Font Recognition - kNN

In this report we are going to attempt to conduct Principal Component Analysis to reduce the feature space we analyze while preserving as much of the variance as we can and then we will use the k Nearest Neighbors algorithm to predict what classification any new data would fall into based on our test dataset. The data we will be using includes digitized images of characters that belong to a specific font and was provided by the University of California Irvine. The fonts used in our analysis are Calibri, Courier, and Times and have 400 features described by the gray level intensity of each pixel in a 20 x 20 image. Each font had its own data set with each feature labeled 'r0c0' – 'r19c19' and values between 0 and 255, and the dependent variable labeled 'font'.

Preliminary Treatment of the Data:

First, we discarded 9 columns from all 3 .csv files. We kept a total of 403 columns, (font, strength, italic, r0c0...r19c19), and discarded any other rows that had missing numerical data. Once this was completed, we defined three classes of images of normal characters as listed in Table 2.

Before we started analyzing the data, we reduced the three different sets of data size by only looking at "normal" characters defined by having a strength of 0.4 and italic value of 0, refer to Table 2. Then we combined the data sets together to create our final data set that we will be working on, with a size of (13835, 403).

"Normal" Characters	**Before Reduction**	**After Reduction**
COURIER (cl1)	(12229, 403)	(4262, 403)
CALIBRI (cl2)	(19068, 403)	(4768, 403)
TIMES (cl3)	(12730, 403)	(4805, 403)

Table 1: Data Sizes

Part 0:

Our first goal was to centralize and normalize the data. We started this by computing the mean, Equation 1, and standard deviation, Equation 2, for each feature. The means and standard deviations of the features are displayed in Figure 1, Figure 2, and Figure 3.

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N} \; where \; N = quantity \; of \; X's$$

Equation 1: Mean

$$\sigma = \sqrt{\frac{\left(X_i - \overline{X}\right)^2}{N}}$$

Equation 2: Standard Deviation

The mean of all the feature means came out to be 92.6, and we will call this mean_main. Looking at the histogram of the means, you may notice that as the mean values start to separate from mean_main the means frequency starts to decrease. The frequency of means creating a bell curve like shape around the mean_main like we might expected. The histogram of standard deviations seems to be telling us that the data for each feature is very spread out since there are so many high values for standard deviations. Figure 3 displays a scatter plot of the means vs. the standard deviations. The figure seems to be telling

us that there may be a positive linear relationship between the mean and standard deviation. This most likely has to do with white pixels in the image. There are certain pixel locations, probably the borders of the image, where it is more common to have white (1) pixels that decreases the mean and if most of the data is 1 our variance will decrease. On the other hand, there are certain pixel locations where it is common to have full intensity (255) pixels mixed with white (1) pixels, and this increases our mean and our variance.

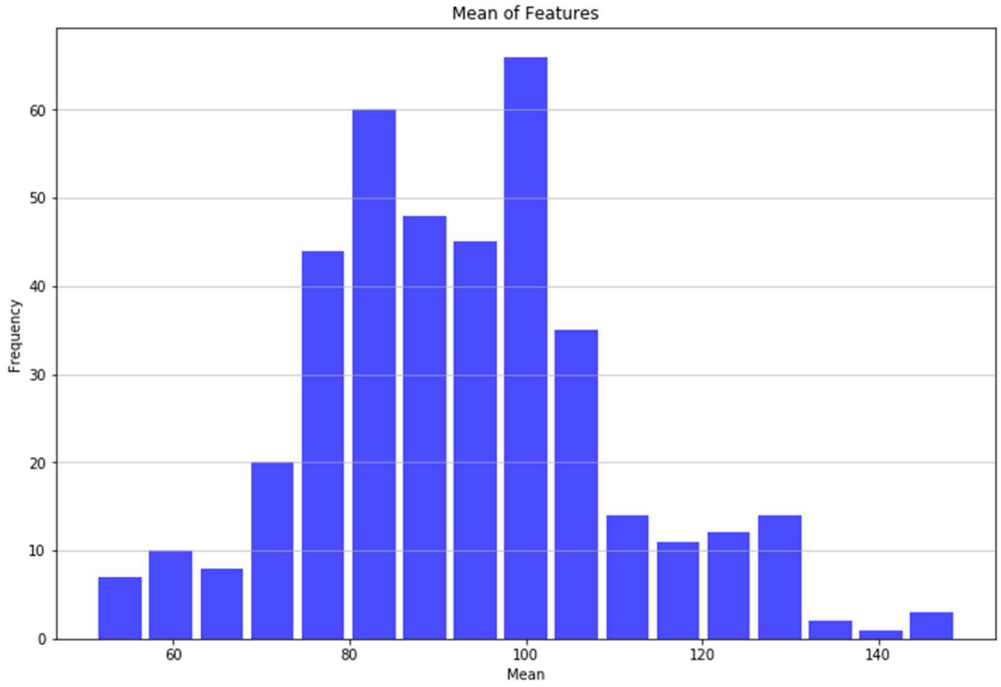

Figure 1: Histogram displaying the mean's of each feature.

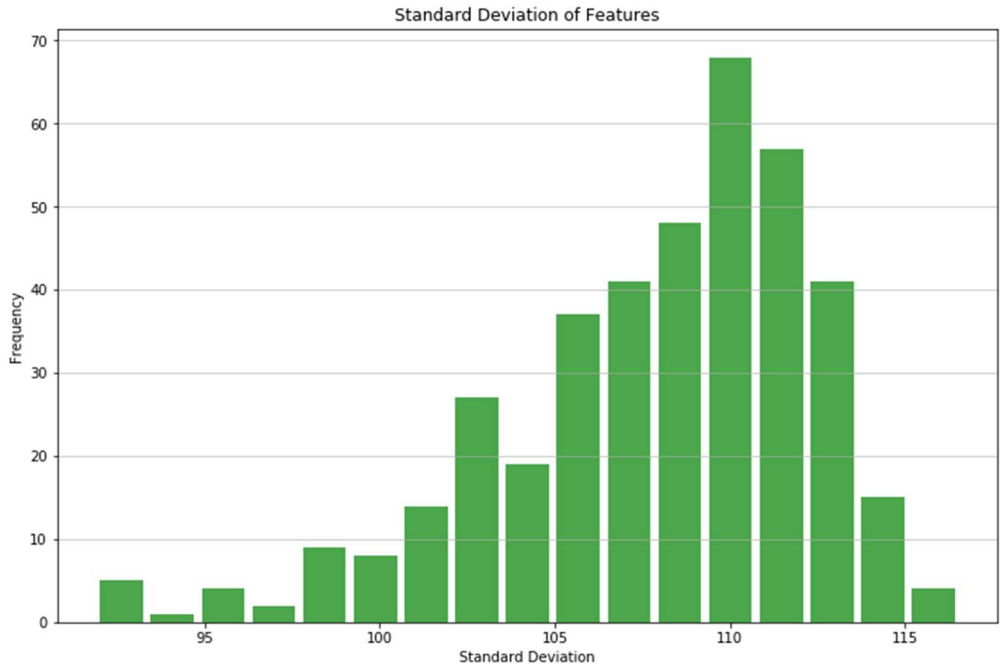

Figure 2: Histogram displaying the standard deviation's of each feature.

