MATH 6350 Fall 2020 MSDS
Homework 3: PCA-KNN Model

Sara Nafaryeh
Tony Nguyen
Thomas Su

All authors played an equal part

Introduction

In this report, we will add to what we started in Homework 2 by looking into improving our KNN function using Principal Component Analysis, PCA. A brief reminder of what was done in Homework 2, we selected three font data sets consisting of digitized images of typed characters and determined using the KNN function in R to predict our best K.

The files were downloaded from a zip file from the following link:

https://archive.ics.uci.edu/ml/machine-learning-databases/00417/

The font types that we choose where: COMIC, BOOKMAN, and MONOTYPE. Each has 412 column features along with total observed cases of 2669, 2388, and 2388 respectively. Each case describes numerically a digitized image of some specific character typed in each of the fonts. The images have 20x20 = 400 pixel sizes, each with its own "gray level" indicated by an integer value of 0 to 255. All the fonts have 412 feature columns, where 400 of them describe the 400 pixels named:

{ r0c0, r0c1,r0c2, … , r19,c17, r19c18, r19c19}

"rLcM" = gray level image intensity for pixel in position **{Row L, Column M}**.

As stated earlier, in Homework 3, we will be more focused on using PCA on the 400-pixel features. Principal Component Analysis, PCA, is a dimensionality reduction method that reduces the dimensionality of a large data set, by capturing as much of the data information as possible into a smaller one that contains most of the information of the large set. When we say data has "high dimensionality", it means that the data set has a large number of features, much like in our data set with 400 features. We could analyze the data using 2D scatter plots for two features at a time, but that could mean

79,800 total plots. [p(p-1)/2, where p = 400 features] Not only is that time consuming, but very difficult to read. Therefore, the main idea of PCA is to reduce the number of variables of a data set, while preserving as much information as possible in one nice plot.

STEP 0: DATA Set up

Keeping our data set up the same as in homework 2, we will briefly go over this section as a refresher. The following steps are taken to get R ready as well as organize our data for the steps that follow. Other than the 400 columns that are associated with the pixels, the data set font files each have the following 12 names:

{ font, fontVariant, m_label, strength, italic, orientation, m_top, m_left, originalH, originalW, h, w }

Of these 12 we need to discard the following 9:

{fontVariant, m_label, orientation, m_top, m_left, originalH, originalW, h, w}

And keep the following 3: ***{font, strength, italic}*** as well as the 400 pixel columns named: ***{ r0c0, r0c1,r0c2, … , r19,c17, r19c18, r19c19}*** therefore we are left with 403 columns. After these steps are completed, we define three CLASSES on images of the "normal" character were we extract all the rows in which our three fonts have both strength of 0.4 and italic of 0:

CL1 = all rows of **comic.csv** file for which {strength = 0.4 and italic=0}

CL2 = all rows of **bookman.csv** file for which {strength = 0.4 and italic=0}

CL3 = all rows of **monotype.csv** file for which {strength = 0.4 and italic=0}

And left with the following row outputs:

```
> n1 = nrow(CL1)
> n2 = nrow(CL2)
> n3 = nrow(CL3)
> N = sum(n1, n2, n3)
> n1;n2;n3;N
[1] 597
[1] 667
[1] 667
[1] 1931
```

The respected row sizes for CLASS1, CLASS2, and CLASS3 are named

n1, n2, n3 = 597, 667, 667 and there sum → N = 1931 cases

We combine them all together into a data set named DATA using the function **rbind()** and see it has dimensions of 1931 rows and 403 columns: the 403 being font, strength, italic, +400 pixels we will call features X1, X2, … X400. Each such feature Xj is observed N times, and its N observed values are listed in the column "j" of DATA.

```
> DATA = rbind(CL1, CL2, CL3)
> str(DATA)
'data.frame':	1931 obs. of  403 variables:
 $ font    : chr  "COMIC" "COMIC" "COMIC" "COMIC" ...
 $ strength: num  0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 ...
 $ italic  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ r0c0    : num  1 1 1 1 1 1 255 255 255 1 ...
 $ r0c1    : num  1 1 1 1 1 1 255 255 213 1 ...
 $ r0c2    : num  1 1 1 1 1 1 255 255 213 1 ...
 $ r0c3    : num  1 1 1 1 1 1 255 255 213 1 ...
 $ r0c4    : num  1 1 1 1 1 1 255 255 213 1 ...
 $ r0c5    : num  1 1 1 1 1 1 255 255 213 1 ...
 $ r0c6    : num  1 1 1 175 86 1 255 255 213 1 ...
 $ r0c7    : num  1 17 1 255 255 33 255 255 213 1 ...
 $ r0c8    : num  48 86 1 255 255 215 255 255 213 1 ...
 $ r0c9    : num  83 166 83 255 255 255 255 255 213 1 ...
 $ r0c10   : num  169 206 255 255 255 154 255 255 213 1 ...
 $ r0c11   : num  169 255 255 255 255 9 255 255 213 1 ...
 $ r0c12   : num  201 255 255 255 255 1 255 255 213 1 ...
 $ r0c13   : num  255 255 255 175 86 1 255 255 213 1 ...
 $ r0c14   : num  255 255 255 1 1 1 255 255 213 1 ...
 $ r0c15   : num  255 126 198 1 1 1 255 255 213 73 ...
 $ r0c16   : num  255 9 120 1 1 1 255 255 213 255 ...
 $ r0c17   : num  255 1 14 1 1 1 255 255 213 255 ...
 $ r0c18   : num  251 1 1 1 1 1 255 255 213 255 ...
 $ r0c19   : num  169 1 1 1 1 1 255 255 255 255 ...
 $ r1c0    : num  1 1 1 1 1 1 255 255 213 1 ...
 $ r1c1    : num  1 1 1 1 1 1 54 255 1 1 ...
 $ r1c2    : num  1 1 1 1 1 1 29 255 1 1 …
 $ r1c3    : num  1 1 1 1 1 1 29 255 1 1 ...
 $ r1c4    : num  1 1 1 191 126 1 29 255 1 1 ...
 $ r1c5    : num  1 1 1 242 227 1 29 255 1 1 ...
 $ r1c6    : num  67 86 1 251 237 19 29 255 1 1 ...
 $ r1c7    : num  174 225 1 103 255 185 29 255 1 1 ...
 $ r1c8    : num  254 255 1 14 255 255 29 255 1 1 ...
  [list output truncated]
```

