diag(s[:r])), V[:r,:])

X = mean\_subtracted\_train\_data

Frobenius\_norm= np.linalg.norm(X-XHatR)

rankR\_approximation\_error.append(Frobenius\_norm)

plt.plot(rankR\_approximation\_error)

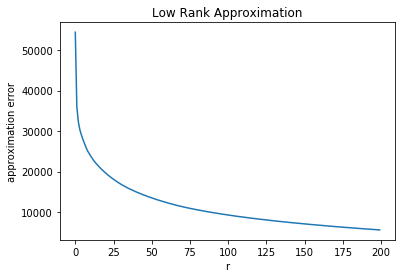
plt.title('Low Rank Approximation')

plt.xlabel('r')

plt.ylabel('approximation error')

plt.show()

**OUTPUT**



**Q1(g).** Eigenface Feature. The top *r* eigenfaces **V***T* [: *r*,:] = {*v*1,*v*2,...,*vr* }*T* span an *r*-dimensional linear subspace of the original image space called *face space*, whose origin is the average face μ, and whose axes are the eigenfaces {*v*1,*v*2,...,*vr*}. Therefore, using the top *r* eigenfaces {*v*1,*v*2,...,*vr*},wecanrepresenta2500-dimensionalfaceimage**z**asan*r*-dimensionalfeature vector **f**: **f** = **V***T* [: *r*,:] **z** = [*v*1,*v*2,...,*vr* ]*T* **z**. 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 **V***T* , **F** should have same number of rows as **X** and *r* columns; similarly for **Xtest**).

**Ans1(g). CODE**

**def generateFeatureMatrixF(X, V, r):**

**return np.dot(X,np.transpose((V)[:r,:]))**

**OUTPUT**

Combined with the output for 1(h)

**Q1(h).** Eigenface Feature. The top *r* eigenfaces **V***T* [: *r*,:] = {*v*1,*v*2,...,*vr* }*T* span an *r*-dimensional linear subspace of the original image space called *face space*, whose origin is the average face μ, and whose axes are the eigenfaces {*v*1,*v*2,...,*vr*}. Therefore, using the top *r* eigenfaces {*v*1,*v*2,...,*vr*},wecanrepresenta2500-dimensionalfaceimage**z**asan*r*-dimensionalfeature vector **f**: **f** = **V***T* [: *r*,:] **z** = [*v*1,*v*2,...,*vr* ]*T* **z**. 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 **V***T* , **F** should have same number of rows as **X** and *r* columns; similarly for **Xtest**).

**Ans1(h). CODE**

from sklearn import linear\_model

logistic\_regression = linear\_model.LogisticRegression()

accuracy = []

for r in range(1,201):

F = generateFeatureMatrixF(mean\_subtracted\_train\_data,V,r)

Ftest = generateFeatureMatrixF(mean\_subtracted\_test\_data,V,r)

#train logistic regression model using F; classifier is trained for each possible output label

logistic\_regression.fit(F,train\_labels)

#test model on Ftest

mean\_accuracy = logistic\_regression.score(Ftest,test\_labels)

accuracy.append(mean\_accuracy)

#Plot the classification accuracy as a function of r

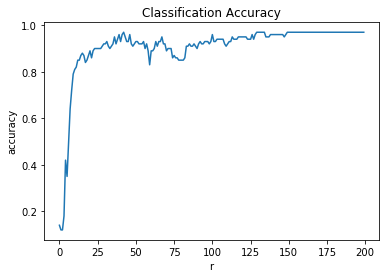
plt.plot(accuracy)

plt.title('Classification Accuracy')

plt.ylabel('accuracy')

plt.xlabel('r')

**OUTPUT**



**Question 1**

**WHAT WE DID**

This assignment makes use of python libraries **Scipy** and **Numpy** to load in the training*(‘./faces.train.txt’*) and testing data*(‘./faces/test.txt’*)**.** The *‘imread’* function in scipy is used to read the data; the data is cleaned of white spaces and the corresponding label to retrieve just the image path. The data is then appended into an array. The size of the training set is 540 x 2500 while the size of the testing set is 100 x 2500; each row is a flattened face image. A random row is selected from both the training and testing data and reshaped into a 50 x 50 matrix and plotted using the **matplotlib.plt** library.