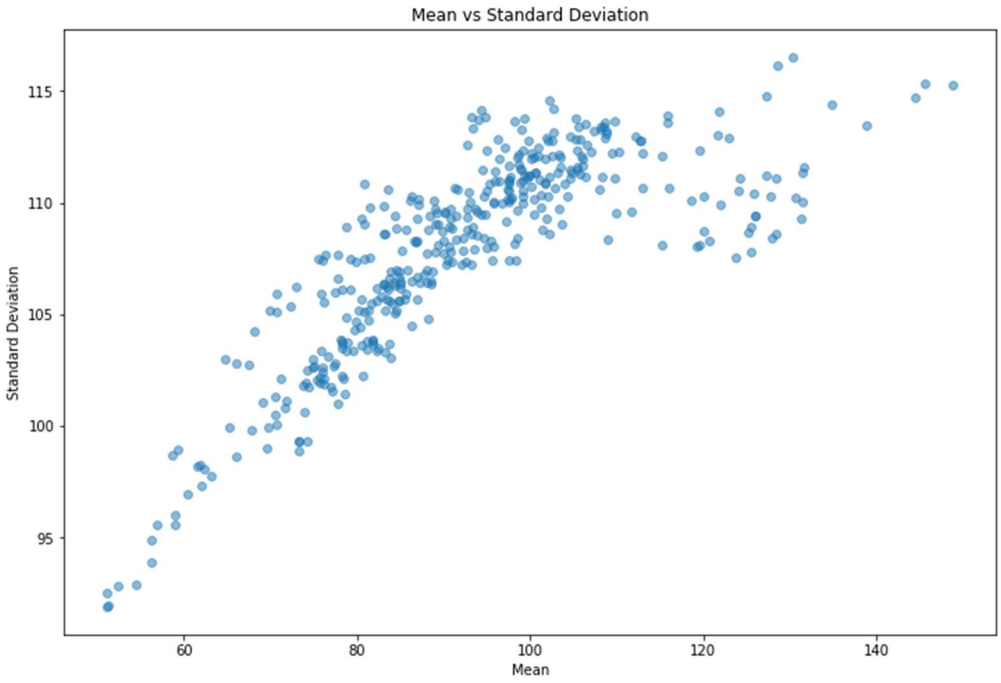

Figure 3: Scatter plot Mean vs. Standard Deviation

Then we used the mean and standard deviation for each feature to standardize the data by the following equation and vector operation.

$$Y_j = \frac{X_j - \overline{X}_j}{\sigma_j}$$

Equation 3: Standardizing a Value

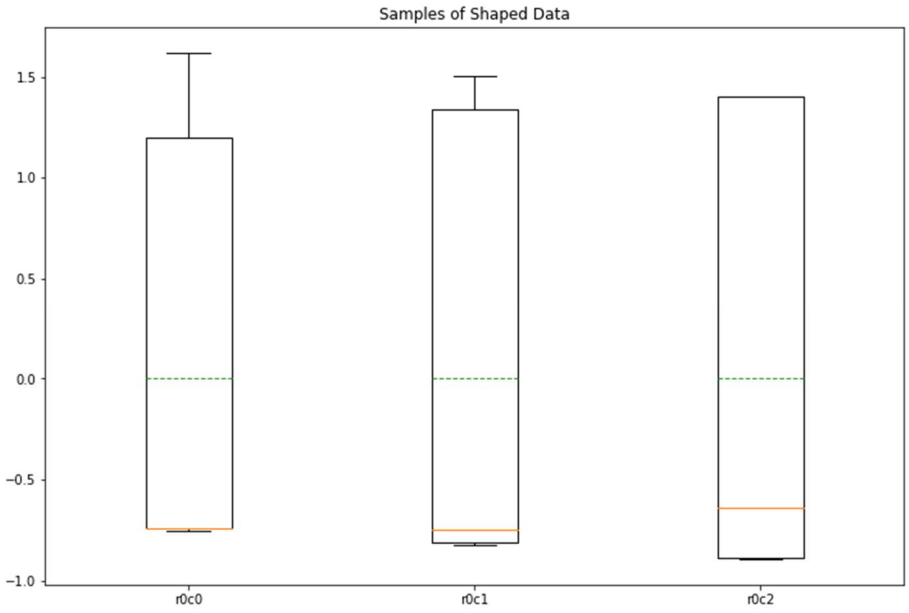

Figure 4: Box plot of the first three features after being standardized.

To give you an idea of what this did to the data, Figure 4 shows a box and whisker plot of the first 3 features in the standardized data. The green line displays the mean values for the feature. As a check we made sure that our means and standard deviations for each feature was 0 and 1, respectively.

Part 1:

Our next goal was to perform Principal Component Analysis (PCA). Using the standardized data, we calculated the correlation matrix for all of the features, using Equation 4. Note, that when $i = j$, then $corr(X_i, X_j) = 1$. Doing this for every feature in the data set creates a symmetric matrix of (400,400) where the diagonal from top left to bottom right values are equal to 1.

$$corr(X_i, X_j) = \frac{cov(X_i, X_j)}{\sigma_{X_i} * \sigma_{X_j}} \quad , \quad where \ -1 \leq corr(X_i, X_j) \leq 1$$

Equation 4: Correlation between X features.

$$cov(X_i, X_j) = \frac{\Sigma_{i=1}^{N}(X_i - \overline{X_i})(X_j - \overline{X_j})}{N}$$

Equation 5: Covariance between X and Y

When observing the correlation values, the sign on the value dictates if the two variables are directly or inversely related, with positive values being directly related, and negative values being inversely related. Values that are closer to 1 or -1 signify that the two variables are closer related, and those values that are closer to zero are less related.

Using the correlation matrix, we were able to determine the eigenvalues by using equation 6. Using this equation gives us the eigenvalues from greatest importance to lowest importance, displayed in Figure 5. Importance is determined by the size of the eigenvalue. Hence, the larger the value, the more important that eigenvalue is.

$$\det(Corr - \lambda I) = 0, \qquad I = Identity\ matrix, \qquad Corr = correlation\ matrix$$