STEP 1: Standardization

We start by standardizing the 400 features. Once again we define the ***start_col*** and ***end_col*** to be the feature X1 starts and X400 ends respectfully. Now that we have created our [1931x403] ***DATA*** set table, we observe the mean, m = mean(X1)....mean(X400), and standard deviation, s = sd(X1) ....sd(400) for each of our pixel features 1 to 400. The following output shows the first 6 values of the mean and standard deviation using head() function in R:

```
> head(m)
    r0c0     r0c1     r0c2     r0c3     r0c4     r0c5
42.24392 58.08597 63.99845 64.72139 69.71362 77.50492
> head(s)
     r0c0      r0c1      r0c2      r0c3      r0c4      r0c5
 85.60722 100.04134 104.25599 101.38599 102.82648 105.25836
```

The most important step to do before running our PCA function is to standardize the data features in order to set everything at a mean of 0 and a standard deviation at 1. It is important to standardize because when we compare measurements they can have different units as one can observe in the table above. In the table below observe the variance of the data before and after standardization. Notice the variance of the DATA of X1 and X2 features before, and after when they all equal to 1. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bais. We scale the data and name it SDATA. Note that X is a data frame from our DATA where we start at feature X1 and end at feature X400.

```
> var(DATA[,4]); var(DATA[,5])
[1] 7328.596
[1] 10008.27
>X = DATA[,start_col: end_col]
> SDATA = scale(X)
> var(SDATA[,4]); var(SDATA[,5])
[1] 1
[1] 1
```

STEP 2: Correlation Matrix Computation

Finally, we compute the 400x400 correlation matrix COR of the 400 features X1, X2,...,...X400, and define it as r. The main idea of this step is to understand whether or not there is a correlation or relationship between the features. It also helps us identify if features are dependent on each other, biased, and or redundant. The correlation matrix is a p*p symmetric matrix where p is the number of dimensions in the data and within our data, we have 400. Since we have a large number of features, for our correlation matrix we will display the first 6 features in a 6x6 matrix. And have all 400x400 in a .csv file attached separately to this report labeled as ***Correlation_matrix_Nafaryeh_Nguyen_Su.csv.***

```
> R = cor(X)
> R[1:6,1:6]
          r0c0      r0c1      r0c2      r0c3      r0c4      r0c5
r0c0 1.0000000 0.8905004 0.7820298 0.7104029 0.5995487 0.4753507
r0c1 0.8905004 1.0000000 0.9238824 0.7981221 0.6309654 0.4742221
r0c2 0.7820298 0.9238824 1.0000000 0.8998703 0.6987910 0.5183781
r0c3 0.7104029 0.7981221 0.8998703 1.0000000 0.8758144 0.6706774
r0c4 0.5995487 0.6309654 0.6987910 0.8758144 1.0000000 0.8618115
r0c5 0.4753507 0.4742221 0.5183781 0.6706774 0.8618115 1.0000000
```

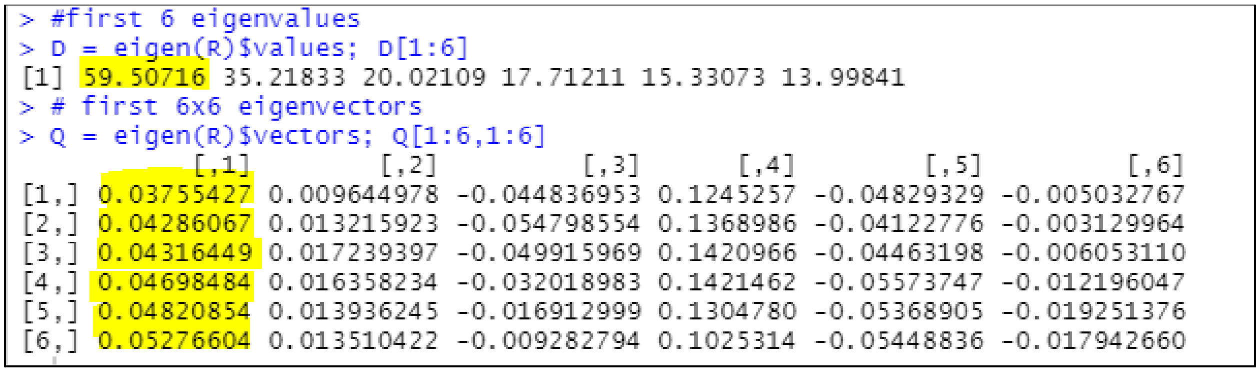
If the covariance value has a positive sign, there is a correlated relationship where the features increase or decrease together (i.e. move in the same direction). If the values have a negative sign, they are considered to show an inversely correlated relationship, where one feature increases while the other decreases. Finally, the covariance of a feature with itself, Cov(Xi,Xi) where i is any positive integer, will always be 1. This can be seen through the diagonal of the matrix. The covariance matrix is commutative, Cov(Xi,Xj) = Cov(Xj,Xi) where i ≠ j, and since it is a symmetric matrix with respect to the diagonal, we observe equal values for upper and lower triangular areas.

As we can observe for the first 6 features, they display positive signs and therefore a correlated relationship. However, we can not forget that this is just a small fraction of our total features and we do observe both positive and negative signs throughout the 400x400 correlation matrix. Now that we have quickly observed the summary of the correlations between all the possible pairs of the features we can move on to eigenvalues and eigenvectors.

