HW-3 PCA - Fonts Dataset

Jose Rodriguez, Johnny Nino Ladino, Dayana Sosa; University of Houston

This paper was prepared as partial completion of the course MATH 6350 (Statistical Learning & Data Mining) taught during the Fall 2020 semester at University of Houston.

Background

This report will cover the questions presented by Dr. Azencott over the fonts dataset from the University of California Irvine Machine Learning Repository. The fonts dataset is made up of 153 files that are in a csv format. Each file represents a font and within each file there are 412 columns with varying number of rows depending on the file chosen. Each row represents a 'case' and each case describes numerically a digitized image of some specific character (letter or digit) type in a certain font. Each image has a 20 by 20 size and therefore 400 pixels which are represented by 400 columns. The three font files named "Courier", "Calibri" and "Times" will be used for this report. The first 12/412 columns describe the font and the other 400 columns represent the gray levels of the 400 pixels of a specific image. For this dataset, font type will be our target variable and the 400 columns describing the pixel gray levels will be our features. The aim of this report will be to implement PCA to classify the font type using the KNN algorithm to achieve a better accuracy. This report will aim to answer all questions presented in Dr. Azencott's homework instructions in a systematic way while analyzing each question and result to have an in depth understanding of this dataset. The underlying code sustaining this report will be created using python and the IDE of Jupyter notebook.

Methodology

The following methodology was used for developing the code and answering the questions in the document

1. Code will be written in python and python libraries such as pandas, numpy, scikit learn etc will be imported
2. Code and report will abide to the question numbers presented in Dr. Azencott's instructions
3. The report/code will use the PCA algorithm to implement dimension reduction on the fonts dataset before applying the KNN algorithm to the data.
4. The report/code will aim to compare the KNN results/model from Homework 2 with the results from treating the data with PCA.

Preliminary Treatment of Data

Each of the three chosen font files has the same column structure. To begin the analysis, data cleaning and preparation needs to be done on the dataset. Out of the 12 non-related pixel columns, we choose to keep only three of them. The three columns that will be kept are font, strength, and italic. Table 1 below explains what each of these columns represent. Any row that contained any missing numerical data was discarded from each of the three files.

FEATURE	DESCRIPTION
FONT	Same as file name. Describes what font the dataset belongs to.
STRENGTH	Column lists values either equal to 0.4 or to 0.7: in each row, strength = 0.4 for normal character; strength = 0.7 for bold character
ITALIC	Column lists values either equal to 0 or to 1; in each row, italic = 0 for normal character; italic = 1 for italic character;

Table 1: 3 Column Descriptions

The three files (referred to as classes) were then filtered down to only include rows containing a strength value of 0.4 and italic value of 0. This filtered the dataset to only images of "normal" characters. The "Calibri", "Courier" and "Times" classes will be referred to as CL1, CL2 and CL3 respectively. Their respective sizes are shown in Table 2 below.

CLASS	ROWS	COLUMNS
CL1 (CALIBRI)	4,768	403
CL2 (COURIER)	4,262	403
CL3 (TIMES)	4,805	403
TOTAL	**13,835**	

Table 2: Class Size

A full data set denoted as DATA was created through the union of the three class CL1, CL2 and CL3 and has a size of 13,835 rows. DATA will contain only the 400 features related to the gray level of the pixels to create a feature matrix. The basic machine learning tool covered in Dr. Azencott's class known as PCA & KNN will be used to attempt a rough automatic classification of DATA into the 3 classes CL1, CL2 and CL3.

Part 0

The means and standard deviations of DATA are computed to be used to standardize the dataset DATA. We standardize the dataset so values in certain features are not over weighed or under weighed when we are applying our classifier. We compute the mean and standard deviation of each column leaving us with $\hat{\mu}(mean) = \mu_{X1}, \dots, \mu_{X400}$ and $\hat{\sigma}(standard\ deviation) = \sigma_{X1}, \dots, \sigma_{X400}$. We then perform the following to rescale DATA:

$$\boldsymbol{SDATA(X_i, X_j) = \frac{DATA(X_i, X_j) - \mu_j}{\sigma_j}}$$

Equation 1: Standardized Matrix Equation

We store the standardized dataset DATA into a separate data frame named SDATA. From SDATA, we compute the correlation matrix, COR, of all 400 features. Table 3 below highlights the first 10 by 10 portion of the COR matrix.

	R0C0	R0C1	R0C2	R0C3	R0C4	R0C5	R0C6	R0C7	R0C8	R0C9
R0C0	1	0.92	0.79	0.63	0.45	0.28	0.17	0.08	0.02	-0.02
R0C1	0.92	1	0.89	0.70	0.51	0.31	0.18	0.08	0.01	-0.03
R0C2	0.79	0.89	1	0.87	0.64	0.42	0.26	0.15	0.06	-0.01
R0C3	0.63	0.70	0.87	1	0.84	0.60	0.40	0.25	0.12	0.03
R0C4	0.45	0.51	0.64	0.84	1	0.83	0.60	0.38	0.20	0.08
R0C5	0.28	0.31	0.42	0.60	0.83	1	0.83	0.58	0.35	0.19
R0C6	0.17	0.18	0.26	0.40	0.60	0.83	1	0.84	0.58	0.39
R0C7	0.08	0.08	0.15	0.25	0.38	0.58	0.84	1	0.84	0.61
R0C8	0.02	0.01	0.06	0.12	0.20	0.35	0.58	0.84	1	0.86
R0C9	-0.02	-0.03	-0.01	0.03	0.08	0.19	0.39	0.61	0.86	1

Table 3: 10 by 10 subset of Correlation Matrix

Part 1 – PCA

The PCA algorithm stands for Principal Component Analysis and is a dimension reduction technique that constructs new features as linear combinations of original features. PCA helps eliminate redundant information and attempts to capture as much information as possible by transforming large sets of features into a smaller set while preserving as much information as possible.