Equation 6: Eigenvalue

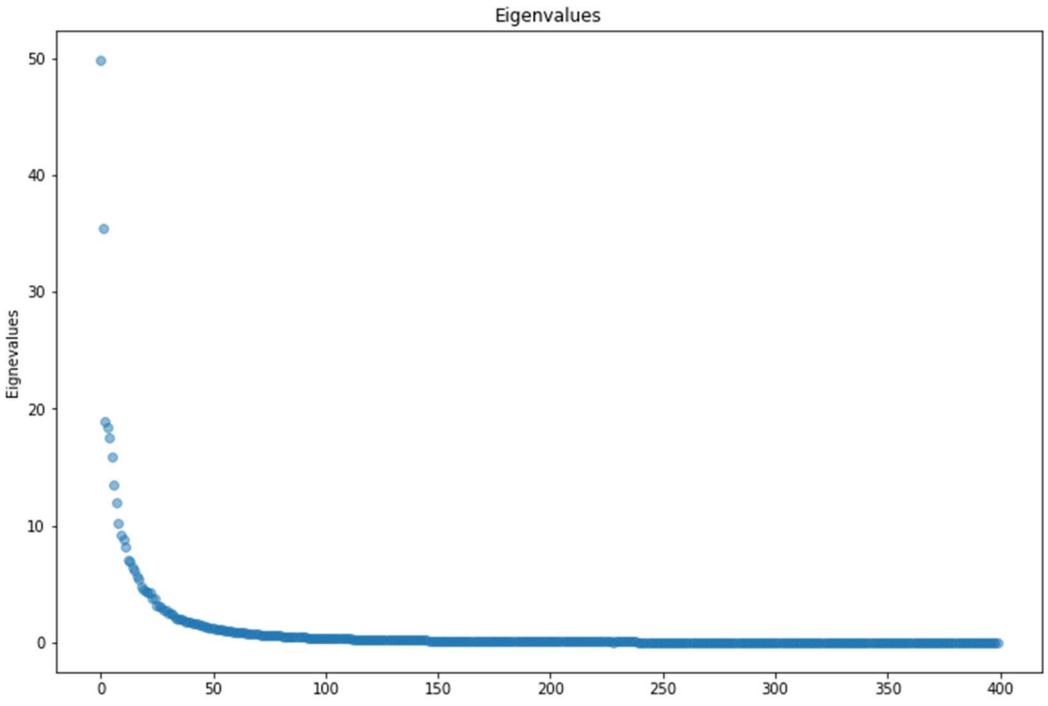

Figure 5: Eigenvalues from greatest to least importance2

All of the eigenvalues together account for all of the variance in the data, so by dividing a single eigenvalue by the sum of the total eigenvalues you get a ratio telling you how much variance is accounted for, refer to Equation 7.

$$R_i = \frac{\lambda_i}{\sum_{i=1}^{N} \lambda_i}$$

Equation 7: Ratio of Eigenvalues equation

By doing a cumulative ratio you can select the number of eigenvalues you want to use to reach the percent of variance accountability desired. In this case we want to create new features for the standardized data that account for more than 90% of the original data. If you refer to Figure 6, you can see that the cumulative ratio has been plotted along with a line displaying the 90% variance accountability and our eigenvalue index, point in yellow, that accounts for 90.10% of the variance. The index of that eigenvalue was 77. Note: if using Python you have to remember your index starts at 0 not 1.

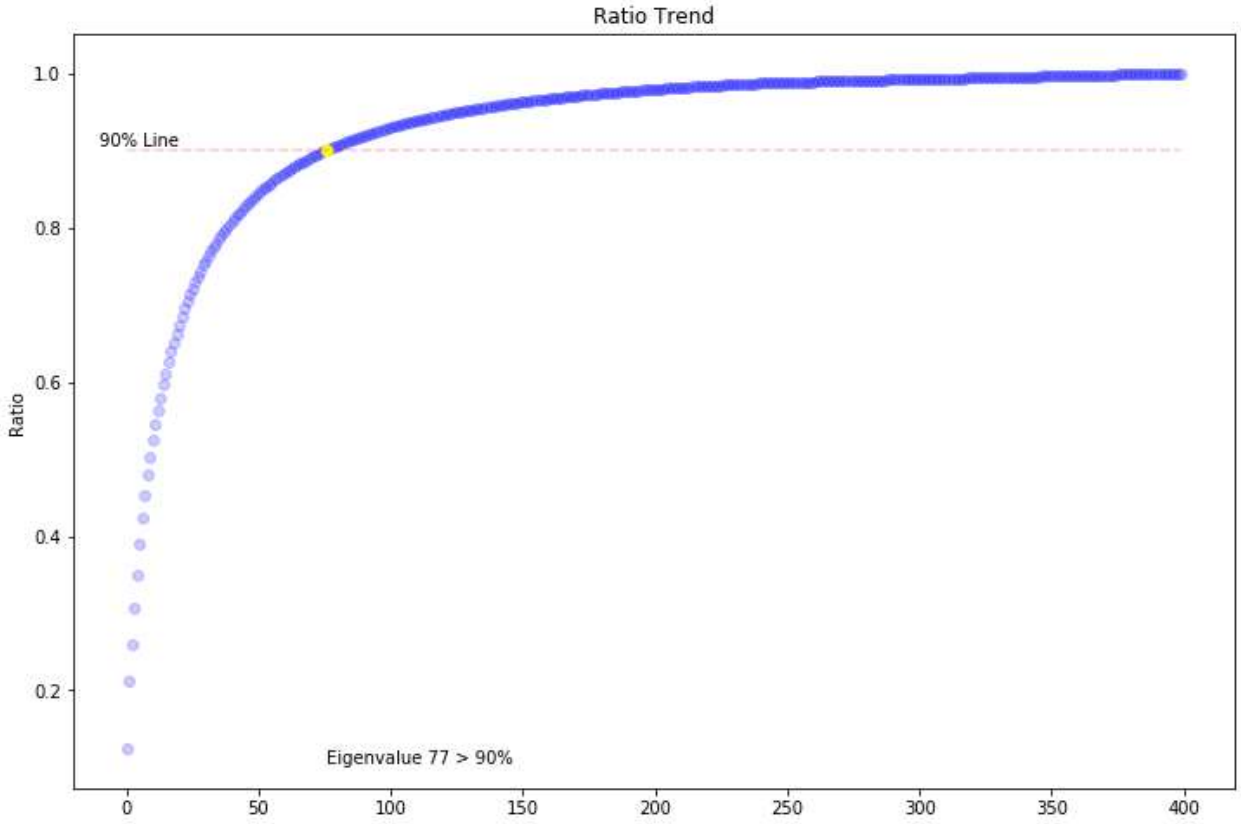

Figure 6: Cumulative Ratio Plot

Now you can use those 77 eigenvalues to calculate the 77 eigenvectors you will need to project your data into a new dimension by using Equation 8.

$$(Corr - \lambda I)X_{\lambda_i} = 0$$

Equation 8: Eigenvector

By doing the dot product of the current observation, i, and the X_λ you get a score for that observation in dimension λ, which is just that point for that dimension, represented by Equation 9. So, we can represent 90% of the variance in all the data by using each observation with the first 77 eigenvectors to map 77 new dimensions, or features, to represent the data.

$$scor_{i\lambda} = < E_i, X_{\lambda_i} >$$

$$<> = dot\ product, \qquad E_i = Observation\ i, \qquad X_{\lambda_i} = Eigenvector\ for\ \lambda_i$$

Equation 9: Score

It is pretty difficult to visually represent 77 dimensions, so we will project the old data into two and three dimensions by using the first two scores and then the first three scores respectively. Refer to Table 3 for the first 10 cumulative ratios to get an idea of how much variance is accounted for in the plots that we present.

