CS464 Introduction to Machine Learning

Spring 2019

Homework 2

Contents

Ayhan Okuyan
21601531
CS464-1

PCA & Eigenfaces

1.1

PCA and SVD are in principal similar eigenvector/eigenvalue centralized methods that are used for dimensionality reduction. In PCA, eigenvalue decomposition of the covariance matrix for a mean centered data is used to find the principal components. However, in SVD data doesn't need to be mean centralized. In both methods, we need to find the covariance matrix of the data as X^TX where X denotes the mean centralized data matrix. Then we can eigenvalue decompose the data as $\Lambda\lambda\Lambda^T$ where Λ is the eigenvector matrix also referred as the eigenmatrix and λ corresponds to the diagonal eigenvalues.

In SVD, we decompose the covariance matrix into USV^T, where U and V are the orthogonal matrices with orthonormal eigenvalues chosen from XX^T and X^TX respectively and S is the diagonal matrix with elements equal to the rank of the covariance matrix equal to the root of the positive eigenvalues of XX^T and X^TX which are the same. Unlike PCA, SVD gives more information regarding diagonalizing the data matrix into special matrices that are easy to manipulate and analyze. If we have used SVD, for image compression, we just needed to drop the more insignificant vectors and transform the data into a new basis with truncated matrix.

1.2

The explained variance is the variance explained by the model. Meaning, the number we choose for how many eigenvectors we will be explaining only a percentage of the total variance. This resembles how much information that we are conserving after the PCA compression. In Principal Component Analysis, the explained variance can be found from the ratio of the sum of eigenvalues whose eigenvectors are used in the eigenmatrix divided by the total variance, where total variance is defined as the individual variances given by the eigenvectors. The table that shows the explained variances is given below, with its respective plot.

$$Explained\ Variance = \frac{\lambda_0 + \lambda_1 + \lambda_2 + \cdots}{\sum_i \lambda_i}$$

K	Percent Explained Variance
16	71.74538005241278
32	80.39070295629575
64	88.24575519340861
128	94.17002065228664
256	97.85694729671786
512	99.56732620377616
1024	100.00000000000007
2048	100.0000000000002
4096	100.00000000000001

Fig. 1: Table for the Percent Explained Variance

Note that that after some time, the percent explained variance goes over 100. This is due to the finite precision processor can provide. Eigen values contain many very small negative values

which are meant to be zero. Hence, these can be classified as computation errors, Hence, we can simply think of the values that are over 100 as 100.

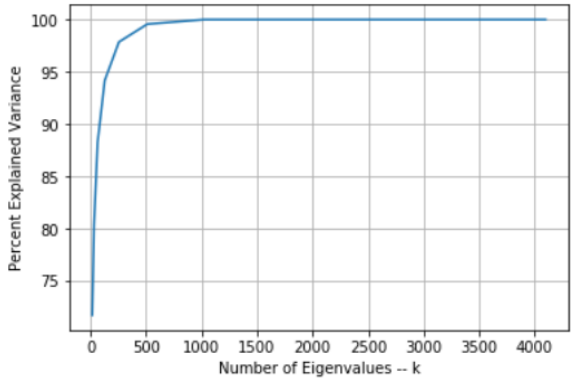

Fig. 2: Percent Explained Variances for the given *k* values

1.3

Assuming chosen *k* eigenvectors, the eigenmatrix that is obtained can be written as $V_{(k,4096)}$. In order to reconstruct the images, we should project this vector back into the image space dimensions (1000,4096). Hence, we need to use the matrix multiplication given below.

$$\widehat{D} = DV^T V + mean$$

where, *D* is the data matrix with the dimensions (1000,4096). The required grid with the original 10 images, the first 10 eigenfaces and the ten different reconstructed images for three different K values are given below.

Fig. 3: 5x10 Face Grid

It is easily seen that as the number of eigenfaces increase, the image gets closer and closer to its original form as the number of eigenfaces increase the explained variance gets closer to the total variance, hence the original image.

1.4

The minimum number of eigenfaces required to reach 95% explained variance is found to be 145, and the total percentage explained variance has become 95.000693% in minimum. The compression ratio with the PCA algorithm is given as [1].

$$CR = \frac{m*n}{2*PC+1}$$

,where m and n are the dimensions of the image that is compressed. In our case, m and n are both equal to 64. Hence, the compression ratio for k=145 eigenvectors become 64*64/(2*145+1), which is equal to 14.0756.

Linear Regression

2.1

The loss function for linear regression is an ordinary least squares loss function (OLS), which is given as,

[1] Principal Component Analysis, Mark Richardson, May 2009, http://www.dsc.ufcg.edu.br/~hmg/disciplinas/posgraduacao/rn-copin-2014.3/material/SignalProcPCA.pdf

$$J_n = ||y - X\beta||^2$$

In order to achieve the general closed loop solution for the multivariate regression, we need to first expand the square and then take derivative with respect to β. In order to minimize the loss, then we need to equate the derivative to zero and solve for β. In this equation, y represents the labels, X is the feature matrix and β is the weight vector. One important thing is that, the X matrix should contain a pseudo-column as its first column filled with ones in order to construct the constant DC gain β_0.

$$J_n = ||y - X\beta||^2 = y^T y - 2\beta^T X^T y + \beta^T X^T X\beta$$

$$\frac{\partial J_n}{\partial \beta} = \nabla_\beta y^T y - 2\nabla_\beta \beta^T X^T y + \nabla_\beta \beta^T X^T X\beta = -2X^T y + 2X^T X\beta = 0$$

$$X^T X\beta = X^T y$$

$$\beta = (X^T X)^{-1} X^T y$$

2.2

With 9 feature columns, the rank of the covariance matrix turns out to be 9, with the inclusion of the first constant weight column. This tells me that this matrix is full rank and hence, $(X^T X)^{-1}$ exists. Hence, we see that, for this data, β exists.

2.3

The coefficients for the trained model became as below.

$$\beta_0 = 35.52032936$$

$$\beta_1 = -0.47656134$$

$$\beta_2 = 7.00054171$$

$$\beta_3 = 1.37221717$$

$$\beta_4 = -4.24724733$$

$$\beta_5 = 68.4775968$$

$$\beta_6 = 286.19995694$$

$$\beta_7 = -180.5873212$$

$$\beta_8 = 24.55935004$$

Then, the mean squared error for the training set has become 18204.227976890255 and the mean squared error for the test set has become 40379.06782240237.

2.4

The positive weights show a positive correlation between the feature and the label, while the negative coefficients show a negative correlation between the feature and the label. In our case, the temperature is positively correlated with the number of bikes, and humidity is inversely

correlated with the number of bikes, which is expected. The magnitude of a coefficient shows how impactful the feature is on the prediction. The graphs for the prediction and label data with respect to humidity are given below.