STEP 3: Eigenvalues and Eigenvectors

The eigenvalues and eigenvalues are linear algebraic concepts from the correlation matrix we define earlier as R, it is needed in order to determine our PCA. Eigenvalues are displayed from highest to lowest, L1 > L2 > … L400, with a sum representing the overall variance. It is important to understand that each eigenvalue has a corresponding eigenvector respectfully displayed V1, V2,...,V400. In our data, we will continue with displaying the first 6 eigenvalues and eigenvectors while having attached the full .csv file of all 400 features with the report labeled as ***Eigenvector_Nafaryeh_Nguyen_Su.csv***

and ***Eigenvalues_Nafaryeh_Nguyen_Su.csv***.



```
> #first 6 eigenvalues
> D = eigen(R)$values; D[1:6]
[1] 59.50716 35.21833 20.02109 17.71211 15.33073 13.99841
> # first 6x6 eigenvectors
> Q = eigen(R)$vectors; Q[1:6,1:6]
           [,1]        [,2]         [,3]      [,4]        [,5]         [,6]
[1,] 0.03755427 0.009644978 -0.044836953 0.1245257 -0.04829329 -0.005032767
[2,] 0.04286067 0.013215923 -0.054798554 0.1368986 -0.04122776 -0.003129964
[3,] 0.04316449 0.017239397 -0.049915969 0.1420966 -0.04463198 -0.006053110
[4,] 0.04698484 0.016358234 -0.032018983 0.1421462 -0.05573747 -0.012196047
[5,] 0.04820854 0.013936245 -0.016912999 0.1304780 -0.05368905 -0.019251376
[6,] 0.05276604 0.013510422 -0.009282794 0.1025314 -0.05448836 -0.017942660
```

Highlighted for easy reading, we can see the largest eigenvalue and our most significant is ***L1 = 59.507*** along with its corresponding eigenvector in column 1 we define as ***V1: [0.03755427; 0.04286067; 0.04316449; 0.04698484; 0.04820854; 0.05276604]*** for the first 6 features of our data. The same goes for the rest of the 5 eigenvalues and eigenvectors in the table above. As one can see the, eigenvalues are sorted in decreasing order L1 > L2 > L3 > L4 > L5 > L6:

59.507 > 35.218 > 20.021 > 17.712 >15.330 > 13.998

We also note that the sum of the eigenvalues is equal to the trace of our correlation matrix we found using the sum() and diag() function in R:

sum(D) = trace = sum(diag(R))= 400, for the 400 features we are analysing.

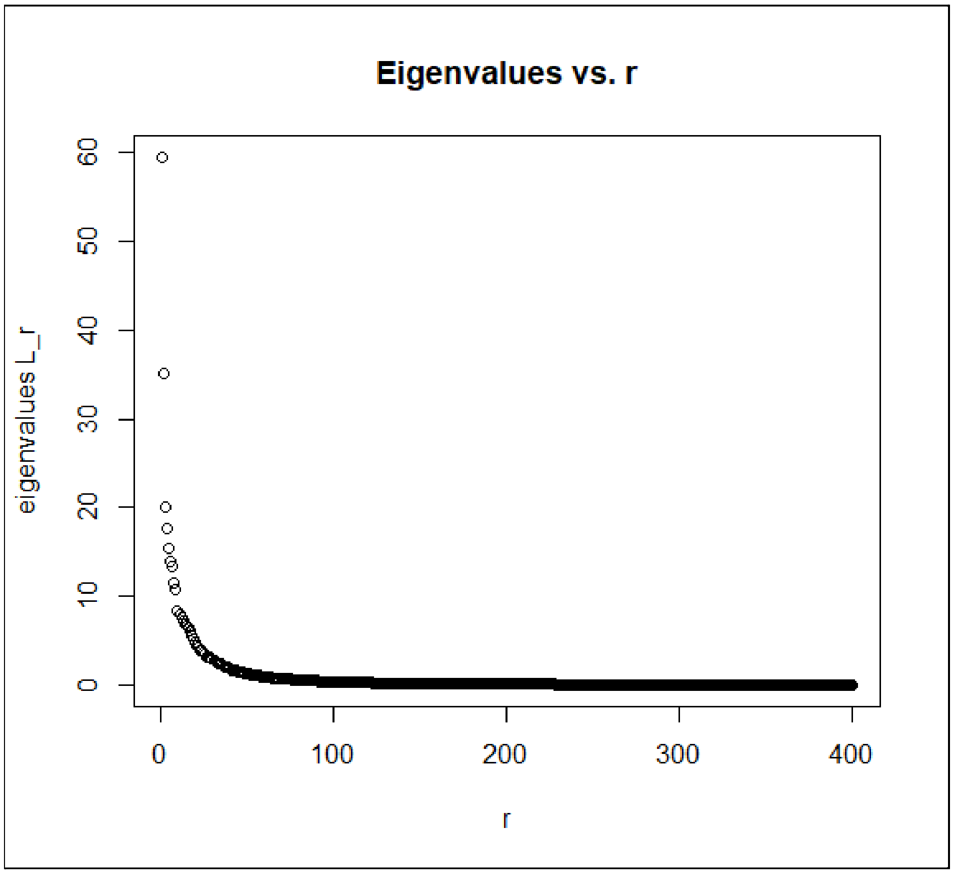


Figure 1: The following graph above is a visual plot of Eigenvalues (defined as D)versus ordinal component r (features from 1 to 400). Where our eigenvalues are in descending order.

STEP 4: Principal Component Analysis

Now we can perform principal component analysis defined as ***pca*** using the ***princomp()*** function in R to automatically plot the first 10 (and important) components. It is plotted and represented as the middle graph in Figure 2. We can also see that our ***pca*** is identical to our ***Eigenvalues vs. "r"*** graph also plotted in Figure 2 left. We can conclude that eigenvectors and eigenvalues are linear algebraic concepts that are needed to compute in order to determine/ understand the principal component of our data. The scree plot in Figure 2 orders the eigenvalues from largest to smallest. Principal components are read as new variables that are constructed as linear combinations, as we can see in the first two graphs in figure 2. These combinations are done in such a way that the principal components are uncorrelated and most of the information within the initial variables is compressed into the first components.