Cumulative Ratio									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0.12	0.21	0.26	0.31	0.35	0.39	0.42	0.45	0.48	0.50

Table 3: Cumulative Ratio for first 10 Eigenvalues

In the 2 dimensional and 3 dimensional plots in Figure 7 and Figure 8, we will account for 21% and 26% of the variance in the data.

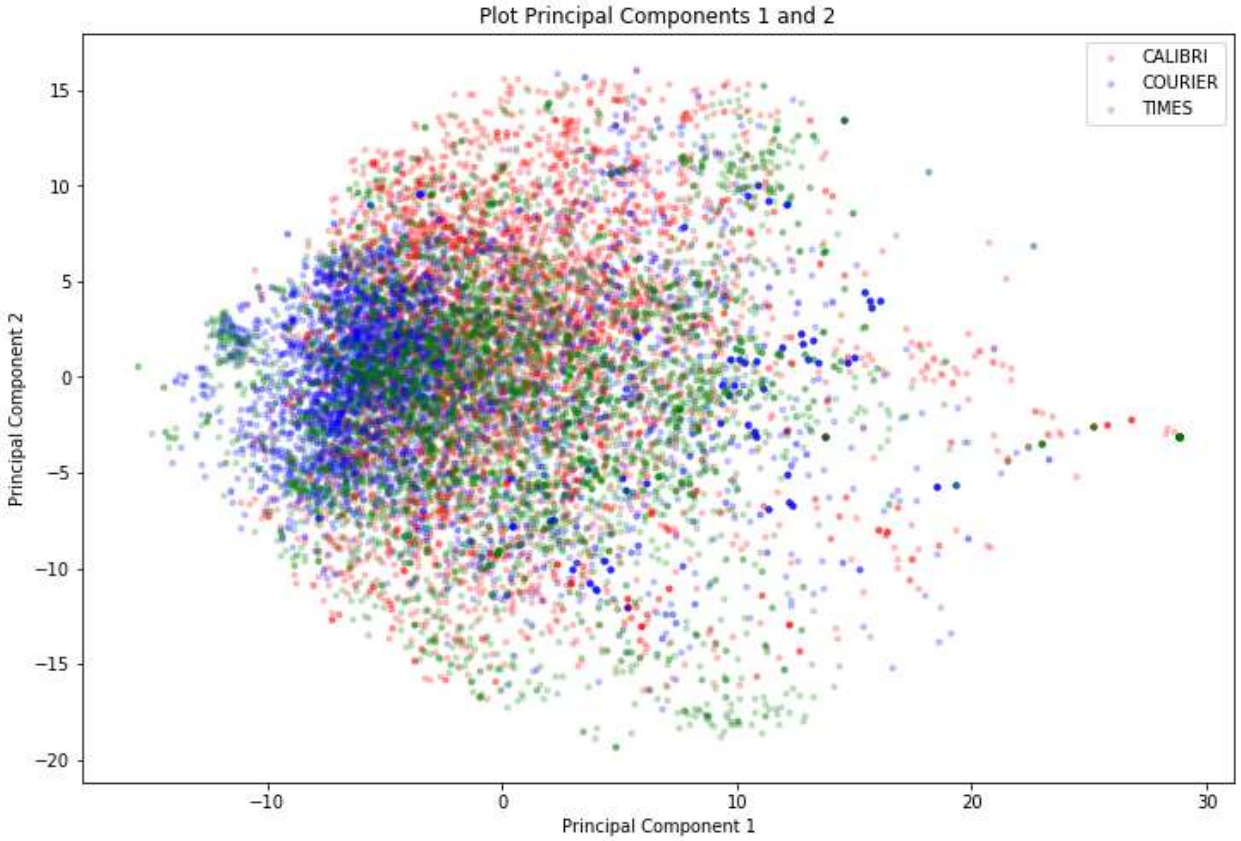

Figure 7: 2-D plot of Principal Component 1 vs Principal Component 2

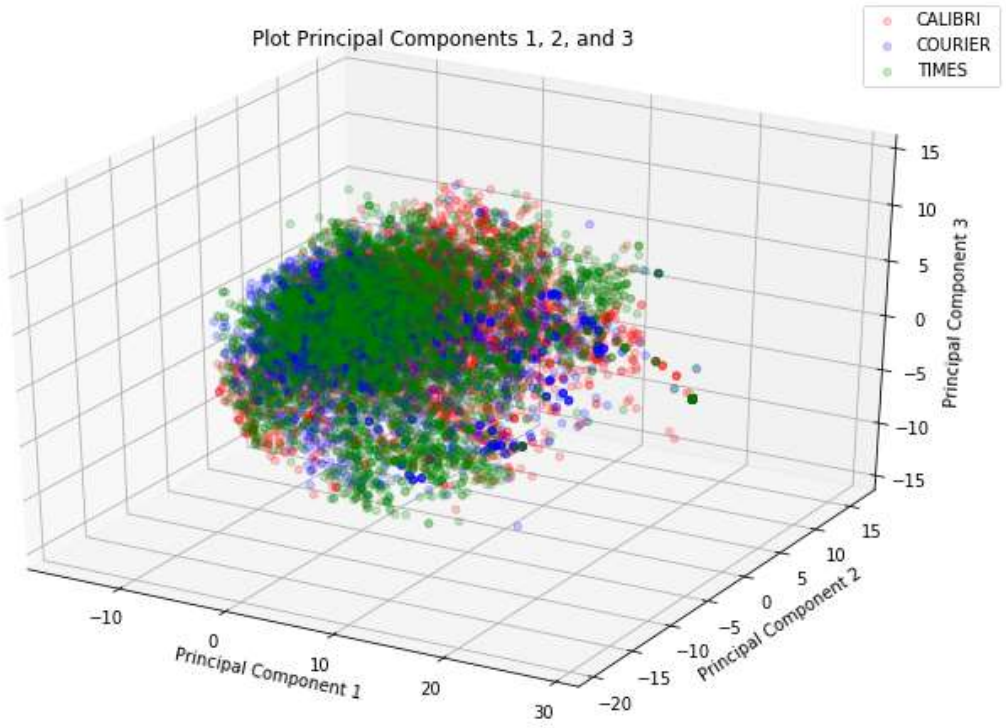

Figure 8: 3-D plot Calibri, Courier, and Times on the axes of Principal Components 1, 2, and 3