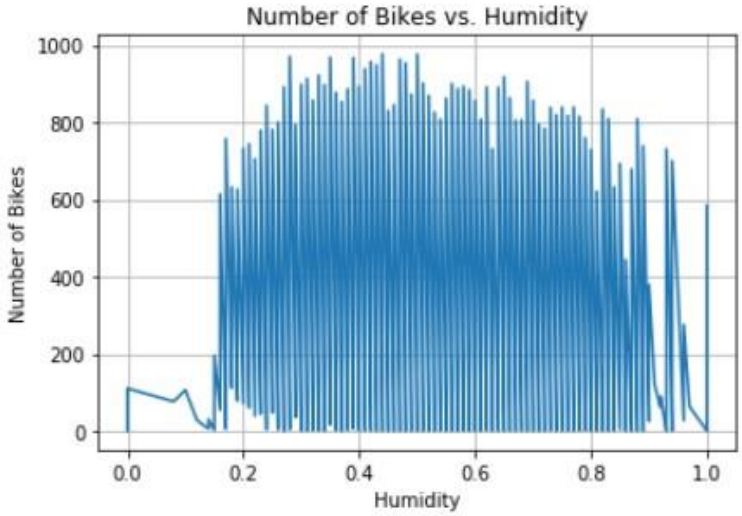

Fig. 4: Training and Test Data with Respect to Humidity

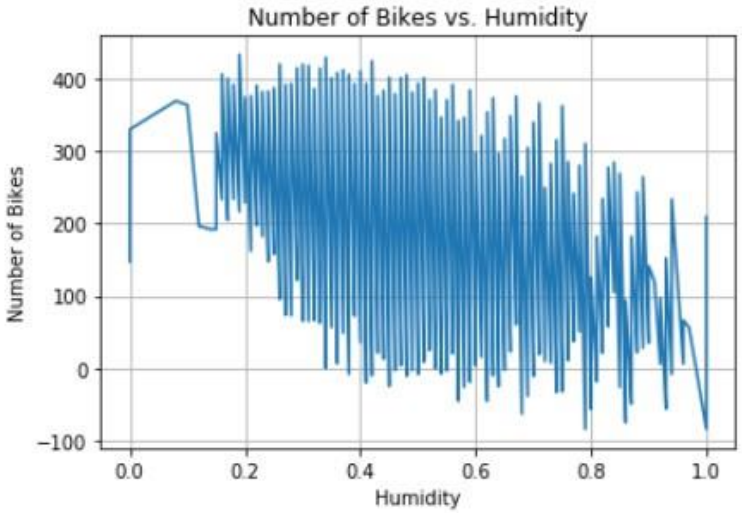

Fig. 5: Training and Test Predictions with Respect to Humidity

There seems to be a linear relationship between humidity and number of bikes, since the real data shows that The rental number is high when the humidity is between 0.2 and 0.85, when we look at the prediction, we see that the same intervals are predicted correctly although the numbers do change.

2.5

This time, we have run the linear regression only using the humidity as the feature set. The plot on the number of bikes with respect to humidity is given below.

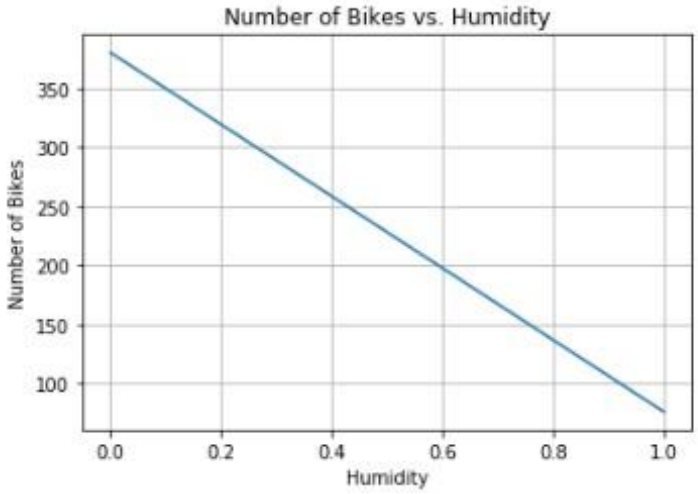

Fig. 6: Training and Test Predictions with Respect to Humidity training only on Humidity

This is a linear model with only one feature; hence, we would accept this model to be a line equation. Overall, this model shows the same trend in the prediction when compared to Fig. 5. The coefficient of humidity on the multivariate training model, the coefficient for the humidity is big when compared with the others, so we see that this is a more important feature than the others mostly. Hence, we would expect the linear regression model with the humidity feature to move on the same trends with the multivariate regression model.

Logistic Regression

3.1

In this part, we have applied a full batch gradient ascent in order to learn the logistic regression parameters for different learning rates. In order to decide on the best model, I have used the log likelihood function as a comparison and calculated the test accuracies. Below is the plot for the max likelihoods vs. learning rate.

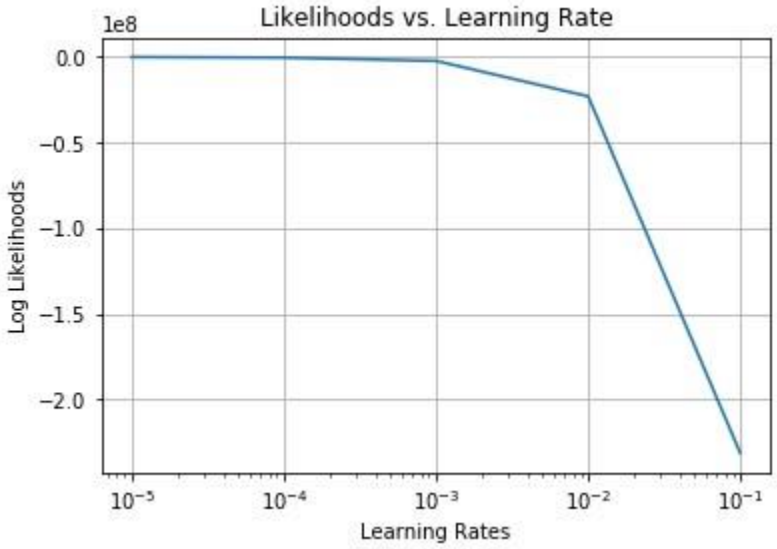

Fig. 7: Accuracy vs Learning Rate on the Test Data

Here, I have decided to use 10^{-5} as the learning rate since it resulted better than the others. Then I tested with the optimal weights the test data and reported the requested statistics. Since, this is a binary classification task, the micro and macro averages would turn out to be the same, and their average would too. The performance metrics are given below.

```
Accuracy: 0.7048549437537004
----------
Confusion Matrix for Positive Class
             actual+  actual-
classifier+     1721      953
classifier-       44      660
---------
Macro Statistics
Macro Precision: 0.7905525430067315
Macro Recall: 0.6921231355013884
Macro NPV: 0.7905525430067315
Macro FPR: 0.3078768644986117
Macro FDR: 0.2094474569932685
Macro F1: 0.6725510329782816
Macro F2: 0.6725831373908745
---------
Micro Statistics
Micro Precision: 0.7048549437537004
Micro Recall: 0.7048549437537004
Micro NPV: 0.7048549437537004
Micro FPR: 0.2951450562462996
Micro FDR: 0.2951450562462996
Micro F1: 0.6819849671091096
Micro F2: 0.655860847494576
----------
```