The Average Face μ was calculated from the whole training set by summing up every column in the matrix X(entitled train\_data) using a the function *‘sum’* in the **numpy** library. This was then divided by the number of faces in the dataset which was calculated by taking the number of rows in the matrix X. The average face was then reshaped into a 50 x 50 matrix and plotted using the **matplotlib.plt** library.

The average face μ was subtracted from every column in X. This was done simply by taking advantage of the fact that the difference between two **numpy** arrays automatically subtracts each corresponding row. Thus subtracting the average face values calculated in 1(c) from the matrix X resulted in the mean subtracted version of X automatically because they are both **numpy** arrays.

The Singular Value Decomposition was performed on training set X to get three matrices: U, Sigma, and V. This was done through the function ‘*linalg.svd’* from the **numpy** library.

The low-rank approximation(error∥X−Xˆr∥F) was calculated as a function of r when r -1,2,…,200. Xˆr was calculated by multiplying all three matrices (U, Sigma, and V) when SVD was performed. The matrices were multiplied using the *‘dot’* function in the **numpy** library. The values of the Sigma matrix were given as one line when SVD was performed but this needed to be transformed to a diagonal matrix in order to be able to be multiplied correctly with the other matrices. Thus the *‘diag’* function in **numpy** was used to transform the single line of values into a matrix where the diagonal contained those values. Then the Frobenius norm was performed on (X−Xˆr) by using the *‘linalg.norm’* in the **numpy** library. The results of the Frobenius\_norm was plotted against r using the **matplotlib.plt** library.

A function to generate r-dimensional feature matrix F and Ftest for training images X and test images Xtest was written. To get F, X was multiplied to the transpose of first r rows of VT; the transpose of V was taken using the *‘transpose’* function in **numpy**. A logistic regression(taken from the **sklearn** library) was trained using the feature matrix generated from the training data(*‘fit’* function ) and then tested using the data from the feature matrix generated from the testing data(*‘score’* function). The mean accuracy gotten from the ‘score’ function is plotted against r=1,2,…200 using the **matplotlib.plt** library.

**INSIGHTS**

* In 1f, the Low-Rank Approximation error was plotted against r=1,2,…200. Results were as expected: at the beginning values of r the error was highest, and as r grew the approximation error exponentially decreased forming a downward curve with the concave closest to the origin. These results were as expected because the xlabel, r, selected the amount of the data that was represented in the matrices U, sigma, and V; in other words, the higher the r, the more data was available to calculate the rank-r approximation of X. From the results of the graph, as more data or rows were taken from the U, sigma, and V matrices the approximation error. This follows the intuitive idea whereby as we have more data, the chance for error exponentially decreases.

**PROGRAMMING EXCERCISES**

Q2. What’s Cooking?

1. Join the What’s Cooking competition on Kaggle. Download the training and test data (in .json). The competition page describes how these files are formatted.

Ans) Downloaded the train.csv and test.csv file from Kaggle.

1. 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?

Ans) Number of Dishes in Training Data – **39774**

Types of Cuisines – **20**

Number of Unique Ingredients in Training Data – **6714**

Number of Unique Ingredients in Testing and Training Data combined – **7137**

**CODE**

import csv

import json

import pandas as pd

import numpy as np

import sys

from sklearn.naive\_bayes import GaussianNB as nb

from sklearn.naive\_bayes import BernoulliNB as nb1

import pickle

%matplotlib inline

import matplotlib.pyplot as plt

#Loading data

train = pd.read\_json('train.json')

test = pd.read\_json('test.json')

#Number of Dishes in Training Data

len(train['id'].value\_counts())

#Number of Cuisines

cuisine = train['cuisine'].value\_counts()

len(cuisine)

ingredients = set()

cuisines = set()

train\_id = list()

test\_id = list()

tfile = open('train.json')

tdata = json.load(tfile)

tfile.close()

test = open('test.json')

testdata = json.load(test)

test.close()

#Segregating relevant data into correponding datasets.

for recipe in tdata:

train\_id.append(recipe['id'])

cuisines.add(recipe['cuisine'])

for elems in recipe['ingredients']:

ingredients.add(elems)

len(train\_id)

len(ingredients) #Number of Unique Ingredients in Training Data

1. Represent each dish by a binary ingredient feature vector. Suppose there are d different in- gredients in total from the training set, represent each dish by a 1 × d binary ingredient vector x , where x i = 1 if the dish contains ingredient i and x i = 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.