The eigenvalues and eigenvectors were computed from the correlation matrix, COR. There are a total of 400 eigenvalues and eigenvectors. The eigenvalues and eigenvectors will be used to determine the principal components of the data. The eigenvectors represent the principal components of COR and are essentially the directions of maximum variance in the feature space. The eigenvalues represent the magnitude of the principal components. Both the eigenvalues and eigenvectors are used to determine how many principal components should be selected for dimension reduction.

Step 0 involved standardizing the original 400 features in the dataset and creating a correlation matrix we call COR of all 400 features. Eigen decomposition was applied to the COR matrix to obtain the 400 eigenvalues and eigenvectors. The first 10 eigenvalues are displayed in Table 4 below. The eigenvectors calculated from COR were saved into a matrix "W".

Eigen Number (r)	Eigenvalue (Lr)
1	49.84
2	35.38
3	18.89
4	18.41
5	17.58
6	15.93
7	13.55
8	11.94
9	10.24
10	9.25

Table 4: First 10 Eigenvalues

In Figure 1, we display a plot of our eigenvalues against their corresponding eigen number. We see there is a decreasing relationship between r and eigenvalue. As r increases, the eigenvalues decrease dramatically and then level out. Eigenvalues that are close to zero represent components that could possibly be discarded. We are interested in discarding 400 – r components, where r will be the first "r" principal components that will explain 95% of our SDATA.

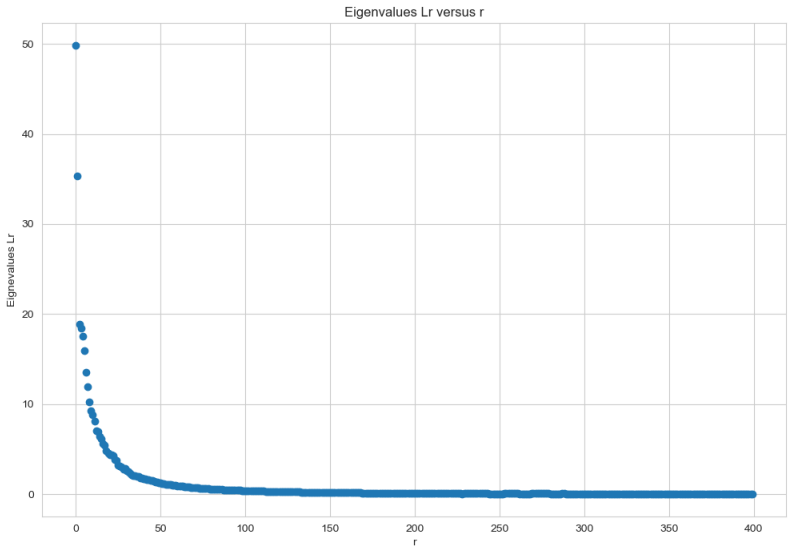

Figure 1: Eigenvalues Lr versus r

Proportion of Variance Explained describes the variance explained by the r principal components. PVE(r) is calculated by the following equation:

$$\boldsymbol{PVE(r) = \frac{1}{p}\sum_{r=1}^{400} L_r,}$$

Equation 2: Percentage of Explained Variance (PVE)

In Equation 2, p is the total number of features in our original standardized data set, SDATA, which in our case is 400. In Figure 2, we have a Scree Plot and Cumulative Scree Plot. In the Scree Plot, we see how much variance of the data is explained by each principal component. From the Scree Plot, the curve quickly decreases and then flattens out. This is because most of the variation of the data is explained by the first principal component, while each subsequent principal component explains less variance than the previous principal component. For the Cumulative Scree Plot, we summed the first r eigenvalues to obtain cumulative PVE for the first r components. Hence, the Cumulative Scree Plot increases as r increases.

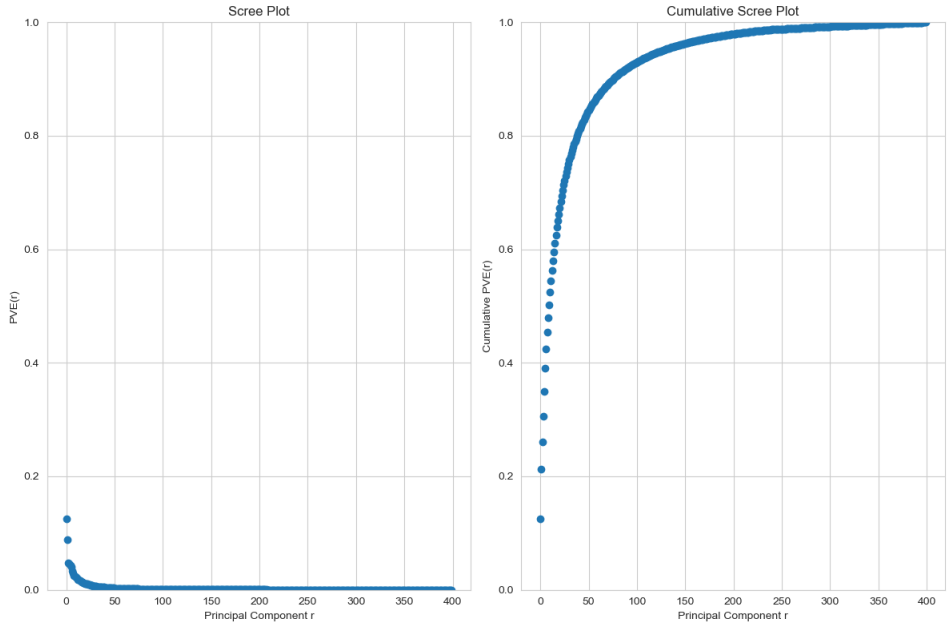

Figure 2: Scree Plot and Cumulative Scree Plot (PVE(r) vs r)

In Figure 3, the r that gives us a PVE of 95% is r = 126. This allows us to reduce the number of features without losing a large amount of information from our data. Now that we found the number of principal components to compress our dataset, we will perform the transformation on SDATA. To get the transformation, we apply the following matrix multiplication:

$$\boldsymbol{Y = W^T * SDATA^T}$$

The transpose of $\boldsymbol{Y}$ will display our new SDATA matrix, which will be a linear combination of our eigenvectors and standardized matrix.