3.2

The mini-batch gradient ascent algorithm requires us to update the weights every batch, with a specified batch size which was given as 32, Since, the sample size was non-divisible by 32, we had a residual batch with smaller size. Hence, we have normalized and again expanded the batch according to the number 32, since we have used that for the other algorithms. The performance measures are given below.

```
Accuracy: 0.6862048549437537
----------
Confusion Matrix for Positive Class
             actual+  actual-
classifier+      853      148
classifier-      912     1465
---------
Macro Statistics
Macro Precision: 0.7342354742438882
Macro Recall: 0.6957658121249269
Macro NPV: 0.7342354742438882
Macro FPR: 0.3042341878750731
Macro FDR: 0.26576452575611176
Macro F1: 0.6755554830677561
Macro F2: 0.6793714828948466
---------
Micro Statistics
Micro Precision: 0.6862048549437537
Micro Recall: 0.6862048549437537
Micro NPV: 0.6862048549437537
Micro FPR: 0.3137951450562463
Micro FDR: 0.3137951450562463
Micro F1: 0.6794426827220804
Micro F2: 0.6686499356061869
----------
```

Then, we have implemented a stochastic gradient ascent algorithm, which is basically a mini batch gradient ascent algorithm with a batch size of 1. The performance measures are given below.

```
Accuracy: 0.6859088217880402
----------
Confusion Matrix for Positive Class
             actual+  actual-
classifier+      852      148
classifier-      913     1465
---------
Macro Statistics
Macro Precision: 0.7340319596299412
Macro Recall: 0.6954825260059467
Macro NPV: 0.7340319596299412
Macro FPR: 0.30451747399405327
Macro FDR: 0.2659680403700589
```

```
Macro F1: 0.6752133530099143
Macro F2: 0.6790471520223464
---------
Micro Statistics
Micro Precision: 0.6859088217880402
Micro Recall: 0.6859088217880402
Micro NPV: 0.6859088217880402
Micro FPR: 0.31409117821195975
Micro FDR: 0.31409117821195975
Micro F1: 0.6791217222392335
Micro F2: 0.6683084375832844
----------
```

3.3

When the classes for the classification tasks are distributed unevenly, precision, recall and accuracy are not good enough metrics, since if some class has high advantage in terms of number to the other the accuracy con dominate due to one class but the other may be drastically mispredicted. As an example, if a classifier, in a binary classification task, would predict everything to be positive and positive samples are dominant, recall and accuracy would be exceptionally high but the classifier itself would be very ineffective if it at the same time classifies the negatives horribly. However, FPR and FDR would help in observing the performance better.

Support Vector Machines (SVMs)

4.1

For SVM, I have decided to use the sklearn.svm library to train the SVM. Then, for each fold, I have trained six different classifiers, each with C values {0.001,0.01,0.1,1,10,100}. Then, used the validation of that batch to test and collect the accuracy. Then, I have taken the mean of accuracies for each C value and plot the mean cross validation accuracies. The plot is given below. The plot is done on a semi log scale in order to observe the changes better.

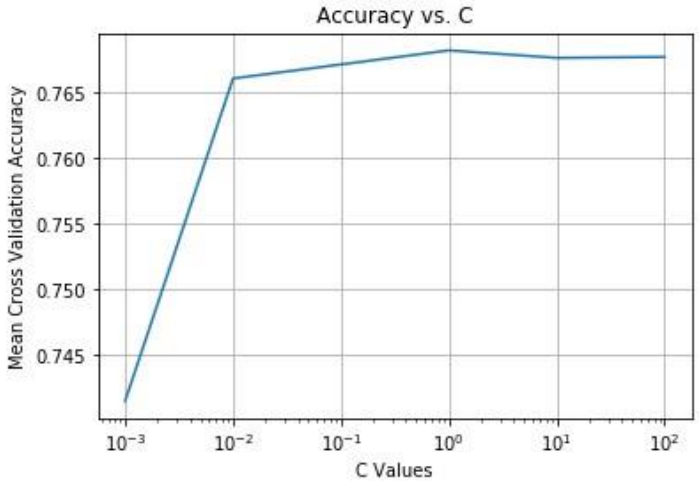

Fig. 8: Mean Cross Validation Accuracy vs C Values

Hence, I have picked C=1 as the optimal C value. Then I have reported the performance metrics on the test data, which are given below.

```
Accuracy: 0.6983422143280047
----------
Confusion Matrix for Positive Class
             actual+  actual-
classifier+      847      101
classifier-      918     1512
---------
Macro Statistics
Macro Precision: 0.7578410689170183
Macro Recall: 0.7086352212634948
Macro NPV: 0.7578410689170183
Macro FPR: 0.2913647787365053
Macro FDR: 0.2421589310829817
Macro F1: 0.6861802340650758
Macro F2: 0.690002901286959
---------
Micro Statistics
Micro Precision: 0.6983422143280047
Micro Recall: 0.6983422143280047
Micro NPV: 0.6983422143280047
Micro FPR: 0.30165778567199525
Micro FDR: 0.30165778567199525
Micro F1: 0.6901035691426383
Micro F2: 0.6764843970881051
----------
```

4.2

This question consisted of the reimplementation of 4.2 with a hard margin SVM with an RBF (Radial Basis Function) kernel. For the C value, I have used the optimized value form the previous question and changed the hyperparameter gamma according to the given interval. Averaged and then plot the mean cross validation vs accuracy plot. The plot is given below.

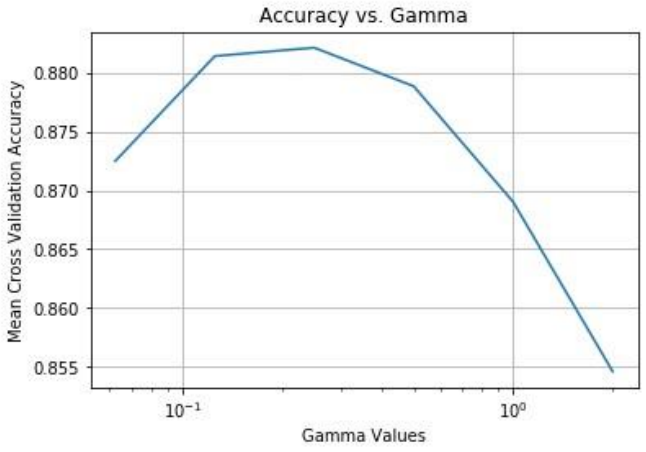

Fig. 9: Mean Cross Validation Accuracy vs Gamma Values

Hence, I have chosen the gamma value that maximized the accuracy, which is 0.25. Hence, with the hyperparameters, I have configured, I have trained the hard margin SVM and reported the results and performance metrics, which are given below.

```
Accuracy: 0.8031379514505624
----------
Confusion Matrix for Positive Class
             actual+  actual-
classifier+     1180       80
classifier-      585     1533
---------
Macro Statistics
Macro Precision: 0.8301519852511354
Macro Recall: 0.809479108307326
Macro NPV: 0.8301519852511354
Macro FPR: 0.19052089169267408
Macro FDR: 0.1698480147488646
Macro F1: 0.8009644457538028
Macro F2: 0.8017668409478503
---------
Micro Statistics
Micro Precision: 0.8031379514505624
Micro Recall: 0.8031379514505624
Micro NPV: 0.8031379514505624
Micro FPR: 0.19686204854943753
Micro FDR: 0.19686204854943753
Micro F1: 0.8014029489651999
Micro F2: 0.795385894960567
----------
```