Therefore, the main idea of PCA is to maximize as much possible information in the first component, then maximum remaining information in the second component, and so on, until we are given a graph that is a steep decreasing curve followed by a bend and then a straight line. The components that are important are the ones in the steep curve before the graph begins to set into a horizontal trend. This is also known as the elbow method.

For instance, by eyeballing our Figure 2 right, we can conclude that a fair amount of variance is explained by the first 100 principal components and that there is an elbow after that principal component. After all, it does appear to represent a horizontal line getting close to 0%. However, this type of visual analysis is inherently unreliable mathematically. Unfortunately, there is no well-accepted objective way to decide how many principal components are enough. In fact, there is no straightforward answer to the question of how many principal components are enough, we can only estimate. This will be discussed once we look into our Percentage of Variance Explained, or PVE.

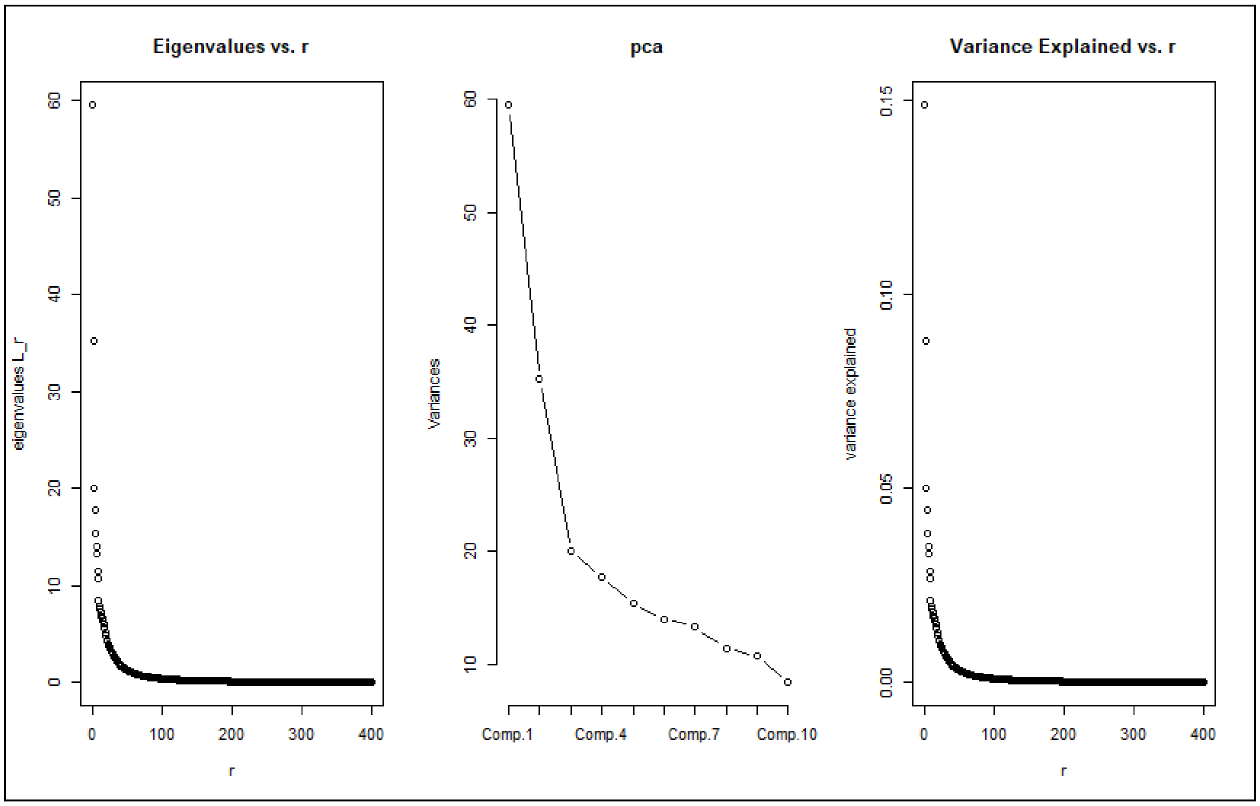


Figure 2. Left: Eigenvalues vs. Ordinal component r. Middle: Scree plot depicting PCA of each component (first 10 dimensions are shown) in DATA. Right: a scree plot depicting the Variance Explained by each of the 400 principal components in our font DATA set.

From our Variance Explained vs r plot on the right, we can see that the first principal component (PC1) explains 14.87% of the variance in the data, the second principal component (PC2) explains 8.80% of the variance of the data, a and third principal component (PC3) explains 5.00% of the variance and so forth.

Now that we know what our variance explained represents, we take a look at our Percentage of Variance Explained we calculated using the ***cumsum(D)/sum(D)*** functions in R, where D is defined as our eigenvalues. We are basically calculating the cumulative distribution of our variance explained. In Figure 3, we can see for our Percentage of Variance Explained (PVE(r)), overall 23.681% is being captured by r = 2 components and 28.686% of the variance is being explained by the first r = 3 components, and so forth. It is until we are by the first r = 100 components where we find 94.011% of the variance explained and finally 100% by all r = 400 components. Therefore, we can infer that a pretty accurate summary of our data falls around 100 dimensions, or ¼th of our data which is good news and we don't have to focus too much on the other 3/4th of the data. We will explore this part further by observing how many components are captured within 95% of the variance explained.