In Table 5, we show the first 10 principal components with their respective first 4 features. Note that we will have Y(n) with n=1, ..., 13,835 and we will have a total of 126 principal components, based on our PCA.

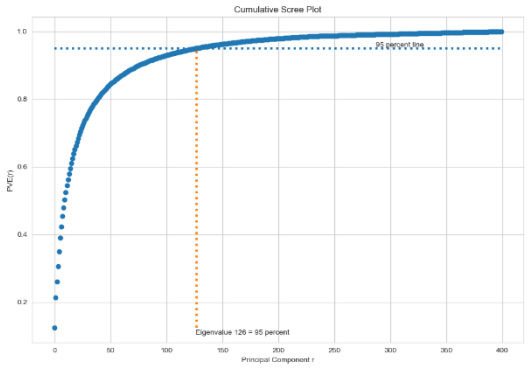

Figure 3: r Such that PVE(r) = 95%

Y(n)	PC1	PC2	PC3	PC4
1	4.50	2.20	2.38	-4.36
2	11.81	0.16	5.73	-2.46
3	-2.28	-3.50	2.29	-1.27
4	7.15	-3.59	6.77	-3.78
5	-2.90	0.95	-0.79	-4.07
6	-3.44	1.29	0.28	-2.48
7	-0.64	5.09	2.35	-3.20
8	-1.71	4.66	4.84	-0.37
9	-2.48	-0.05	4.41	-0.63
10	-1.36	10.67	-4.91	-0.56

Table 5: First 4 features of the First 10 Principal Components

In Figure 4, we plot the first four principal components into pairs for each case. For each Principal Component pair plot we display each point between them onto a scatter plot and then add color to the classes to demonstrate the spatial density of each class for that component. We can see in that our Principal Component 3 vs Principal Component 4 plot is the most concentrated in comparisons to the other plots. In the Principal Component 2 vs Principal Component 4 plot, we see that Calibri is more spread out compared to Courier. It is difficult to see clear concentration of Times cases, as these points are rather spread out in most of the plots. This could possibly indicate that the cases belonging in the Times class will be more difficult to classify and distinguish. In both the Principal Component 1 vs Principal Component 2 and Principal Component 1 vs Principal Component 4 plot, we see that Courier cases are more concentrated to the left of the Y-axis, indicating that Principal Component 1 can distinguish a large amount of Courier cases.