Appendix

Python Code of Question 1 (.ipynb File)

```
#!/usr/bin/env python
# coding: utf-8

# # CS464 Introduction to Machine Learning Homework 2 Question 1

# In[1]:


import cv2
import glob
from matplotlib import pyplot as plt
import numpy as np


# In[2]:
```

```
#define PCA function
def PCA(eigenvalues, eigenvectors, k):
    #argsort the eigenvalues in descending order
    eigvals_ind = np.argsort(eigenvalues)[::-1]
    #sort the eigenvalues and eigenvectors accordingly
    eigenvalues = eigenvalues[eigvals_ind]
    eigenvectors = eigenvectors[:,eigvals_ind]
    #select k biggest eigenvectors
    eigfcs = eigenvectors[:,:k]
    #get the transpose of the eigenmatrix
    eigfcs = eigfcs.T
    return eigfcs


# In[3]:


#define the function to project the eigenmatrix into the image space
def reconstruct(data_mn, eigfcs, mean):
    #data==D, eigenfaces==E
    #reconstructed_data = (D*E')*E + mean
    proj = np.matmul(data_mn,eigfcs.T)
    proj = np.matmul(proj,eigfcs) + mean
    return proj


# In[4]:


#read dataset
images = [cv2.imread(file,cv2.IMREAD_UNCHANGED) for file in
glob.glob('lfwdataset/*.pgm')]


# In[5]:


#set dimensions of the data
data_num = 1000
dim = 64


# In[6]:


#flatten the images and convert the images into a 1000x4096 data matrix
row_images = np.arange(4096*1000).reshape(data_num,dim*dim)
for i in range(1000):
    row_images[i,:] = images[i].flatten()
```

```
# In[7]:


#get the mean of each column and subtract it from the data matrix
mean = np.mean(row_images, axis=0)
data_mn = row_images - mean


# In[8]:


#calculate X^TX
cov_mat = np.matmul(data_mn.T,data_mn)


# In[9]:


#extract eigenvalues of X'X
#used linalg.eigh instead of linalg.eig since the scatter matrix is symmetric
eigenvalues, eigenvectors = np.linalg.eigh(cov_mat)
for i in range(eigenvalues.shape[0]):
    print(eigenvalues[i])


# # Question 1.2

# In[10]:


#explained variances
#argsort the eigenvalues in descending order
eigvals_ind = np.argsort(eigenvalues)[::-1]
#sort the eigenvalues and eigenvectors accordingly
eigenvalues = eigenvalues[eigvals_ind]
eigenvectors = eigenvectors[:,eigvals_ind]
k_vals = np.array([16, 32, 64, 128, 256, 512, 1024, 2048, 4096])
tot_var = np.sum(eigenvalues)
vas = np.zeros(k_vals.shape[0])
for ind in range(k_vals.shape[0]):
    vas[ind] = np.sum(eigenvalues[:k_vals[ind]]/tot_var*100.0)
print("Percent Explained Variances")
for i in range(vas.shape[0]):
    print("PEV for k=" + str(k_vals[i]) + " ", vas[i])
plt.plot(k_vals,vas)
plt.xlabel("Number of Eigenvalues -- k")
plt.ylabel("Percent Explained Variance")
plt.grid()
plt.show()
```

```
# # Question 1.3

# In[11]:


col_num = 10
row_num = 5
# plot originals
for ind in range(col_num):
    plt.subplot(row_num,col_num,ind+1)
    plt.imshow(row_images[ind].reshape((64,64)),cmap='gray')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.gca().axes.get_xaxis().set_visible(False)
#plot the first 6 eigenfaces
k=col_num
eigenfcs = PCA(eigenvalues,eigenvectors,k)
for ind in range(col_num):
    plt.subplot(row_num,col_num,ind+col_num+1)
    plt.imshow(eigenfcs[ind].reshape((64,64)),cmap='gray')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.gca().axes.get_xaxis().set_visible(False)
k_vals = np.array([36, 128, 512])
#plot the reconstructed faces
for ind in range(k_vals.shape[0]):
    eigfcs = PCA(eigenvalues, eigenvectors,k_vals[ind])
    rcn = reconstruct(data_mn, eigfcs, mean)
    for i in range(col_num):
        plt.subplot(row_num,col_num,col_num*(ind+2)+i+1)
        plt.imshow(rcn[i].reshape((64,64)),cmap='gray')
        plt.gca().axes.get_yaxis().set_visible(False)
        plt.gca().axes.get_xaxis().set_visible(False)
plt.show()


# ## Question 1.4

# In[12]:


var = 0;
count = 0;
while(var < 0.95):
    var += eigenvalues[count]/tot_var
    count += 1
print("Number of eigenfaces required: " + str(count))
print("The total explained variance: " + str(var))


# In[ ]:
```