Ans) **CODE**

#Represent Each Dish by a binary ingredient feature vector

cuisines = list(cuisines)

cuisines = dict([(cuisine, i) for i, cuisine in enumerate(cuisines)])

ingredients = list(ingredients)

ingredients = dict([(ingredient, i) for i, ingredient in enumerate(ingredients)])

X = []

Y = []

for recipe in tdata:

inglist = [0 for x in range(len(ingredients))]

for ingredient in recipe['ingredients']:

pos = ingredients[ingredient]

inglist[pos] = 1.0

X.append(inglist)

cuisinepos = cuisines[recipe['cuisine']]

Y.append(cuisinepos)

X\_test = []

for recipe in testdata:

inglist = [0 for x in range(len(ingredients))]

for ingredient in recipe['ingredients']:

pos = ingredients[ingredient]

inglist[pos] = 1.0

X\_test.append(inglist)

#Binary Ingredient Feature Vector (n X d)

len(X)

len(X[0])

len(X\_test) #Binary Ingredient Feature Vector (n X d) in Testing Data

len(X\_test[0]) #Number of ingredients have increased from 6714 to 7137 because Test Data has more ingredients

1. 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.

Ans) **CODE**

#Setting up environment for 3-Fold Cross Validation

from sklearn import cross\_validation

all\_folds = cross\_validation.KFold(len(Xa), n\_folds=3)

print("Genrated Folds")

for i in all\_folds:

train,test=i

print("%s %s" % (train, test))

#Gaussian Prior Naive Bayes

from sklearn.naive\_bayes import GaussianNB

clf = GaussianNB()

result = []

for i in all\_folds:

train,test=i

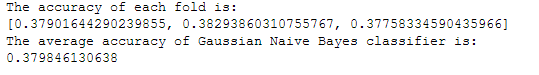
result.append(clf.fit(Xa[train],Ya[train]).score(Xa[test],Ya[test]))

print("The accuracy of each fold is: ")

print(result)

print("The average accuracy of Gaussian Naive Bayes classifier is: ")

print(sum(result) / float(len(result)))



#Bernoulli Prior Naive Bayes

from sklearn.naive\_bayes import BernoulliNB

clf = BernoulliNB()

result\_Bernoulli = []

for i in all\_folds:

train,test=i

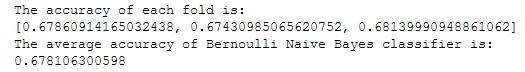
result\_Bernoulli.append(clf.fit(Xa[train],Ya[train]).score(Xa[test],Ya[test]))

print("The accuracy of each fold is: ")

print(result\_Bernoulli)

print("The average accuracy of Bernoulli Naive Bayes classifier is: ")

print(sum(result\_Bernoulli) / float(len(result\_Bernoulli)))



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

Ans) The Gaussian prior had an accuracy rate of 37.9% while the Bernoulli prior had an accuracy rate of 68.3%. The Gaussian Naive Bayes model is generally used for continuous data where the underlying data distribution is assumed to be a Gaussian (Normal) distribution and each feature is a real number. The Bernoulli Naive Bayes model describes whether or not an event occurred (for example whether a certain element exists in a dataset, not how many times it occurs). The dataset that the models were performed on was a simple dataset that recorded whether an 'ingredient' was present(appropriate for Bernoulli) rather than a continuous, discrete, and quantitative dataset(appropriate for Gaussian). Thus, the Bernoulli model was more accurate.

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

Ans) #Logistic Regression using 3-Fold Cross Validation

from sklearn import linear\_model

result\_Logistic = []

cls = linear\_model.LogisticRegression()

for i in all\_folds:

train,test=i

result\_Logistic.append(cls.fit(Xa[train], Ya[train]).score(Xa[test], Ya[test]))

print("The accuracy of each fold is: ")

print(result\_Logistic)

print("The average accuracy of Logistic Regression classifier is: ")

print(sum(result\_Logistic) / float(len(result)))

final\_result = []

cfr = linear\_model.LogisticRegression()

final\_result = cfr.fit(Xa,Ya).score(Xa,Ya)

print("The accuracy of Logistic Regression on entire training data is: ")

print(final\_result)



**INSIGHTS**

* **Test data had extra unique ingredients other than train data and feature engineering them increased our accuracy from 77 to 87 percent.**

**The train data had 100 redundant**