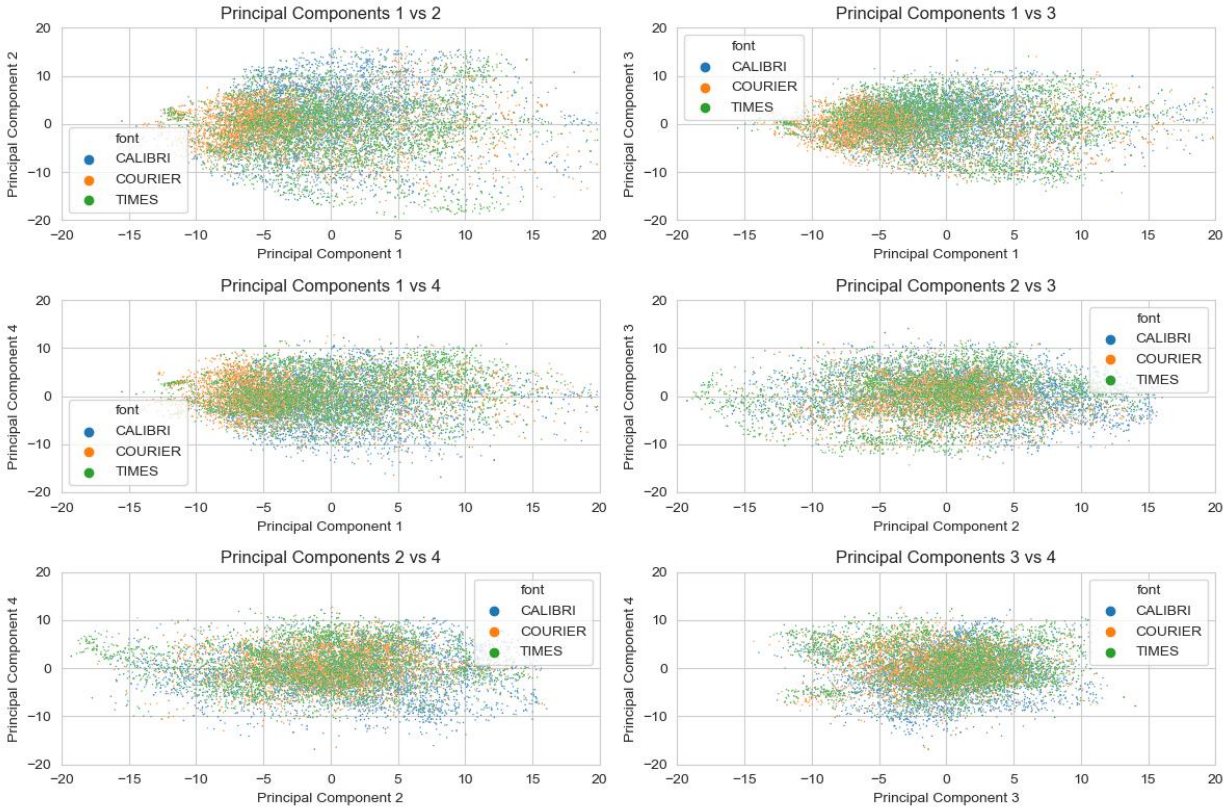

Figure 4: 6 Color Graphic Scatterplot Displays of the 3 Classes

Part 2 – KNN Homework 2 Comparison

We run the KNN algorithm for k = 5, 10, 15, 20, 30, 40, 50, 100, as we did in Homework 2, after implementing PCA. We compare the Test Accuracy results from KNN before and after implementing PCA in Figure 5. We see that both plots have similar behavior since Test Accuracy decreases as we increase k. The most obvious difference between the plots is seen in the Post-PCA plot when k is between 10 and 15. Here, we see that the slope is less steep then continues to decrease uniformly.

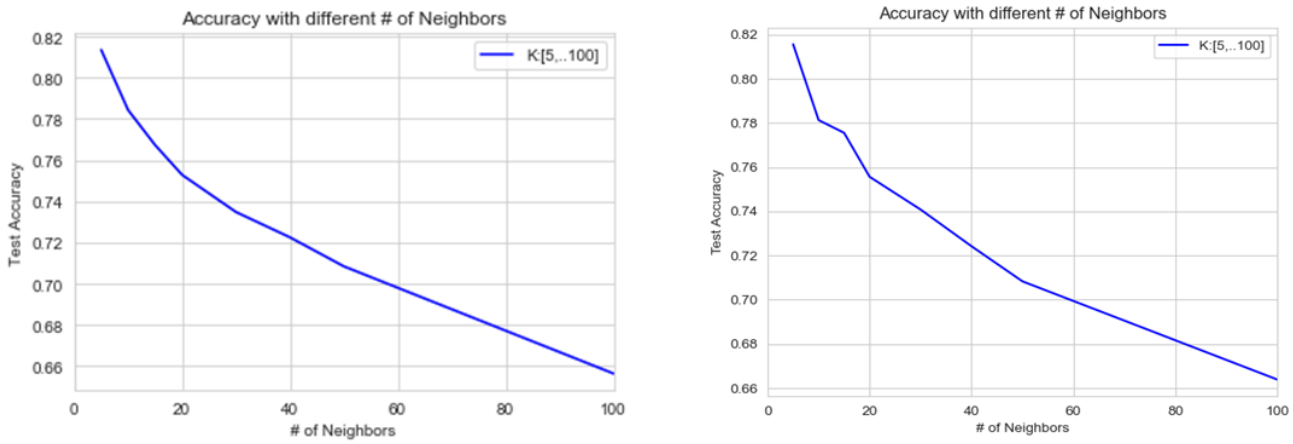

Figure 5: Comparison of Test Accuracy for Various v Values (Left: Pre-PCA, Right: Post-PCA)

We then rerun KNN on the set k = 5, 6, 7, 8, 9, 10. The plot on the left of Figure 6 displays our Train and Test accuracy scores before applying PCA. The plot on the right of Figure 6 displays our Train and Test accuracy scores after applying PCA. We see that both plots have very similar accuracy scores in both the Train and Test set and overall similar behavior in the plots. In Homework 2, we determined that k = 7 was the best number of neighbors that gave us the highest accuracy score without our model being overfitted. If we recall, k = 7 gave us an accuracy score of 85.6% on our train set and an accuracy score of 80.6% on our test set. Our delta between the train accuracy score and test score is 5.0%. We performed KNN after implementing PCA and compared our results for the same k values in Table 7.

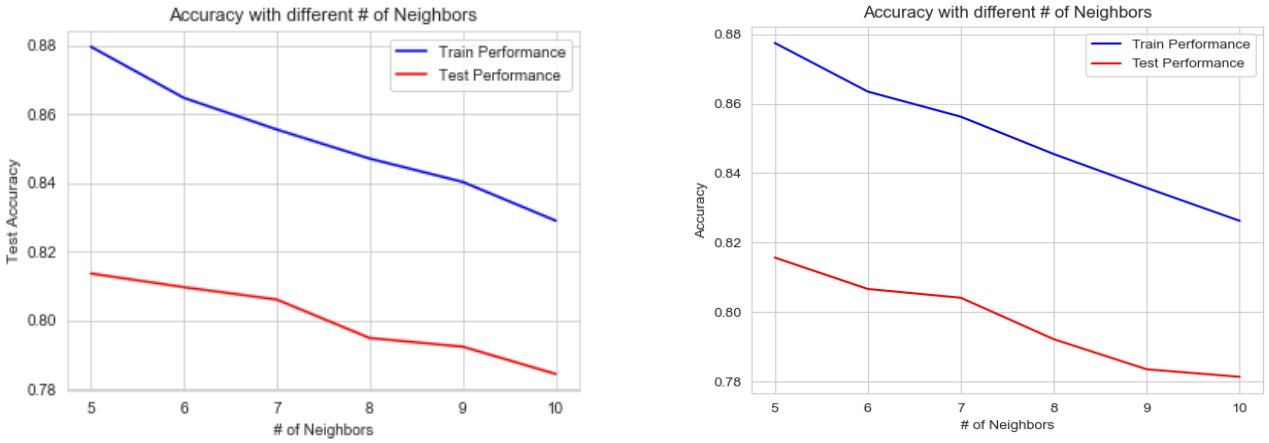

Figure 6: Comparison of Train and Test Accuracy k= 5-10 (Left: Pre-PCA, Right: Post-PCA)