Python Code of Question 2 (.ipynb File)

```
#!/usr/bin/env python
# coding: utf-8

# # CS464 Introduction to Machine Learning Homework 2 Question 2 - Linear
Regression

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# In[2]:


def linRegCoef(x,y):
    cov = np.matmul(x.T,x)
    cov_inv = np.linalg.inv(cov)
    beta = np.matmul(np.matmul(cov_inv,x.T),y)
    return beta


# In[3]:


def predict(x,w):
    return np.matmul(x, w)


# In[4]:


def mse(pr, y):
    return np.matmul((y-pr).T,(y-pr))/pr.shape[0]


# In[5]:


#import data
x_train = pd.read_csv('question-2-train-features.csv', header=None, sep=',',
names=["month","hr",

"weekday","weathersit",
```

```
"temp","atemp","hum",

"windspeed"])
y_train = pd.read_csv('question-2-train-labels.csv', header=None, sep=',',
names=["bikes"])
x_test = pd.read_csv('question-2-test-features.csv', header=None, sep=',',
names=["month","hr",

"weekday","weathersit",

"temp","atemp","hum",

"windspeed"])
y_test = pd.read_csv('question-2-test-labels.csv', header=None, names=["bikes"])

x_test.insert(loc=0, column='pseudo', value=np.ones(x_test.shape[0]))
x_train.insert(loc=0, column='pseudo', value=np.ones(x_train.shape[0]))

x_train_arr = np.asarray(x_train)
y_train_arr = np.asarray(y_train)
x_test_arr = np.asarray(x_test)
y_test_arr = np.asarray(y_test)
print(x_train.head(5))
print(y_train.head(5))
print(x_train.shape)
print(x_test.head(5))
print(y_test.head(5))
print(x_test.shape)


# # Quesiton 2.2
# ## Data Rank

# In[6]:


print(np.linalg.matrix_rank(np.matmul(x_train_arr.T,x_train_arr)))
# it tells me the matrix is full rank (rxc matric where r>c, col rank = 8 = # of
columns)


# # Question 2.3
# ## Linear Regression on the Whole Dataset

# In[7]:


#calculate weights
weights = linRegCoef(x_train_arr,y_train_arr)
print(weights.shape)
```

```
print(weights)


# In[8]:


#training error
tr_pred = predict(x_train_arr,weights)
tr_pred_error = mse(tr_pred,y_train_arr).reshape(1)
print("Prediction Error: " + str(tr_pred_error[0]))


# In[9]:


#test error
test_pred = predict(x_test_arr,weights)
test_pred_error = mse(test_pred,y_test_arr).reshape(1)
print("Test Error: " + str(test_pred_error[0]))


# # Question 2.4
# ## Plot Predicted Number of Bikes vs Humidity

# In[10]:


trtest_hum = pd.concat([x_train.iloc[:,[7]],x_test.iloc[:,[7]]],axis=0)
trtest_hum_arr = np.asarray(trtest_hum)
trtest_bikes = pd.concat([pd.DataFrame(tr_pred), pd.DataFrame(test_pred)],axis=0)
trtest_bikes_arr = np.asarray(trtest_bikes)
xsorted, ysorted = zip(*sorted(zip(trtest_hum_arr, trtest_bikes_arr)))
plt.plot(xsorted,ysorted)
plt.ylabel("Number of Bikes")
plt.xlabel("Humidity")
plt.title("Number of Bikes vs. Humidity")
plt.grid()
plt.show()


# ## Plot the Number of Bikes vs Humidity

# In[11]:


trtest_hum = pd.concat([x_train.iloc[:,[7]],x_test.iloc[:,[7]]],axis=0)
trtest_hum_arr = np.asarray(trtest_hum)
trtest_bikes = pd.concat([y_train, y_test],axis=0)
trtest_bikes_arr = np.asarray(trtest_bikes)
xsorted, ysorted = zip(*sorted(zip(trtest_hum_arr, trtest_bikes_arr)))
plt.plot(xsorted,ysorted)
```

```
plt.ylabel("Number of Bikes")
plt.xlabel("Humidity")
plt.title("Number of Bikes vs. Humidity")
plt.grid()
plt.show()


# # Question 2.5
# ## Linear Regression using only Normalized Humidity

# In[12]:


x_tr_hum = x_train.iloc[:,[0,7]]
x_test_hum = x_test.iloc[:,[0,7]]

print(x_tr_hum.head(5))
print(y_train.head(5))
print(x_test_hum.head(5))
print(y_test.head(5))

x_tr_hum_arr = np.asarray(x_tr_hum)
x_test_hum_arr = np.asarray(x_test_hum)


# In[14]:


#calculate weights
hum_weights = linRegCoef(np.concatenate([x_tr_hum_arr,
x_test_hum_arr],axis=0),np.concatenate([y_train_arr,y_test_arr],axis=0))
print(hum_weights.shape)
print(hum_weights)


# In[15]:


#training error
tr_pred_hum = predict(x_tr_hum_arr,hum_weights)
tr_pred_error_hum = mse(tr_pred_hum,y_train_arr).reshape(1)
print("Prediction Error: " + str(tr_pred_error_hum[0]))


# In[16]:


#test error
test_pred_hum = predict(x_test_hum_arr,hum_weights)
test_pred_error = mse(test_pred_hum,y_test_arr).reshape(1)
print("Test Error: " + str(test_pred_error[0]))
```

```
# In[17]:


trtest_bikes_hum = pd.concat([pd.DataFrame(tr_pred_hum),
pd.DataFrame(test_pred_hum)],axis=0)
trtest_bikes_arr_hum = np.asarray(trtest_bikes_hum)
xsorted, ysorted = zip(*sorted(zip(trtest_hum_arr,trtest_bikes_arr_hum)))
plt.plot(xsorted,ysorted)
plt.ylabel("Number of Bikes")
plt.xlabel("Humidity")
plt.title("Number of Bikes vs. Humidity")
plt.grid()
plt.show()


# In[ ]:
```