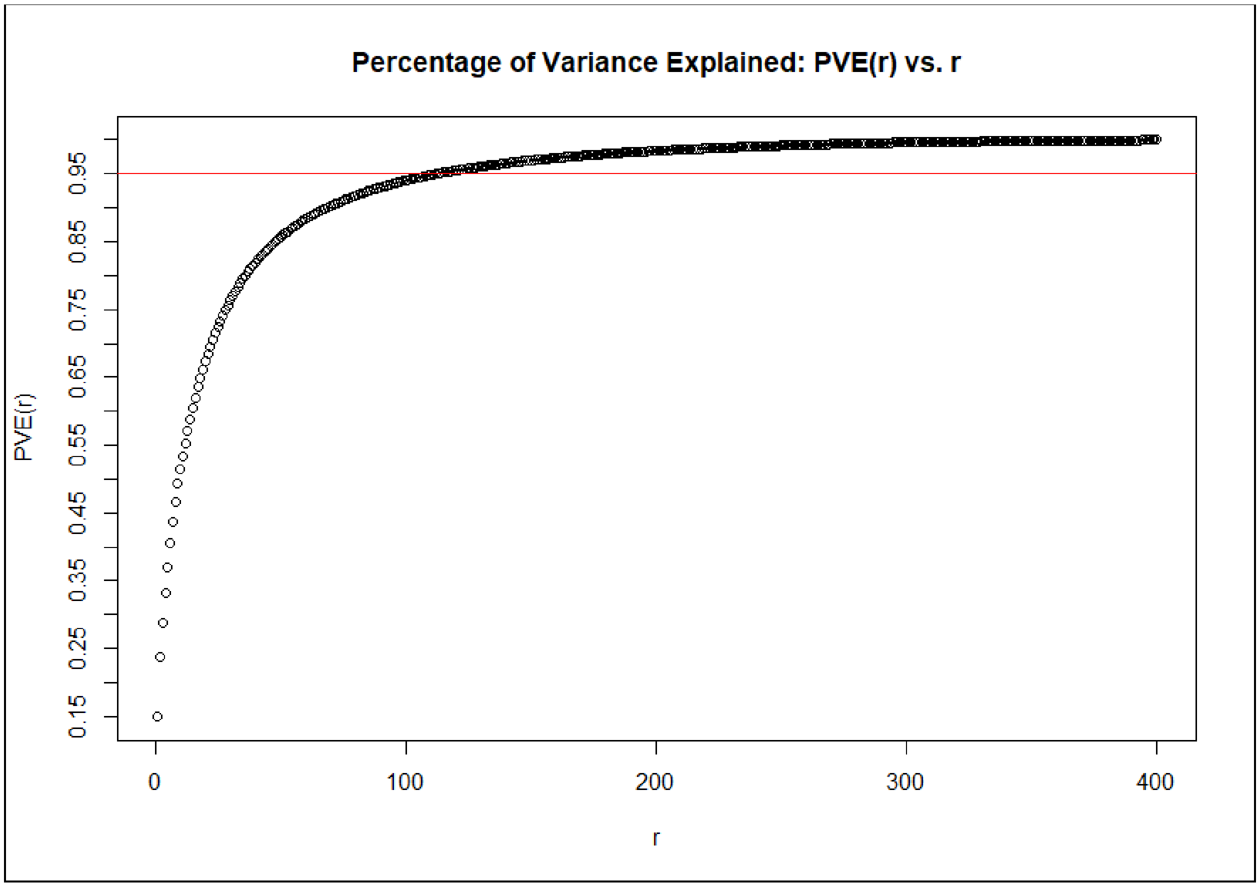
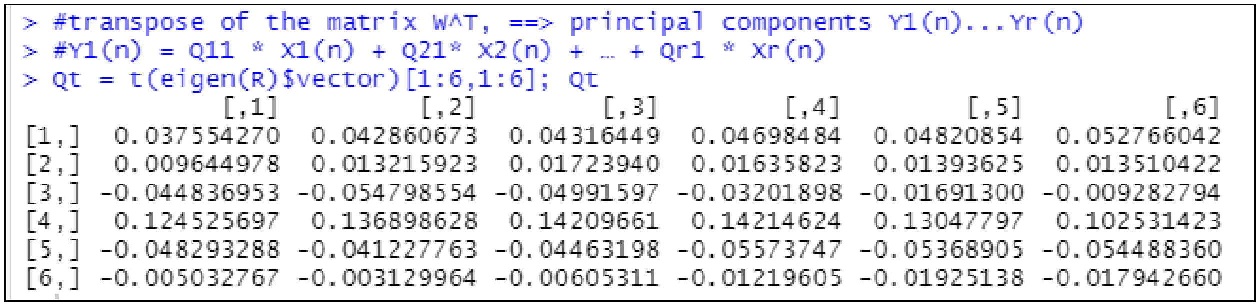


Figure 3. Percentage of Variance Explained by every 400 principal components in the font DATA set. The plot of the PVE(r) vs r. X-axes from 0 to 400 features, Y-axes from 0.0 to 1.00 (percentage)

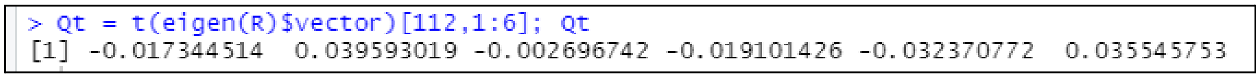
```
> pve = cumsum(D)/sum(D)
> pve[1:6]
[1] 0.1487679 0.2368137 0.2868664 0.3311467 0.3694735 0.4044696

> pve[50]
[1] 0.8567385
> pve[100]
[1] 0.9401131
> pve[200]
[1] 0.9839055
> pve[400]
[1] 1

> pve[112]
[1] 0.9497139
```

To select the optimal number of principal components we could eyeball it using Figure 3 PVE graph or we can use trial and error in R. We are looking at the PVE(r) ratio as a function of the number of components. The choice of PCs is entirely dependent on the tradeoff between dimensionality reduction and information loss. Automatic classification can often be improved if r new features generated by PCA with PVE(r) = 95%. The PVE graph we are familiar with above shows a red horizontal line to represent nearly 95% (PVE(r) = 95%) of the variance. After observing the values or r at 50,100, and 200, we narrow down our result to find our result at r = 112. At this 95% variance, we can be attributed to just 112 out of 400 features that are most important. Figure 3 provides a pretty accurate summary of the data using just 112 dimensions (or about 28%).