Comparing our results between pre and post PCA, we see that all our scores slightly increase in the Post-PCA when k takes on values between 5 and 8. Looking at Table 7, we compare our train accuracy score and test accuracy score to our pre application of PCA. For post PCA, we get a train score of 85.9% and a test score of 80.7%. Our score improved by 0.03% and 0.01% respectively. Our deltas remain relatively small. However, we do not see a bounce back like we did in the pre PCA iteration of KNN.

k	Train Accuracy	Test Accuracy	Delta
5	0.880	0.814	0.066
6	0.865	0.810	0.055
7	0.856	0.806	0.050
8	0.847	0.795	0.052
9	0.840	0.792	0.048
10	0.829	0.784	0.045

Table 6: Training and Test Accuracy for Varying k values (Pre-PCA)

k	Train Accuracy	Test Accuracy	Delta
5	0.881	0.817	0.064
6	0.867	0.812	0.055
7	0.859	0.807	0.052
8	0.849	0.798	0.051
9	0.840	0.790	0.049
10	0.825	0.782	0.043

Table 7: Training and Test Accuracy for Varying k values (Post-PCA)

We produced the Train and Test Set Confusion matrices before and after implementing PCA to compare the results using kbest=7. Looking at the Train Set results, we see that performing PCA yielded slightly better results in correctly classifying Calibri and Courier cases. For the Times class, we see slightly better results when we do not perform PCA. We see similar patterns in the Test Confusion matrices when comparing results before and after implementing PCA. KNN

performed better after implementing PCA when classifying Calibri and Courier cases. However, KNN performs better at classifying Times when we do not implement PCA.

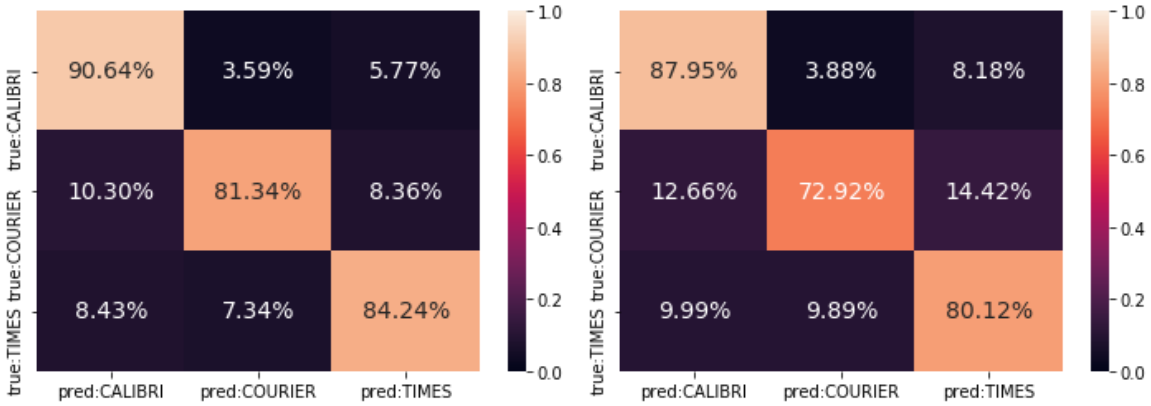

Figure 7: Train and Test Confusion Matrix (Pre PCA) for Kbest = 7

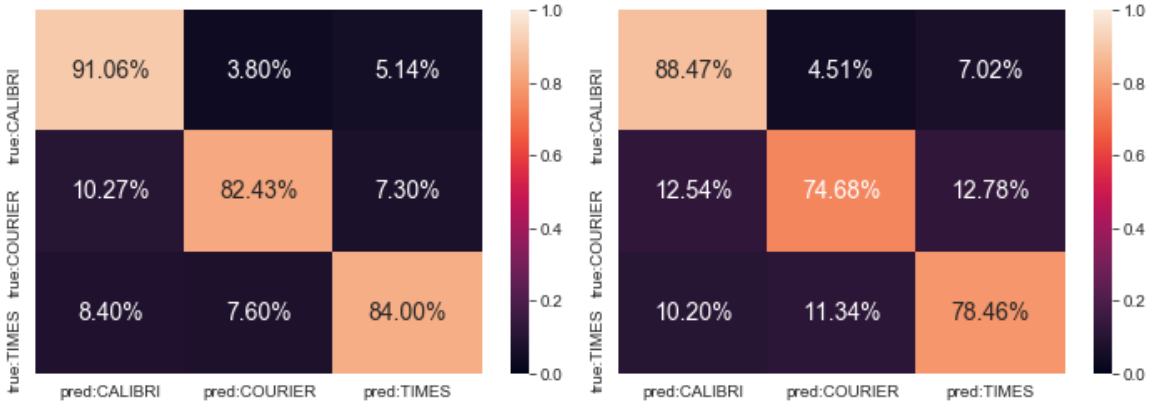

Figure 8: Train and Test Confusion Matrix (Post-PCA) for Kbest = 7

Table 8 below displays the Test Accuracy scores before and after implementing PCA and running KNN with kbest=7. We see that by using all 400 features of the original SDATA, we obtaing a Test Accuracy of 80.6%. After performing PCA, we compressed our number of features to 126 and obtained a Test Accuracy of 80.7%. While we did yield a slightly better performance of 0.01% we could say this is a very negligible difference. We would also like to point out that KNN does worse in our test set for the Times class after applying PCA. This could potentially be due to the fact that in Figure 4, our Principal Components are unable to separate Times cases into any particular quadrant and these first 4 components have the heaviest weight to them.

k	Pre-PCA	Post-PCA
7	0.806	0.807

The goal of PCA is to explain most of the variability in the data with a smaller number of variables than the original data set. We found a lower-dimensional representation of the data that captures as much of the information as possible and produced similar, if not better, results than our pre-PCA approach when implementing KNN to classify our dataset.