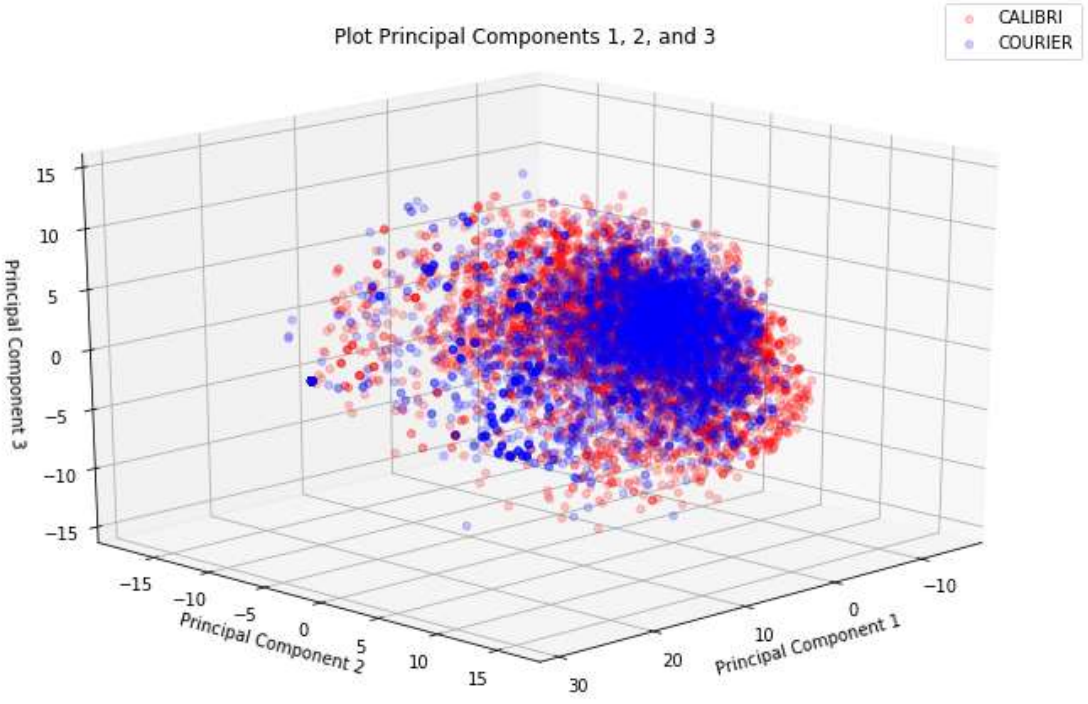

Figure 9: 3-D plot Calibri and Courier on the axes of Principal Components 1, 2, and 3

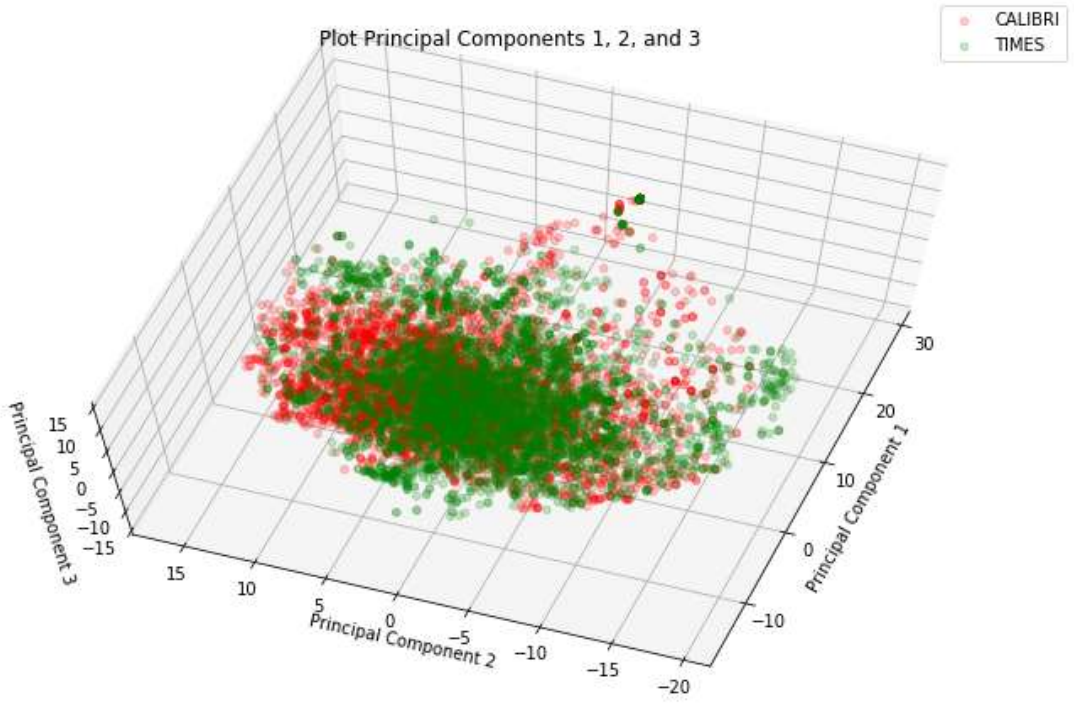

Figure 10: 3-D plot Calibri and Times on the axes of Principal Components 1, 2, and 3

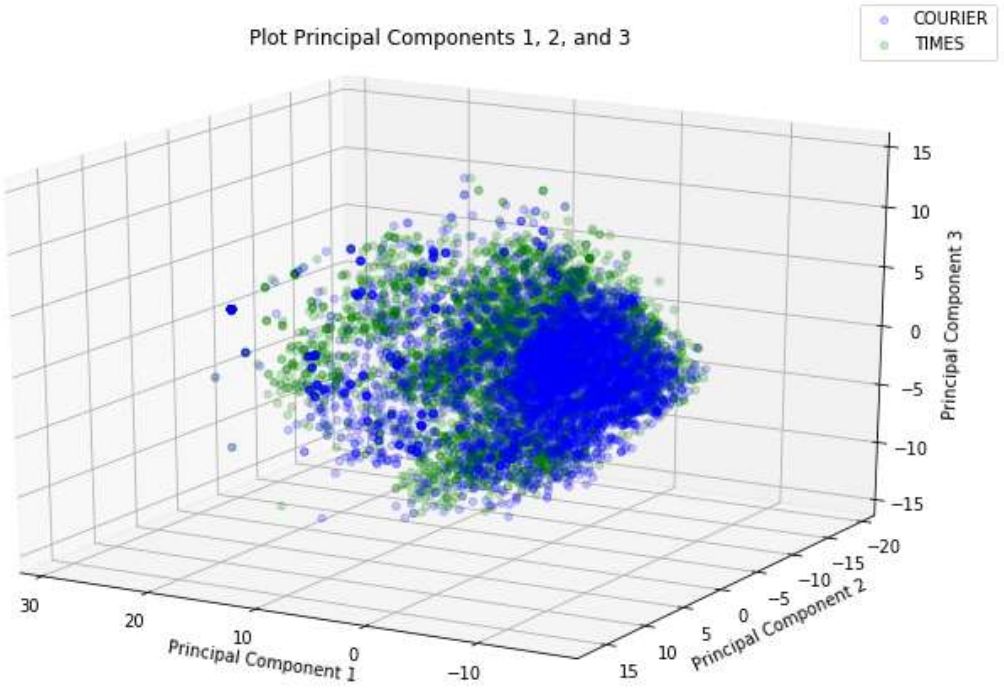

Figure 11: 3-D plot Courier and Times on the axes of Principal Components 1, 2, and 3

Part 2:

Once completing the PCA portion of our code, we created a KNN classification in order to see how well we could predict future fonts. To do this we split our entire standardized data into two sets, 80% of it being training data to train the automatic classification algorithm, and 20% being test data to see how well our algorithm did. In order to make sure we got 80% of each classification within the standardized data, we took 80% from each class for training and 20% from each class for testing, refer to part 2 of code.