For the final part of this step we will compute the PCA: new features also known as principal components.
Y1(n), Y2(n),.., Yr(n) for each case in n and r = 112. We are basically taking the transpose of our 400x400 matrix Q of eigenvectors we obtained in our *STEP 3: Eigenvalues and Eigenvectors* and define it as ***Qt***. Qt is now a matrix of coefficients of new features where our columns are our 400 features and the rows represent our principal components Yi, where i = 1 to 400. However, remember that we will only be observing the principal components of Y1(n) to Y112(n) since we assigned our new r = 112 from our PVE(112) = 95%.



```
> #transpose of the matrix W^T, ==> principal components Y1(n)...Yr(n)
> #Y1(n) = Q11 * X1(n) + Q21* X2(n) + … + Qr1 * Xr(n)
> Qt = t(eigen(R)$vector)[1:6,1:6]; Qt
             [,1]         [,2]        [,3]        [,4]        [,5]         [,6]
[1,]  0.037554270  0.042860673  0.04316449  0.04698484  0.04820854  0.052766042
[2,]  0.009644978  0.013215923  0.01723940  0.01635823  0.01393625  0.013510422
[3,] -0.044836953 -0.054798554 -0.04991597 -0.03201898 -0.01691300 -0.009282794
[4,]  0.124525697  0.136898628  0.14209661  0.14214624  0.13047797  0.102531423
[5,] -0.048293288 -0.041227763 -0.04463198 -0.05573747 -0.05368905 -0.054488360
[6,] -0.005032767 -0.003129964 -0.00605311 -0.01219605 -0.01925138 -0.017942660
```



```
> Qt = t(eigen(R)$vector)[112,1:6]; Qt
[1] -0.017344514  0.039593019 -0.002696742 -0.019101426 -0.032370772  0.035545753
```

That being said:

Y1(n) = 0.0375*F1(n) + 0.0428*F2(n) + 0.0431*F3(n) +0.0469*F4(n) + 0.0482*F5(n) + 0.0527*F6(n)
Y2(n) = 0.0096*F1(n) + 0.0132*F2(n) + 0.0172*F3(n) + 0.0163*F4(n) + 0.0139*F5(n) + 0.01351*F6(n)
…
Y112(n) = -0.0050*F1(n) + 0.0395*F2(n) - 0.0026*F3(n) - 0.0191*F4(n) -0.0323*F5(n) - 0.0355*F6(n)

Our vector Yi(n) where i = 1 to 400 of our 400 new features is our new column vector Y1(n) …Y400(n). Our vector X(n) of our original 400 features = column vector X1(n) …X400(n). These give us a matrix formula: Y(n) = ***Qt*** * X(n). However, as stated earlier, we will only keep the first r =112 coordinates of Y(n) to get the reduced vector of r =112 new features.

In conclusion, the goal of PCA was to find a low-dimensional representation of the observation that explains a good fraction of the variance and we believe we found a reasonable value of new features r = 112 out of our 400. Compression helps eliminate unnecessary, redundant, or noisy information. Using linear compression through PCA is helpful when we have large numerical features. However, one must realize this is not always the case and there are other methods and functions that can produce better results.

Step 5: Apply KNN using only the “r” as new features

Before applying the KNN algorithm, It’s needed to find out the optimal value of k as the number of nearest neighbors based on feature similarity in order to get better performance and accuracy. But, finding the optimal value of k is one of the hardest steps in the KNN algorithm, because there is no straightforward and structured method at all to calculate it.

A small value of k has a lower value of bias and a higher variance which is associated generally with a more complex model, and a large value of k has a higher value of bias and a lower variance which is associated with a simpler model. Also, choosing a large value of k is not effective and efficient, because it computationally costs a lot. Although a larger value of k will get higher accuracy, the model will also go under-fitted which means the error rate will again increase. Therefore, the optimal k should be chosen as a small value, for the reason that noise will make a higher influence on the result.

Based on the above idea, and observations taking throughout STEP 3 and 4 on Eigenvalues, eigenvectors and PCA we can finally use KNN to find our best k. Using the “r” = 112 new features Y1,Y2,...Y112 we calculated in STEP 4, we get a matrix formula: Y(n) = ***Qt*** * X(n), where Qt is our eigenvector matrix from eigenvalues 1 to 112 (in descending order) and X(n) is standardized font DATA. Just as in Homework2 we separate the dataset by classes CL1, CL2, CL3 for our 3 fonts: Comic, Bookman, Monotype respectfully. After this we observe that our classes still hold the following number of cases in each: n1 = 597, n2 = 667, n3 = 667. Next we make sure to test 20% of each class and train the other 80%. Following the steps taken in Homework 2 we define our ***TRAINSET_TARGET, TESTSET_TARGE, TESTSET, and TRAINSET*** to obtain the following "K versus accuracy" plot. The best range for k can be identified, and the optimal k should be between 1 and 50. It looks like the first ten of k are potential candidates, because too low accuracy is not expected. So how to decide what the optimal k will be next? The answer is the Elbow Method that performs the k-means clustering on the data set for a range of k.