Summary

We performed pre-processing and filtering of the data to properly setup the data. Next, we normalized the features to be used in the classification. We then computed the correlation matrix, eigenvalues, and eigenvectors to perform a transformation on SDATA. We calculated PVE which told us that the first 126 components describe 95% of SDATA. We then used the new transformation to create our train and test sets. The KNN algorithm after implementing PCA was run in multiple iterations to compare our results with our previous iterations of KNN without PCA. We compared our kbest value of k=7 before and after implementing PCA and found that we had a slightly better performance when implementing PCA. Implementing PCA allowed us to reduce the number of features and decrease the runtime of our code while successfully applying the KNN algorithm to automatically classify arbitrary cases into one of the three font classes.

Acknowledgements

The authors gratefully acknowledge the guidance of Professor Robert Azencott.

Fonts Dataset Source

This dataset was taken from the University of California Irvine repository of machine learning datasets. The dataset can be found at following link: https://archive.ics.uci.edu/ml/machine-learning-databases/00417/

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013) *An Introduction to Statistical Learning with applications in R*, www.StatLearning.com, Springer-Verlag, New York

Work Distribution

Jose Rodriguez: 33%
Dayana Sosa: 33%
Johnny Nino Ladino: 33%

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import math

# Import text files of three specific fonts
Courier = pd.read_csv(r"C:\Users\Rodri\Desktop\Fall 2020\MATH 6350 - Statistical Learning & Data Mining\2. Homework\
HW-2\fonts\COURIER.csv")
Calibri = pd.read_csv(r"C:\Users\Rodri\Desktop\Fall 2020\MATH 6350 - Statistical Learning & Data Mining\2. Homework\
HW-2\fonts\CALIBRI.csv")
Times = pd.read_csv(r"C:\Users\Rodri\Desktop\Fall 2020\MATH 6350 - Statistical Learning & Data Mining\2. Homework\
HW-2\fonts\TIMES.csv")

################################################
#  ** Preliminary Treatment of the dataset **  #
################################################

# Discard the 9 columns below
# fontVariant, m_label, orientation, m_top, m_left, originalH, originalW, h, w
Courier.drop(columns = ['fontVariant', 'm_label', 'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h', 'w'], inplace=True
)
Calibri.drop(columns = ['fontVariant', 'm_label', 'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h', 'w'], inplace=True)
Times.drop(columns = ['fontVariant', 'm_label', 'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h', 'w'], inplace=True)

# any row containing missing numerical data should be be discarded
Courier.dropna(how='any', inplace = True)
Calibri.dropna(how='any', inplace = True)
Times.dropna(how='any', inplace = True)

# define then three CLASSES of images of "normal" characters as follows
# strngth =0.4 and italic = 0
CL1 =(Courier[(Courier['strength']==0.4) & (Courier['italic']==0)])
CL2 =(Calibri[(Calibri['strength']==0.4) & (Calibri['italic']==0)])
CL3 =(Times[(Courier['strength']==0.4) & (Times['italic']==0)])
# display their respective size n1, n2, n3
print('CL1 is made up  '+ str(CL1.shape) + " rows and columns and a total value of "+ str(CL1.size) + " variables")
print('CL2 is made up  '+ str(CL2.shape) + " rows and columns and a total value of "+ str(CL2.size) + " variables")
print('CL3 is made up  '+ str(CL3.shape) + " rows and columns and a total value of "+ str(CL3.size) + " variables")
# union of the three classes CL1 , CL2, CL3 and hence regroups N= n1 +n2 +n3 cases
DATA = pd.concat([CL1, CL2, CL3])

######################
#  **** PART 0 ****  #
######################

# mean and standard deviation
m = DATA.iloc[:,3:].mean()
sd = DATA.iloc[:,3:].std()
# standardizing the features - rescaled data matrix
SDATA = (DATA.iloc[:,3:] - m)/sd

##################
#  ** PART 1 **  #
```

```
##################

# Adding the font column to the standardized matrix
SDATA_ID = pd.concat([DATA.iloc[:,0], SDATA], axis=1)
# Creating CL1, CL2 and CL3 out of standardized matrix
SCL1 = SDATA_ID[SDATA_ID['font'] == 'COURIER']
SCL2 = SDATA_ID[SDATA_ID['font'] == 'CALIBRI']
SCL3 = SDATA_ID[SDATA_ID['font'] == 'TIMES']

## 1.0)
testCL1 = SCL1.sample(frac = 0.2, random_state = 0)
trainCL1 = SCL1[~SCL1.index.isin(testCL1.index)]

testCL2 = SCL2.sample(frac = 0.2, random_state = 0)
trainCL2 = SCL2[~SCL2.index.isin(testCL2.index)]

testCL3 = SCL3.sample(frac = 0.2, random_state = 0)
trainCL3 = SCL3[~SCL3.index.isin(testCL3.index)]

TESTSET  = pd.concat([testCL1, testCL2, testCL3])
TRAINSET = pd.concat([trainCL1, trainCL2, trainCL3])

# Creating X and Y values of KNN from Training and Test Datasets
X_Train = TRAINSET.iloc[:,1:]
Y_Train = TRAINSET.iloc[:,0]

X_Test = TESTSET.iloc[:,1:]
Y_Test = TESTSET.iloc[:,0]


#########################################################
###           ** HW - 3 Related **              ###
#########################################################