K nearest neighbor works by plotting the data of investigation in an R^n space and calculates the Euclidean distance between that point to all the training data set. After calculating the distance, the algorithm gets the k nearest points and their associated cluster. Out of those k point, the point of investigation gets assigned to the classification with the most frequency. After creating the classification method, we used the learning dataset as the test dataset. After running the classification, we had 2 clusters associated to every point (2767 points). One cluster being the true value of that point and the second being the cluster our classification method determined that point should be. We then compared the results of the true classification and our classification to create the confusion matrix. An example of the confusion matrix is shown in table 4.

N=2767 k=5	**Predicted Calibri**	**Predicted Courier**	**Predicted Times**
Actual Calibri (954)	842	44	68
Actual Courier (852)	105	644	103
Actual Times (961)	97	99	765

Table 3: Confusion Matrix K=5

To analyze the confusion matrix, we can see the rows show what the true value of the associated cluster is and the columns show what the KNN classification determined it to be. On the diagonal it shows the number of cases that classified correctly and all other values were incorrect. Once we calculated the confusion matrix, we summed the diagonal and divided it by the total number of points in the matrix in order to determine the accuracy of our model. Another way to view this matrix is as the percent confusion matrix. You can see that each row sums up to 100%. For example, Calibri was correctly classified in Calibri for 88% of the test set and incorrectly placed in Courier for 5% and Times for 7% of the test set, summing to a total of 100%.

N=13835 k=5	**Predicted Calibri**	**Predicted Courier**	**Predicted Times**
Actual Calibri	88%	5%	7%
Actual Courier	12%	76%	12%
Actual Times	10%	10%	80%

Table 4: Percent Confusion Matrix, k=5

After running our model with a different number of neighbors ranging from 1 to 200, we get the results shown in figure 12.

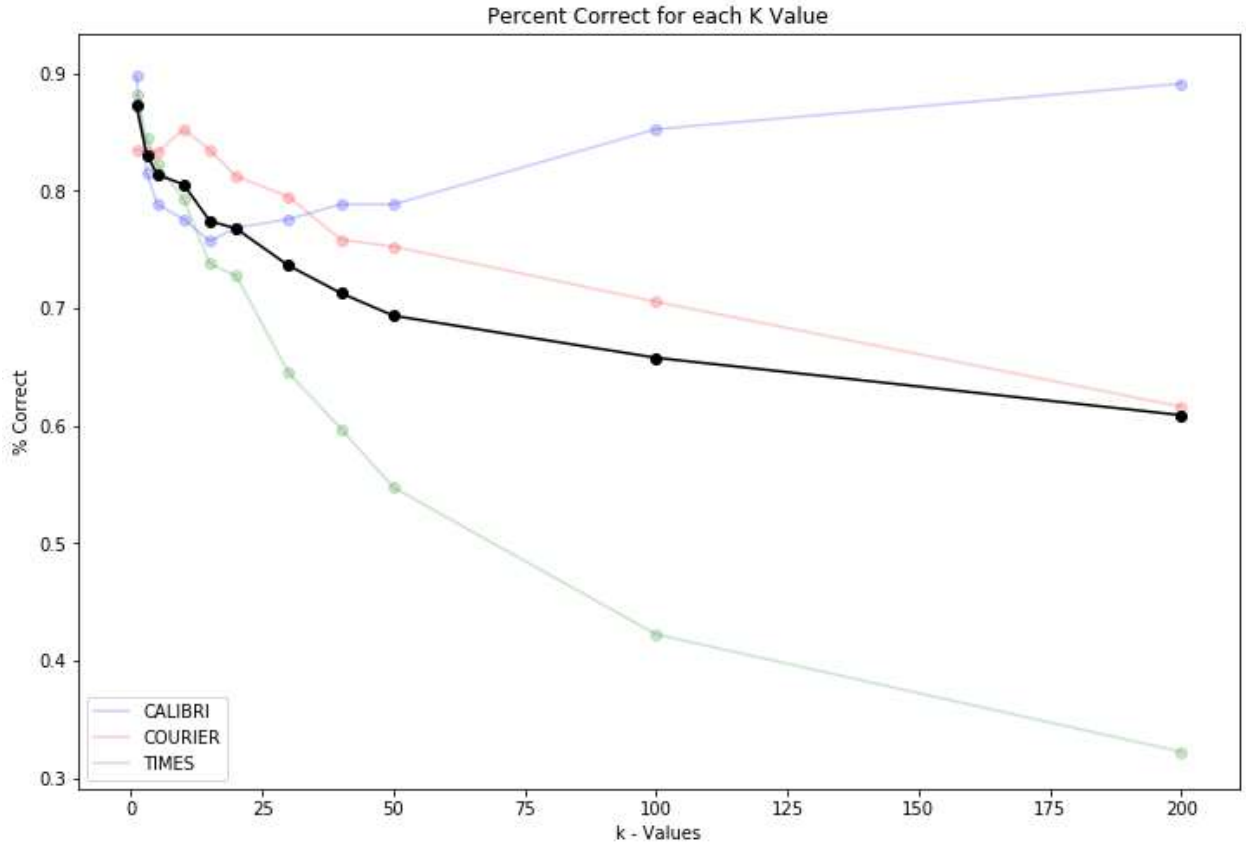

Figure 5: Accuracy vs number of Neighbors

Based on the kNN classification if we only look at the 5 nearest points to our point of investigation, we see a 81% correct interpretation, if we have 3 this number goes up to 83%, and if we only get the single nearest neighbor, there will be a 87% correct interpretation. It is typically not ideal to go down to a single nearest neighbor because it could lead to over fitting of the data, which could lead to even more errors in the classification. Based on our kNN predictions, it appears that 5 neighbors would be a reasonable quantity of neighbors due to their higher accuracy and are less likely to over-fit the data, like 1 or 2 neighbors may do. The faded lines in the background of Fig. 12 are the individual percent accuracies for each classification, labeled by color in the bottom left. The combination of these three accuracies make up the total accuracy line plotted in black. The thing to note here is that every run will result in slightly different percent accuracy values, creating a confidence interval that will vary based on your desired level of significance.