Python Code of Question 3 (.ipynb File)

```
#!/usr/bin/env python
# coding: utf-8

# # CS464 Introduction to Machine Learning Homework 2 Question 2 - Logistic
Regression

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random


# In[2]:


def logit_mat(x,w):
    z = np.exp(-1*np.matmul(x,w))
    return 1/(1+z)


# In[3]:


def predict(x,w):
```

```
    length = x.shape[0]
    prediction = np.zeros(length)
    temp = np.matmul(x, w)
    for i in range(length):
        if(temp[i] > 0):
            prediction[i] = 1
        else:
            prediction[i] = 0
    return prediction


# In[4]:


def batch_ascent(x,w,y):
    return np.matmul(x.T,y-logit_mat(x,w))


# In[5]:


#accuracy and confusion matrix calculation
def performanceCalc(pred, y):
    #accuracy
    sampleSize = y.shape[0]
    true_cnt = 0
    for i in range(sampleSize):
        if(pred[i] == y[i]):
            true_cnt += 1
    accuracy = true_cnt / sampleSize

    tr_pos = 0;
    tr_neg = 0;
    fls_pos = 0;
    fls_neg = 0;
    for i in range(sampleSize):
        if(pred[i] == 1 and y[i] ==1):
            tr_pos += 1
        elif(pred[i] == 0 and y[i] == 0):
            tr_neg += 1
        elif(pred[i] == 1 and y[i] == 0):
            fls_pos += 1
        elif(pred[i] == 0 and y[i] == 1):
            fls_neg += 1


    #performance metrics
    precision = tr_pos / (tr_pos+fls_pos)
    recall = tr_pos / (tr_pos+fls_neg)
    npv = tr_neg / (tr_neg+fls_neg)
    fpr = fls_pos / (fls_pos+tr_neg)
```

```
    fdr = fls_pos / (tr_pos+fls_pos)
    f1 = (2*precision*recall)/(precision+recall)
    f2 = (5*precision*recall)/(4*precision+recall)
    return accuracy, tr_pos, tr_neg, fls_pos,fls_neg , precision, recall, npv,
fpr, fdr, f1, f2


# In[6]:


def macroMicroAvr(pred, y):
    accuracy_pos, tr_pos_pos, tr_neg_pos, fls_pos_pos,fls_neg_pos ,
precision_pos, recall_pos, npv_pos, fpr_pos, fdr_pos, f1_pos, f2_pos =
performanceCalc(pred,y)
    pred_neg = (pred==0).astype(float)
    y_neg = (y==0).astype(float)
    accuracy_neg, tr_pos_neg, tr_neg_neg, fls_pos_neg, fls_neg_neg,
precision_neg, recall_neg, npv_neg, fpr_neg, fdr_neg, f1_neg, f2_neg =
performanceCalc(pred_neg,y_neg)

    print("Accuracy: " + str(accuracy_pos))
    print("----------")
    print("Confusion Matrix for Positive Class")
    conf_matrix = pd.DataFrame([[tr_pos_pos, fls_pos_pos],[fls_neg_pos,
tr_neg_pos]])
    conf_matrix.columns = ["actual+", "actual-"]
    conf_matrix.rename(index={0:'classifier+',1:'classifier-'}, inplace=True)
    print(conf_matrix)

    #macro
    mac_precision = (precision_pos+precision_neg)/2
    mac_recall = (recall_pos+recall_neg)/2
    mac_npv = (npv_pos+npv_neg)/2
    mac_fpr = (fpr_pos+fpr_neg)/2
    mac_fdr = (fdr_pos+fdr_neg)/2
    mac_f1 = (f1_pos+f1_neg)/2
    mac_f2 = (f2_pos+f2_neg)/2
    #micro
    mic_precision =
(tr_pos_pos+tr_pos_neg)/(tr_pos_pos+tr_pos_neg+fls_pos_pos+fls_pos_neg)
    mic_recall =
(tr_pos_pos+tr_pos_neg)/(tr_pos_pos+tr_pos_neg+fls_neg_pos+fls_neg_neg)
    mic_npv =
(tr_neg_pos+tr_neg_neg)/(tr_neg_pos+tr_neg_neg+fls_neg_pos+fls_neg_neg)
    mic_fpr =
(fls_pos_pos+fls_pos_neg)/(fls_pos_pos+fls_pos_neg+tr_neg_pos+tr_neg_neg)
    mic_fdr =
(fls_pos_pos+fls_pos_neg)/(fls_pos_pos+fls_pos_neg+tr_pos_pos+tr_pos_neg)
    mic_f1 =
(2*precision_pos*recall_pos+2*precision_neg*recall_neg)/(precision_pos+recall_pos
+precision_neg+recall_neg)
```

```
    mic_f2 =
(5*precision_pos*recall_pos+5*precision_neg*recall_neg)/(4*precision_pos+recall_p
os+4*precision_neg+recall_neg)

    print("---------")
    print("Macro Statistics")
    print("Macro Precision: " + str(mac_precision))
    print("Macro Recall: " + str(mac_recall))
    print("Macro NPV: " + str(mac_npv))
    print("Macro FPR: " + str(mac_fpr))
    print("Macro FDR: " + str(mac_fdr))
    print("Macro F1: " + str(mac_f1))
    print("Macro F2: " + str(mac_f2))
    print("---------")
    print("Micro Statistics")
    print("Micro Precision: " + str(mic_precision))
    print("Micro Recall: " + str(mic_recall))
    print("Micro NPV: " + str(mic_npv))
    print("Micro FPR: " + str(mic_fpr))
    print("Micro FDR: " + str(mic_fdr))
    print("Micro F1: " + str(mic_f1))
    print("Micro F2: " + str(mic_f2))
    print("----------")


# In[7]:


def accuracyCalc(pred, y):
    sampleSize = y.shape[0]
    true_cnt = 0
    for i in range(sampleSize):
        if(pred[i] == y[i]):
            true_cnt += 1
    accuracy = true_cnt / sampleSize
    return accuracy


# In[8]:


def loglikelihood(w,x,y):
    z = np.matmul(x,w)
    return np.sum(np.multiply(y,z) - np.log(1+np.exp(z)))


# In[9]:


def chunks(l, n):
    for i in range(0, len(l), n):
```

```
        yield l[i:i+n]


# In[10]:


#import data
x_train = pd.read_csv('question-2-train-features.csv', header=None, sep=',',
names=["month","hr",

"weekday","weathersit",

"temp","atemp","hum",

"windspeed"])
y_train = pd.read_csv('question-2-train-labels.csv', header=None, sep=',',
names=["bikes"])
x_test = pd.read_csv('question-2-test-features.csv', header=None, sep=',',
names=["month","hr",

"weekday","weathersit",

"temp","atemp","hum",

"windspeed"])
y_test = pd.read_csv('question-2-test-labels.csv', header=None, names=["bikes"])

x_test.insert(loc=0, column='pseudo', value=np.ones(x_test.shape[0]))
x_train.insert(loc=0, column='pseudo', value=np.ones(x_train.shape[0]))

x_train_arr = np.asarray(x_train)
y_train_arr = np.asarray(y_train)
x_test_arr = np.asarray(x_test)
y_test_arr = np.asarray(y_test)
print(x_train.head(5))
print(y_train.head(5))
print(x_train.shape)
print(x_test.head(5))
print(y_test.head(5))
print(x_test.shape)


# In[11]:


#discretizing the labels
#find mean of labels
all_label = pd.concat([y_test, y_train], axis=0, sort = True)
label_mean = np.mean(all_label.values)
print(label_mean)
#discretizing
```

```
for i in range(y_test.shape[0]):
    if(y_test_arr[i] >= label_mean):
        y_test_arr[i] = 1
    else:
        y_test_arr[i] = 0
y_test = pd.DataFrame(y_test_arr)
print(y_test.shape)
for i in range(y_train.shape[0]):
    if(y_train_arr[i] >= label_mean):
        y_train_arr[i] = 1
    else:
        y_train_arr[i] = 0
y_train = pd.DataFrame(y_train_arr)
print(y_train.shape)


# # Question 3.1
# ## Try different learning rates to choose the best one.

# In[12]:


#initialize the weight vector
weight_temp = np.zeros((x_train_arr.shape[1],1))
#apply gradient ascent to find the logistic regression parameters
iterationNo = 1000
learn_rates = np.array([10**-5, 10**-4, 10**-3, 10**-2, 10**-1])
#init weight matrix
weights = np.zeros((x_train_arr.shape[1],learn_rates.shape[0]))
cnt = 0
for rate in learn_rates:
    for i in range(iterationNo):
        weight_temp += rate * batch_ascent(x_train_arr,weight_temp,y_train_arr)
    weights[:,cnt] = weight_temp.reshape(x_train_arr.shape[1])
    cnt+=1
    #reinitialize the weight vector
    weight_temp = np.zeros((x_train_arr.shape[1],1))
print(weights)


# In[13]:


#choosing will be based on the max log likelihood
likes = np.zeros((weights.shape[1],1))
for i in range(weights.shape[1]):
    likes[i] =
loglikelihood(weights[:,i].reshape((weights.shape[0],1)),x_train_arr,y_train_arr)
print(likes)
```

```
# In[14]:


plt.semilogx(learn_rates,likes)
plt.xlabel("Learning Rates")
plt.ylabel("Log Likelihoods")
plt.title("Likelihoods vs. Learning Rate")
plt.grid()
plt.show()


# In[15]:


opt_weight = weights[:,np.argmax(likes)]
print(opt_weight)
opt_rate = learn_rates[np.argmax(likes)]
print(opt_rate)


# ## Test and Report Results

# In[16]:


prediction = predict(x_test_arr,opt_weight.reshape((opt_weight.shape[0],1)))
macroMicroAvr(prediction, y_test_arr)


# # Question 3.2
# ## Mini-Batch Gradient Ascent

# In[17]:


batch_size = 32
iterationNo = 1000
batch_no = np.ceil(x_train_arr.shape[0] / batch_size)

min_btc_weights = np.zeros((x_train.shape[1],1))

np.random.seed(0)
for i in range(iterationNo):
    idx = np.random.permutation(len(x_train_arr))
    x_tr_arr_shuf,y_tr_arr_shuf = x_train_arr[idx], y_train_arr[idx]
    x_cnks = list(chunks(x_tr_arr_shuf, batch_size))
    y_cnks = list(chunks(y_tr_arr_shuf, batch_size))
    for b_ind in range(int(batch_no)):
        batch_x = x_cnks[b_ind]
        batch_y = y_cnks[b_ind]
        min_btc_weights += (opt_rate/batch_size) *
```

```
(batch_ascent(batch_x,min_btc_weights,batch_y))*batch_x.shape[0]
print(min_btc_weights)



# In[18]:


min_btc_prediction = predict(x_test_arr,min_btc_weights)
macroMicroAvr(min_btc_prediction, y_test_arr)


# ## Stochastic Gradient Ascent

# In[19]:


#initialize the weight vector
stoc_weights = np.zeros((x_train.shape[1],1))
#apply gradient ascent to find the logistic regression parameters
iterationNo = 1000
np.random.seed(0)
cnt=0
for i in range(iterationNo):
    idx = np.random.permutation(len(x_train_arr))
    x_tr_arr_shuf,y_tr_arr_shuf = x_train_arr[idx], y_train_arr[idx]
    for b_ind in range(x_tr_arr_shuf.shape[0]):
        rand_feature =
x_tr_arr_shuf[b_ind,:].reshape((x_tr_arr_shuf.shape[1],1)).T
        rand_label = y_tr_arr_shuf[b_ind,:].reshape((y_tr_arr_shuf.shape[1],1))
        stoc_weights += opt_rate *
batch_ascent(rand_feature,stoc_weights,rand_label)
print(stoc_weights)


# In[20]:


sto_prediction = predict(x_test_arr,stoc_weights)
macroMicroAvr(sto_prediction, y_test_arr)
```