# Compute Correlation Matrix COR of all 400 features
COR = SDATA.corr()
# COR.iloc[:10,:10]
eigs = np.linalg.eig(COR)

# Compute the eigenvalues L1 > L2 > ... > L400 of COR
eig_value = eigs[0]
# eig_value[:10]

# Compute the matrix W of column eigenvectors for COR
W = eigs[1]

# Plot the eigenvalues Lr versus r (decreasing curve)
fsz=(12,8)
fig1, ax1 = plt.subplots(figsize=fsz)
ax1.scatter(range(SDATA.shape[1]), eig_value)
ax1.set_title('Eigenvalues Lr versus r')
ax1.set_ylabel('Eignevalues Lr')
ax1.set_xlabel('r')
plt.savefig('Eigenvalues Lr versus r.png')

# Plot the percentage of variance explained PVE(r) versus r (increasing curve)
PVE = eig_value / sum(eig_value)
cumsum_PVE = np.cumsum(PVE)

#DataFrame
```

```
num = list(range(401))
num =  num[1:]
cumsum_PVE_df = pd.DataFrame([cumsum_PVE], columns = num)
cumsum_PVE_df.iloc[:,:10] #first 10

# Plots
fig = plt.figure(figsize=fsz)
ax = fig.subplots(1,2)
# Scree Plot
plt.subplot(121)
plt.title('Scree Plot')
plt.xlabel('PVE(r)')
plt.ylabel('Principal Component r')
plt.scatter(range(SDATA.shape[1]), PVE)
plt.ylim(0, 1)
# Cumulative Scree Plot
plt.subplot(122)
plt.title('Cumulative Scree Plot')
plt.xlabel('PVE(r)')
plt.ylabel('Principal Component r')
plt.scatter(range(SDATA.shape[1]), cumsum_PVE)
plt.ylim(0, 1)

fig.tight_layout()
plt.savefig('Cumulative Scree Plot and Scree Plot.png')

# Compute r such that PVE(r) = 95 %
x = 0.95
r_x = len(cumsum_PVE[cumsum_PVE <= x])
# Cumulative Scree plot for PVE(r) = 95%
fig3, ax3 = plt.subplots(figsize=fsz)
ax3.set_title('Cumulative Scree Plot')
ax3.set_ylabel('PVE(r)')
txt1 = "%i percent line" % (x*100)
txt2 = "Eigenvalue %i = %i percent" % (r_x, x*100)
ax3.text(330,cumsum_PVE[r_x], txt1, horizontalalignment = 'right', verticalalignment = 'bottom')
ax3.text(r_x,.1, txt2, verticalalignment = 'bottom')
ax3.scatter(range(SDATA.shape[1]), cumsum_PVE)
ax3.plot(range(SDATA.shape[1]),[x]*SDATA.shape[1], linestyle = ':')
ax3.plot([r_x+1]*21,np.linspace(cumsum_PVE[0],x,21), linestyle = ':')
plt.savefig('Cumulative Scree Plot with 95% PVE Highlighted.png')

# Compute the principle components Y1(n) .. Yr(n) for each case n
PC = np.matmul(np.transpose(W), np.transpose(SDATA))

# Apply KNN using only the "r" new features Y1 .. Yr / Compare to HW2 results
from sklearn.decomposition import PCA

pca_model = PCA(n_components=r_x)
#pca_model = PCA(.95) # could do this too which gives r_x of 126
pca_model.fit(X_Train)
#pca.n_components_
X_Train_PCA = pca_model.transform(X_Train)
X_Test_PCA = pca_model.transform(X_Test)

################################
#### KNN Processes From HW-2 ###
################################

## 1.2)
```

```
k = [5, 10 , 15, 20, 30, 40, 50 , 100]
trainperf = []
testperf = []

# KNN Algo For Loop
for i in range(len(k)):
    # Fitting classifier to the Training set
    classifier = KNeighborsClassifier(n_neighbors = k[i])
    classifier.fit(X_Train_PCA, Y_Train)
    # Predicting the Test set results
    Y_Pred_Train = classifier.predict(X_Train_PCA)
    Y_Pred = classifier.predict(X_Test_PCA)
    # computing the % of correct classification on test set to a list & printing results
    trainperf.append(metrics.accuracy_score(Y_Train, Y_Pred_Train))
    testperf.append(metrics.accuracy_score(Y_Test, Y_Pred))
    #print("K = ", k[i], "Test Accuracy:", metrics.accuracy_score(Y_Test, Y_Pred))

#Plot the curve testperfK versus K to try to identify a best range [a < K <b] of values for the integer K
test_train_perf_dict = {'k': k, 'Train Accuracy': trainperf ,'Test Accuracy': testperf}
test_train_perf_df = pd.DataFrame(test_train_perf_dict)

#testperf_df.plot.line(x="k",y = "Test Accuracy")
#Line Plot
plt.figure()
sns.set_style("whitegrid")
sns.lineplot(x=k, y=testperf, color = 'blue', label = "K:[5,..100]")
plt.title('Accuracy with different # of Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Test Accuracy')
plt.xlim(0, 100)
plt.savefig('K Accuracy.png')

## 1.3)

k_N = [5, 6, 7 , 8, 9, 10]
trainperf_N = []
testperf_N = []

# KNN Algo For Loop
for i in range(len(k_N)):
    # Fitting classifier to the Training set
    classifier = KNeighborsClassifier(n_neighbors = k_N[i])
    classifier.fit(X_Train_PCA, Y_Train)
    # Predicting the Test set results
    Y_Pred_Train = classifier.predict(X_Train_PCA)
    Y_Pred = classifier.predict(X_Test_PCA)
    # computing the % of correct classification on test set to a list & printing results
    trainperf_N.append(metrics.accuracy_score(Y_Train, Y_Pred_Train))
    testperf_N.append(metrics.accuracy_score(Y_Test, Y_Pred))