Preliminary Data Treatment	4 sec
Part 0	14 sec
Part 1	10 sec
Part 2	438 sec
Total Time	467 sec (7.8 min)

Code:

```
"""
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 23 04:37:58 2019

@author: Raul Paz
      Dustin Vasquez
      Stephen Rivera
"""
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NeighborhoodComponentsAnalysis as NCA
from sklearn.neighbors import KNeighborsClassifier as KNC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.utils import shuffle
import time

t0=time.time()
fsz=(12,8)

def get_data(file):
   """Create a function for loading the csv and editing the file the way we want
   for this excel format
   """

   import pandas as pd

   #Import csv as a Data Frame, drop columns do now want, drop rows with value na
   data = pd.read_csv(file)
   data = data.drop(['fontVariant', 'm_label',  'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h',
'w' ], axis = 1)
   data.dropna()
   print(data.shape)

   data = data.loc[(data.strength ==.4) & (data.italic == 0)]

   return data

#load the files and assign to class
```

```
file = r'C:\Users\Dustin\Desktop\Masters Program\Fall Semester\aaaaStatistical Learning and Data mining\Homework and Readings\Homework\HW2\CALIBRI.csv'
data = get_data(file)
cl2 = data.copy()

file = r'C:\Users\Dustin\Desktop\Masters Program\Fall Semester\aaaaStatistical Learning and Data mining\Homework and Readings\Homework\HW2\COURIER.csv'
data = get_data(file)
cl1 = data.copy()

file = r'C:\Users\Dustin\Desktop\Masters Program\Fall Semester\aaaaStatistical Learning and Data mining\Homework and Readings\Homework\HW2\TIMES.csv'
data = get_data(file)
cl3 = data.copy()

del data, file

#Print the sizes
n_cl1 = cl1.shape; print( n_cl1)
n_cl2 = cl2.shape; print( n_cl2)
n_cl3 = cl3.shape; print( n_cl3)
v = 400

#combine the three classes for full set of Data
data=pd.concat([cl1,cl2,cl3],axis=0)
data.index = range(len(data))
n_data = data.shape; print(n_data)
if n_data[0] == (n_cl1[0]+n_cl2[0]+n_cl3[0]): #check
   print('\nLooks good')

t1=time.time()
"""
Part 0
"""
## mean and standard deviation
m=data[data.columns[3:]].mean() #Mean
sd=data[data.columns[3:]].std() #Standard Deviation

## Plots
#Scatter Plot
plt.figure(figsize=fsz)
plt.scatter(m, sd, alpha=0.5)
plt.title('Mean vs Standard Deviation')
plt.ylabel('Standard Deviation')
plt.xlabel('Mean')
```

```
#Histogram
#mean
plt.figure(figsize=fsz)
n, bins, patches = plt.hist(x=m, bins='auto', color='blue', alpha=0.7, rwidth=0.85)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Mean')
plt.ylabel('Frequency')
plt.title('Mean of Features')
maxfreq = n.max()
#standard deviation
plt.figure(figsize=fsz)
n, bins, patches = plt.hist(x=sd, bins='auto', color='green', alpha=0.7, rwidth=0.85)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Standard Deviation')
plt.ylabel('Frequency')
plt.title('Standard Deviation of Features')
maxfreq = n.max()

## Centralize and stadardize the matrix
data_s = (data[data.columns[3:]]-m)/sd #Centralizing and Standardizing the Data
#print(data_s.mean(), data_s.std())   #Check the values , m=0 sd=1
""" just for visualization of centralized data
#plot of centralized and standardized data
fig1, ax1 = plt.subplots(figsize=fsz)
ax1.set_title('Samples of Shaped Data')
ax1.boxplot([data_s[data_s.columns[3]],data_s[data_s.columns[4]],data_s[data_s.columns[5]]],
meanline = True, showmeans=True, labels = ['r0c0','r0c1','r0c2'])
"""
t2=time.time()
"""
Part 1
"""
## correlation matrix
corr_m = data_s.corr()
eigs = np.linalg.eig(corr_m)

## eigen values
eig_value = eigs[0]
eig_vecto = eigs[1]
print(sum(eig_vecto[:,0]**2))
print('\n Sum of eig: ', sum(eig_value))

## Rj
ratio = np.cumsum(eig_value)/sum(eig_value)
```

```
# Where does Rj>.90
tratio_ind = np.where(ratio>.90) #seperates the values
r_ind=min(tratio_ind[0]) #gets the index
r_val = ratio[r_ind] #gets the value
del tratio_ind

## Plots
#plot eig_values
fig2, ax2 = plt.subplots(figsize=fsz)
ax2.set_title('Eigenvalues')
ax2.set_ylabel('Eignevalues')
ax2.scatter(range(len(eig_value)), eig_value, alpha=0.5)
#Plot Rj
fig3, ax3 = plt.subplots(figsize=fsz)
ax3.set_title('Ratio Trend')
ax3.set_ylabel('Ratio')
ax3.scatter(range(len(ratio)), ratio, c='b', alpha=.2)
ax3.scatter(r_ind, r_val, c='yellow', alpha=1)
ax3.plot(range(len(ratio)), [.90]*400, 'r--', alpha=.2)
ax3.text(20,r_val,'90% Line', horizontalalignment = 'right', verticalalignment = 'bottom')
ax3.text(r_ind,.1,'Eigenvalue 77 > 90%', verticalalignment = 'bottom')

## Plot seperation between top 3 Eigenvalues
#The normalized (unit “length”) eigenvectors, such that the column
#v[:,i] is the eigenvector corresponding to the eigenvalue w[i].
projected=np.dot(data_s,eig_vecto) #Eigenvalues are already transposed
projected = pd.concat([data[data.columns[0:3]],pd.DataFrame(projected)],axis=1)
data_sp = projected[projected.columns[0:6]]
#print(projected[projected.columns[3]].mean())
#Seperating indices for each font
cal=list(data_sp[data_sp['font'] == 'CALIBRI'].index)
cou=list(data_sp[data_sp['font'] == 'COURIER'].index)
tnr=list(data_sp[data_sp['font'] == 'TIMES'].index)
print((len(cal)+len(cou)+len(tnr)), len(data_sp)) #check
#2Dplot
plt.figure(figsize=fsz)
plt.title('Plot Principal Components 1 and 2')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.scatter(data_sp.iloc[cal,3], data_sp.iloc[cal,4], c='r' , alpha = .2, s=10)
plt.scatter(data_sp.iloc[cou,3], data_sp.iloc[cou,4], c='b', alpha = .2, s=10 )
plt.scatter(data_sp.iloc[tnr,3], data_sp.iloc[tnr,4], c='g', alpha = .2, s=10 )
plt.legend(['CALIBRI','COURIER' ,'TIMES'])
#plot three dimensional
fig=plt.figure(figsize=fsz)
```

```
ax4 = fig.add_subplot(111, projection='3d')
#ax.view_init(135, 135)
ax4.set_title('Plot Principal Components 1, 2, and 3')