Python Code of Question 4 (.ipynb File)

```
#!/usr/bin/env python
# coding: utf-8

# # CS464 Introduction to Machine Learning Homework 2 Question 2 - Logistic
Regression
#

# In[1]:
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm


# In[2]:


#accuracy and confusion matrix calculation
def performanceCalc(pred, y):
    #accuracy
    sampleSize = y.shape[0]
    true_cnt = 0
    for i in range(sampleSize):
        if(pred[i] == y[i]):
            true_cnt += 1
    accuracy = true_cnt / sampleSize

    tr_pos = 0;
    tr_neg = 0;
    fls_pos = 0;
    fls_neg = 0;
    for i in range(sampleSize):
        if(pred[i] == 1 and y[i] ==1):
            tr_pos += 1
        elif(pred[i] == 0 and y[i] == 0):
            tr_neg += 1
        elif(pred[i] == 1 and y[i] == 0):
            fls_pos += 1
        elif(pred[i] == 0 and y[i] == 1):
            fls_neg += 1


    #performance metrics
    precision = tr_pos / (tr_pos+fls_pos)
    recall = tr_pos / (tr_pos+fls_neg)
    npv = tr_neg / (tr_neg+fls_neg)
    fpr = fls_pos / (fls_pos+tr_neg)
    fdr = fls_pos / (tr_pos+fls_pos)
    f1 = (2*precision*recall)/(precision+recall)
    f2 = (5*precision*recall)/(4*precision+recall)
    return accuracy, tr_pos, tr_neg, fls_pos,fls_neg , precision, recall, npv,
fpr, fdr, f1, f2


# In[3]:
```

```
def macroMicroAvr(pred, y):
    accuracy_pos, tr_pos_pos, tr_neg_pos, fls_pos_pos,fls_neg_pos ,
precision_pos, recall_pos, npv_pos, fpr_pos, fdr_pos, f1_pos, f2_pos =
performanceCalc(pred,y)
    pred_neg = (pred==0).astype(float)
    y_neg = (y==0).astype(float)
    accuracy_neg, tr_pos_neg, tr_neg_neg, fls_pos_neg, fls_neg_neg,
precision_neg, recall_neg, npv_neg, fpr_neg, fdr_neg, f1_neg, f2_neg =
performanceCalc(pred_neg,y_neg)

    print("Accuracy: " + str(accuracy_pos))
    print("----------")
    print("Confusion Matrix for Positive Class")
    conf_matrix = pd.DataFrame([[tr_pos_pos, fls_pos_pos],[fls_neg_pos,
tr_neg_pos]])
    conf_matrix.columns = ["actual+", "actual-"]
    conf_matrix.rename(index={0:'classifier+',1:'classifier-'}, inplace=True)
    print(conf_matrix)

    #macro
    mac_precision = (precision_pos+precision_neg)/2
    mac_recall = (recall_pos+recall_neg)/2
    mac_npv = (npv_pos+npv_neg)/2
    mac_fpr = (fpr_pos+fpr_neg)/2
    mac_fdr = (fdr_pos+fdr_neg)/2
    mac_f1 = (f1_pos+f1_neg)/2
    mac_f2 = (f2_pos+f2_neg)/2
    #micro
    mic_precision =
(tr_pos_pos+tr_pos_neg)/(tr_pos_pos+tr_pos_neg+fls_pos_pos+fls_pos_neg)
    mic_recall =
(tr_pos_pos+tr_pos_neg)/(tr_pos_pos+tr_pos_neg+fls_neg_pos+fls_neg_neg)
    mic_npv =
(tr_neg_pos+tr_neg_neg)/(tr_neg_pos+tr_neg_neg+fls_neg_pos+fls_neg_neg)
    mic_fpr =
(fls_pos_pos+fls_pos_neg)/(fls_pos_pos+fls_pos_neg+tr_neg_pos+tr_neg_neg)
    mic_fdr =
(fls_pos_pos+fls_pos_neg)/(fls_pos_pos+fls_pos_neg+tr_pos_pos+tr_pos_neg)
    mic_f1 =
(2*precision_pos*recall_pos+2*precision_neg*recall_neg)/(precision_pos+recall_pos
+precision_neg+recall_neg)
    mic_f2 =
(5*precision_pos*recall_pos+5*precision_neg*recall_neg)/(4*precision_pos+recall_p
os+4*precision_neg+recall_neg)

    print("---------")
    print("Macro Statistics")
    print("Macro Precision: " + str(mac_precision))
    print("Macro Recall: " + str(mac_recall))
    print("Macro NPV: " + str(mac_npv))
    print("Macro FPR: " + str(mac_fpr))
```

```
    print("Macro FDR: " + str(mac_fdr))
    print("Macro F1: " + str(mac_f1))
    print("Macro F2: " + str(mac_f2))
    print("---------")
    print("Micro Statistics")
    print("Micro Precision: " + str(mic_precision))
    print("Micro Recall: " + str(mic_recall))
    print("Micro NPV: " + str(mic_npv))
    print("Micro FPR: " + str(mic_fpr))
    print("Micro FDR: " + str(mic_fdr))
    print("Micro F1: " + str(mic_f1))
    print("Micro F2: " + str(mic_f2))
    print("----------")


# In[4]:


def accuracyCalc(pred, y):
    sampleSize = y.shape[0]
    true_cnt = 0
    for i in range(sampleSize):
        if(pred[i] == y[i]):
            true_cnt += 1
    accuracy = true_cnt / sampleSize
    return accuracy


# In[5]:


def chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i+n]


# In[6]:


#import data
x_train = pd.read_csv('question-2-train-features.csv', header=None, sep=',',
names=["month","hr",

"weekday","weathersit",

"temp","atemp","hum",

"windspeed"])
y_train = pd.read_csv('question-2-train-labels.csv', header=None, sep=',',
names=["bikes"])
x_test = pd.read_csv('question-2-test-features.csv', header=None, sep=',',
```

```
names=["month","hr",

"weekday","weathersit",

"temp","atemp","hum",

"windspeed"])
y_test = pd.read_csv('question-2-test-labels.csv', header=None, names=["bikes"])