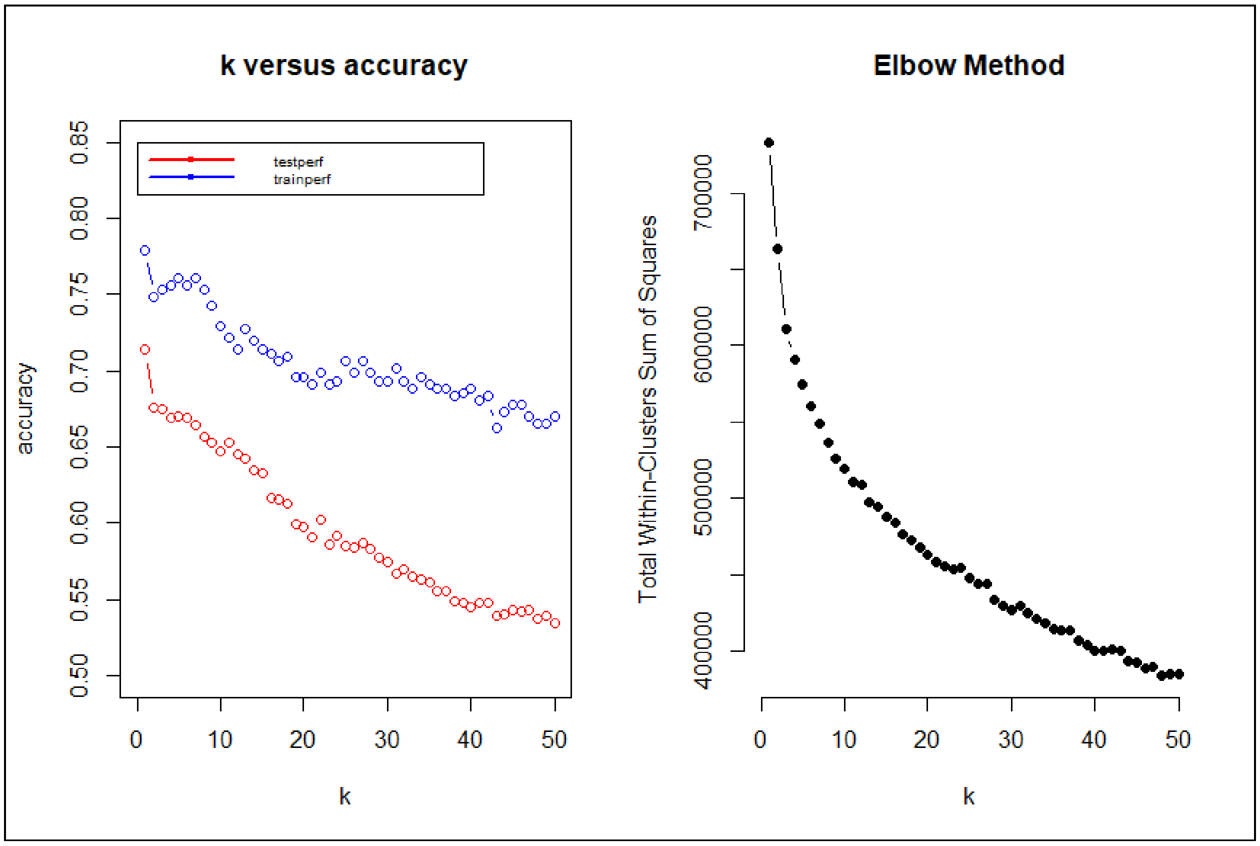
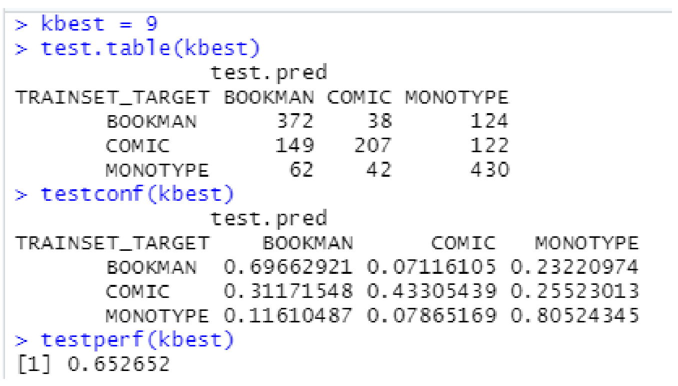
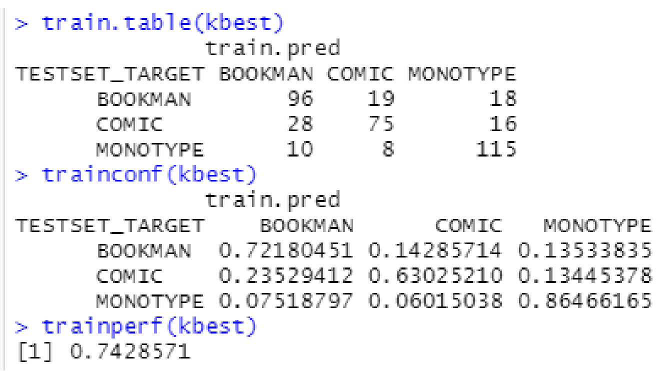


Figure 4. Left: K vs accuracy plot of testperf in red and trainpref in blue. Right: using the elbow method to identify our optimal value of k.

The Elbow Method consists of plotting the number of clusters k against the total within-cluster sum of squares. The total within-cluster sum of squares is a measure of the variability of the observations within each cluster k. Observe that, when k = 9, the ratio of between_SS/total_SS starts to change slowly and remains less changing compared to other k's in the graph. The cluster k = 9 can also consider the elbow point of the curve. Therefore, the optimal k ***kbest = 9*** should be a good option.



```
> kbest = 9
> test.table(kbest)
                 test.pred
TRAINSET_TARGET BOOKMAN COMIC MONOTYPE
       BOOKMAN      372    38      124
       COMIC        149   207      122
       MONOTYPE      62    42      430
> testconf(kbest)
                 test.pred
TRAINSET_TARGET    BOOKMAN      COMIC   MONOTYPE
       BOOKMAN  0.69662921 0.07116105 0.23220974
       COMIC    0.31171548 0.43305439 0.25523013
       MONOTYPE 0.11610487 0.07865169 0.80524345
> testperf(kbest)
[1] 0.652652
```



```
> train.table(kbest)
                 train.pred
TESTSET_TARGET BOOKMAN COMIC MONOTYPE
      BOOKMAN       96    19       18
      COMIC         28    75       16
      MONOTYPE      10     8      115
> trainconf(kbest)
                 train.pred
TESTSET_TARGET    BOOKMAN      COMIC   MONOTYPE
      BOOKMAN  0.72180451 0.14285714 0.13533835
      COMIC    0.23529412 0.63025210 0.13445378
      MONOTYPE 0.07518797 0.06015038 0.86466165
> trainperf(kbest)
[1] 0.7428571
```

In Homework 2, we stated that our best k was k = 1 with a testpref percent accuracy of 71.34%, and trainpref percent accuracy of 77.92%. Comparing our results to that in Homework 2 we can clearly see that at k = 9 is the better "best k" with testpref at 65.26% and trainpref at 74.28% . Even though at k = 1, both our test and trianpref both give us higher percentage results, when selecting k to be 1, we risk our data to be impacted by noise or irrelevant data. By first using PCA we were able to reduce the number of variables of a data set, while preserving as much important information as possible. Therefore, we are confident in our results of k = 9 for this report. However, we can not say that this is the best and only way to analyse the data, there are many methods and functions that one can use that will give us different results, it is important to understand how to use and interpret them.