ax4.set_xlabel('Principal Component 1')
ax4.set_ylabel('Principal Component 2')
ax4.set_zlabel('Principal Component 3')
ax4.scatter(data_sp.iloc[cal,3], data_sp.iloc[cal,4], data_sp.iloc[cal,5], c='r' , alpha = .2)
ax4.scatter(data_sp.iloc[cou,3], data_sp.iloc[cou,4], data_sp.iloc[cou,5], c='b', alpha = .2)
ax4.scatter(data_sp.iloc[tnr,3], data_sp.iloc[tnr,4], data_sp.iloc[tnr,5], c='g', alpha = .2)
ax4.legend(['CALIBRI','COURIER' ,'TIMES'])
#plot calibri and courier
fig=plt.figure(figsize=fsz)
ax5 = fig.add_subplot(111, projection='3d')
ax5.view_init(20, 45)
ax5.set_title('Plot Principal Components 1, 2, and 3')
ax5.set_xlabel('Principal Component 1')
ax5.set_ylabel('Principal Component 2')
ax5.set_zlabel('Principal Component 3')
ax5.scatter(data_sp.iloc[cal,3], data_sp.iloc[cal,4], data_sp.iloc[cal,5], c='r' , alpha = .2)
ax5.scatter(data_sp.iloc[cou,3], data_sp.iloc[cou,4], data_sp.iloc[cou,5], c='b', alpha = .2)
ax5.legend(['CALIBRI','COURIER'])
#plot calibri and times
fig=plt.figure(figsize=fsz)
ax6 = fig.add_subplot(111, projection='3d')
ax6.view_init(70, 200)
ax6.set_title('Plot Principal Components 1, 2, and 3')
ax6.set_xlabel('Principal Component 1')
ax6.set_ylabel('Principal Component 2')
ax6.set_zlabel('Principal Component 3')
ax6.scatter(data_sp.iloc[cal,3], data_sp.iloc[cal,4], data_sp.iloc[cal,5], c='r' , alpha = .2)
ax6.scatter(data_sp.iloc[tnr,3], data_sp.iloc[tnr,4], data_sp.iloc[tnr,5], c='g', alpha = .2)
ax6.legend(['CALIBRI','TIMES'])
#print(ax6.azim)
#plot courier and times
fig=plt.figure(figsize=fsz)
ax7 = fig.add_subplot(111, projection='3d')
ax7.view_init(20, 120)
ax7.set_title('Plot Principal Components 1, 2, and 3')
ax7.set_xlabel('Principal Component 1')
ax7.set_ylabel('Principal Component 2')
ax7.set_zlabel('Principal Component 3')
ax7.scatter(data_sp.iloc[cou,3], data_sp.iloc[cou,4], data_sp.iloc[cou,5], c='b', alpha = .2)
ax7.scatter(data_sp.iloc[tnr,3], data_sp.iloc[tnr,4], data_sp.iloc[tnr,5], c='g', alpha = .2)
ax7.legend(['COURIER' ,'TIMES'])
#print(ax7.azim)
```

```
t3=time.time()
"""
Part 2
"""
#Proportion out the section for test and train
tt_data = pd.concat([data[data.columns[0]],data_s],axis=1)
c1 = shuffle(tt_data[tt_data['font'] == 'COURIER']) #shuffles the first cluster
c1_pivot = int(round(n_cl1[0]*.8,0)) #round the pivot point
c1_train = c1[0:c1_pivot] #get train for classifiction
c1_test = c1[c1_pivot:] #get test for classification
c2 = shuffle(tt_data[tt_data['font'] == 'CALIBRI']) #shuffles the second cluster
c2_pivot = int(round(n_cl2[0]*.8,0)) #round the pivot point
c2_train = c2[0:c2_pivot] #get train for classifiction
c2_test = c2[c2_pivot:] #get test for classification
c3 = shuffle(tt_data[tt_data['font'] == 'TIMES']) #shuffles the thirds cluster
c3_pivot = int(round(n_cl3[0]*.8,0)) #round the pivot point
c3_train = c3[0:c3_pivot] #get train for classifiction
c3_test = c3[c3_pivot:] #get test for classification

#Get your X,Y Train and X,y Test
train = pd.concat([c1_train,c2_train,c3_train],axis=0) #all of train Data
train.index = range(len(train))
test = pd.concat([c1_test,c2_test,c3_test],axis=0) #all of test Data
test.index = range(len(test))

nbs=k=[1, 3, 5, 10 , 15, 20, 30, 40, 50 ,100,200] #neighbors

X_train, X_test = train[train.columns[1:]] , test[test.columns[1:]]
y_train, y_test = train[train.columns[0]] , test[test.columns[0]]

def KNN(X_train,y_train,X_test, nbs):
   pred=[]
   for i in range(len(nbs)):
      neigh = KNC(n_neighbors=nbs[i]) #Built in function that chooses the best method
      neigh.fit(X_train, y_train)
      preda=list(neigh.predict(X_test))
      pred.append(preda) #code with all of the predictions

   #print(pred)
   return pred

pred=KNN(X_train, y_train, X_test, nbs) #call the funciton

def percentage(pred, cal, cou, tnr, y_test):
   cal=len(y_test[y_test == 'CALIBRI'])
```

```
    cou=len(y_test[y_test == 'COURIER'])
    tnr=len(y_test[y_test == 'TIMES'])
    idx = y_test.index
    total_per = []
    cal_per = []
    cou_per = []
    tnr_per = []
    for j in range(len(pred)):
        correct = 0
        wrong = 0
        cal_wrong=0
        cou_wrong = 0
        tnr_wrong = 0
        for i in range(len(pred[j])):
            if (pred[j][i] == y_test[idx[i]]) == True:
                correct += 1
            else:
                wrong += 1
                if pred[j][i] == 'CALIBRI':
                    cal_wrong += 1
                    continue
                if pred[j][i] == 'COURIER':
                    cou_wrong += 1
                    continue
                if pred[j][i] == 'TIMES':
                    tnr_wrong += 1
                    continue
        total_per.append(correct/len(pred[j]))
        cal_per.append((cal-cal_wrong)/cal)
        cou_per.append((cou-cou_wrong)/cou)
        tnr_per.append((tnr-tnr_wrong)/tnr)
    return total_per, cal_per, cou_per, tnr_per

total_per, cal_per, cou_per, tnr_per = percentage(pred,cal,cou,tnr, y_test)
print(cal_per, cou_per, tnr_per)
print(total_per)

#Plot Accuracy
plt.figure(figsize=fsz)
plt.scatter(nbs,cal_per,c='b', alpha = .2)
plt.plot(nbs,cal_per,'b', alpha = .2)
plt.scatter(nbs,cou_per,c='r', alpha = .2)
plt.plot(nbs,cou_per,'r', alpha = .2)
plt.scatter(nbs,tnr_per,c='g', alpha = .2)
plt.plot(nbs,tnr_per,'g', alpha = .2)
```

```
plt.plot(nbs,total_per,'black', alpha = 1)
plt.scatter(nbs,total_per,c='black', alpha = 1)
plt.xlabel('k - Values')
plt.ylabel('% Correct')
plt.title('Percent Correct for each K Value')
plt.legend(['CALIBRI','COURIER' ,'TIMES'])

#Confusion Matrix
conf_m = confusion_matrix(y_test, pred[3])
class_report = classification_report(y_test, pred[0])

t4=time.time()
time = [(t1-t0)/60,(t2-t1)/60,(t3-t2)/60,(t4-t3)/60]
total_time = time[3]-time[0]
print('Total Time: ',total_time)
print('Times: ', time)
```