x_train_arr = np.asarray(x_train)
y_train_arr = np.asarray(y_train)
x_test_arr = np.asarray(x_test)
y_test_arr = np.asarray(y_test)
print(x_train.head(5))
print(y_train.head(5))
print(x_train.shape)
print(x_test.head(5))
print(y_test.head(5))
print(x_test.shape)


# In[7]:


#discretizing the labels
#find mean of labels
all_label = pd.concat([y_test, y_train], axis=0, sort = True)
label_mean = np.mean(all_label.values)
print(label_mean)
#discretizing
for i in range(y_test.shape[0]):
    if(y_test_arr[i] >= label_mean):
        y_test_arr[i] = 1
    else:
        y_test_arr[i] = 0
y_test = pd.DataFrame(y_test_arr)
print(y_test.shape)
for i in range(y_train.shape[0]):
    if(y_train_arr[i] >= label_mean):
        y_train_arr[i] = 1
    else:
        y_train_arr[i] = 0
y_train = pd.DataFrame(y_train_arr)
print(y_train.shape)


# # Question 4.1
# ## Batches for Cross Validation

# In[8]:
```

```
#set the rng seed
np.random.seed(0)
#set cross validation parameters
k=10
fold_size = x_train_arr.shape[0]/k
#shuffle the dataset
idx = np.random.permutation(len(x_train_arr))
x_tr_arr_shuf,y_tr_arr_shuf = x_train_arr[idx], y_train_arr[idx]
#divide the data into folds
x_cnks = list(chunks(x_tr_arr_shuf, int(fold_size)))
y_cnks = list(chunks(y_tr_arr_shuf, int(fold_size)))


# ## Cross Validation

# In[9]:


c_vals = np.array([10**-3, 10**-2,10**-1, 1.0,10**1, 10**2])
accs = np.ones((k,c_vals.shape[0]))
print(accs.shape)
for i in range(k):
    print("Fold: " + str(i))
    for c_ind in range(c_vals.shape[0]):
        print("C: " + str(c_vals[c_ind]))
        #set training feats
        cr_val_tr_x = list.copy(x_cnks)
        del cr_val_tr_x[i]
        cr_val_tr_x = np.asarray(cr_val_tr_x)
        cr_val_tr_x = cr_val_tr_x.reshape((12600,x_train_arr.shape[1]))
        #set training labels
        cr_val_tr_y = list.copy(y_cnks)
        del cr_val_tr_y[i]
        cr_val_tr_y = np.asarray(cr_val_tr_y)
        cr_val_tr_y = cr_val_tr_y.reshape((12600,y_train_arr.shape[1]))
        #set validation feats
        cr_val_x = x_cnks[i]
        #set validation labels
        cr_val_y = y_cnks[i]
        #train
        clf = svm.LinearSVC(C = c_vals[c_ind], dual=False)
        clf.fit(cr_val_tr_x,np.ravel(cr_val_tr_y))
        #test on validation
        pred = clf.predict(cr_val_x)
        #report accuracy
        acc =  accuracyCalc(pred,cr_val_y)
        print("Acc = ", acc)
        accs[i,c_ind] = acc
```

```
# In[10]:


#mean calculation
mean_v = np.mean(accs,axis=0)
print(mean_v)
#plot the mean cross val accuracy
plt.semilogx(c_vals,mean_v)
plt.title("Accuracy vs. C")
plt.xlabel("C Values")
plt.ylabel("Mean Cross Validation Accuracy")
plt.grid()
plt.show()


# In[11]:


#get the max c value
opt_c = c_vals[np.argmax(mean_v)]


# In[12]:


#test on the test set with optimal C
clf = svm.LinearSVC(C =opt_c, dual=False)
clf.fit(x_train_arr,np.ravel(y_train_arr))
pred = clf.predict(x_test_arr)
macroMicroAvr(pred,y_test_arr)


# # Question 4.2

# In[13]:


gamma_vals = np.array([2**-4, 2**-3, 2**-2, 2**-1, 2**0, 2**1])
accs = np.ones((k,gamma_vals.shape[0]))
print(accs.shape)
for i in range(k):
    print("Fold: " + str(i))
    for gamma_ind in range(gamma_vals.shape[0]):
        print("Gamma: " + str(gamma_vals[gamma_ind]))
        #set training feats
        cr_val_tr_x = list.copy(x_cnks)
        del cr_val_tr_x[i]
        cr_val_tr_x = np.asarray(cr_val_tr_x)
        cr_val_tr_x = cr_val_tr_x.reshape((12600,x_train_arr.shape[1]))
        #set training labels
        cr_val_tr_y = list.copy(y_cnks)
```

```
        del cr_val_tr_y[i]
        cr_val_tr_y = np.asarray(cr_val_tr_y)
        cr_val_tr_y = cr_val_tr_y.reshape((12600,y_train_arr.shape[1]))
        #set validation feats
        cr_val_x = x_cnks[i]
        #set validation labels
        cr_val_y = y_cnks[i]
        #train
        clf = svm.SVC(kernel='rbf',gamma = gamma_vals[gamma_ind],C=opt_c)
        clf.fit(cr_val_tr_x,np.ravel(cr_val_tr_y))
        #test on validation
        pred = clf.predict(cr_val_x)
        #report accuracy
        acc =  accuracyCalc(pred,cr_val_y)
        print("Acc = ", acc)
        accs[i,gamma_ind] = acc


# In[14]:


mean_v = np.mean(accs,axis=0)
print(mean_v)
plt.semilogx(gamma_vals,mean_v)
plt.title("Accuracy vs. Gamma")
plt.xlabel("Gamma Values")
plt.ylabel("Mean Cross Validation Accuracy")
plt.grid()
plt.show()


# In[15]:


#get the max gamma value
opt_gamma = gamma_vals[np.argmax(mean_v)]


# In[16]:


clf = svm.SVC(kernel='rbf',gamma=opt_gamma, C=opt_c)
clf.fit(x_train_arr,np.ravel(y_train_arr))
pred = clf.predict(x_test_arr)
macroMicroAvr(pred,y_test_arr)
```