#Plot the curve testperfK versus K to try to identify a best range [a < K <b] of values for the integer K
#Line Plot
f, ax = plt.subplots(1, 1)
sns.lineplot(x = k_N, y = trainperf_N, color="blue", label="Train Performance")
sns.lineplot(x = k_N, y = testperf_N, color="red", label="Test Performance")
plt.title('Accuracy with different # of Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Accuracy')
ax.legend()
plt.savefig('K Accuracy - Narrowed Down.png')
```

```
#create dataframe of new data
test_train_perf_dict_N = {'k': k_N, 'Train Accuracy': trainperf_N ,'Test Accuracy': testperf_N}
test_train_perf_df_N = pd.DataFrame(test_train_perf_dict_N)
#combine dataframes containing both sets of runs
test_train_perf_df_compiled = test_train_perf_df.append(test_train_perf_df_N, ignore_index =True)

#Choosing the "best" value kbest for the integer K
test_train_perf_df_compiled.sort_values(by=['Test Accuracy'], ascending= False, ignore_index=True, inplace=True)
#kbest = int(test_train_perf_df_compiled.iloc[0,:][0])

## 1.4)

## KNN Algo Run  - KBest Run
kbest = 7

# KNN Algo for Kbest
# Fitting classifier to the Training set
classifier = KNeighborsClassifier(n_neighbors = kbest)
classifier.fit(X_Train_PCA, Y_Train)
# Predicting both set results
Y_Pred_Train = classifier.predict(X_Train_PCA)
Y_Pred = classifier.predict(X_Test_PCA)

# Making the Confusion Matrix for Train & Test data
Ordinary_Distance_trainconf = confusion_matrix(Y_Train, Y_Pred_Train, labels = ['CALIBRI', 'COURIER', 'TIMES'], nor
malize = 'true')
Ordinary_Distance_testconf = confusion_matrix(Y_Test, Y_Pred, labels = ['CALIBRI', 'COURIER', 'TIMES'], normalize = 't
rue')
# Performance for both sets
Ordinary_Distance_trainperf = metrics.accuracy_score(Y_Train, Y_Pred_Train)
Ordinary_Distance_testperf = metrics.accuracy_score(Y_Test, Y_Pred)

# Visuals for Report
cmtx_o_train = pd.DataFrame(Ordinary_Distance_trainconf,
   index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
   columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])
cmtx_o_test = pd.DataFrame(Ordinary_Distance_testconf,
   index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
   columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])

# Confusion Matrix Plots For Ordinary Only
f, axes = plt.subplots(1, 2)
sns.heatmap(cmtx_o_train, annot=True, fmt = '.02%', annot_kws={"size": 8}, vmin = 0, vmax = 1, ax = axes[0])
sns.heatmap(cmtx_o_test, annot=True, fmt = '.02%', annot_kws={"size": 8}, vmin = 0, vmax = 1, ax = axes[1])
axes[0].set_title("Ordinary Distance - Train CM")
axes[1].set_title("Ordinary Distance - Test CM")
f.tight_layout()
plt.savefig("Confusion Matrix - Ordinary Distance (kbest).png")

# Compute the 6 color graphic scatterplots displays of your 3 classes in the 6 planes
# Y1,Y2    Y1,Y3    Y1,Y4     Y2,Y3     Y2,Y4       Y3,Y4       Interpetation?

# 3 variable resultant PCA to visualize
PC_3D = PC.T.iloc[:,0:4]
# PC_3D.iloc[:10,:] 10 by 10

# Color code based on true answer
color = DATA.iloc[:,0]
```

```
# 2D plots
fig = plt.figure(figsize=fsz)
ax = fig.subplots(3,2)
# Y1,Y2
plt.subplot(321)
plt.title('Principal Components 1 vs 2')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
sns.scatterplot(x=PC_3D.iloc[:,0], y=PC_3D.iloc[:,1], hue=color, s=1)
plt.ylim(-20, 20)
plt.xlim(-20, 20)
# Y1,Y3
plt.subplot(322)
plt.title('Principal Components 1 vs 3')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 3')
sns.scatterplot(x=PC_3D.iloc[:,0], y=PC_3D.iloc[:,2], hue=color, s=1)
plt.ylim(-20, 20)
plt.xlim(-20, 20)
# Y1,Y4
plt.subplot(323)
plt.title('Principal Components 1 vs 4')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 4')
sns.scatterplot(x=PC_3D.iloc[:,0], y=PC_3D.iloc[:,3], hue=color, s=1)
plt.ylim(-20, 20)
plt.xlim(-20, 20)
# Y2,Y3
plt.subplot(324)
plt.title('Principal Components 2 vs 3')
plt.xlabel('Principal Component 2')
plt.ylabel('Principal Component 3')
sns.scatterplot(x=PC_3D.iloc[:,1], y=PC_3D.iloc[:,2], hue=color, s=1)
plt.ylim(-20, 20)
plt.xlim(-20, 20)
# Y2,Y4
plt.subplot(325)
plt.title('Principal Components 2 vs 4')
plt.xlabel('Principal Component 2')
plt.ylabel('Principal Component 4')
sns.scatterplot(x=PC_3D.iloc[:,1], y=PC_3D.iloc[:,3], hue=color, s=1)
plt.ylim(-20, 20)
plt.xlim(-20, 20)
# Y3,Y4
plt.subplot(326)
plt.title('Principal Components 3 vs 4')
plt.xlabel('Principal Component 3')
plt.ylabel('Principal Component 4')
sns.scatterplot(x=PC_3D.iloc[:,2], y=PC_3D.iloc[:,3], hue=color, s=1)
plt.ylim(-20, 20)
plt.xlim(-20, 20)

fig.tight_layout()
plt.savefig("6 Color Graphic Scatterplots.png")
```