Final part of this report we will compute 6 color graphic scatter plot displays of our 3 classes in the 6 planes using only the first 4 principal components. As you can see in Figure 5 below, Y1, Y2, Y3, and Y4 are the first principal components of our font DATA, along with their 6 planes. Note that blue =Comic, red =Bookman, and green =Monotype.

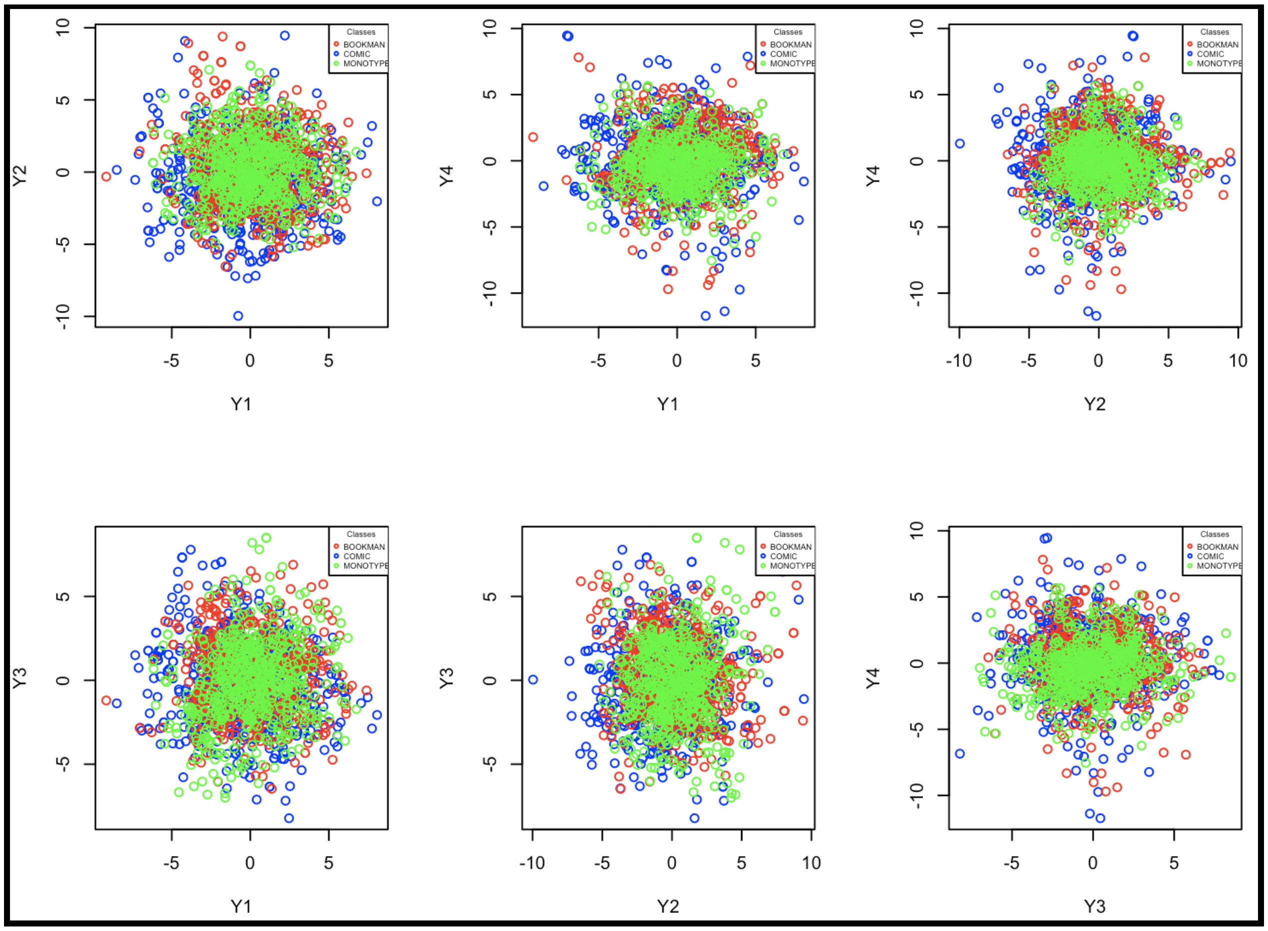


Figure 5. PCA plots of our three classes: blue=comic, red=bookman, and green=monotype. TopLeft: PC plot of Y1 vs. Y2, TopMiddle: Y1 vs. Y4, TopRight:Y2 vs.Y4, BottomLeft: Y1 vs. Y3, BottomMiddle: Y2 vs. Y3, BottomRight: Y3 vs. Y4

Figure 5 is also known as a score plot of our PCA; it indicates the projection of the data onto the span of 2 principal components at a time. The score plot for all 4 principal components appears to be clustered within the zero marker with a few points scattered all around it. Since this appears to be the case for all 6 planes, we can suggest our data follows a normal distribution. Out of these 6 planes we observe PCs of Y1 and Y4 to have tighter clusters than the rest and fewer points scattered all around it.

Once again, we can not say that this is the best and only way to analyse a dataset, there are many methods and functions that one can use that help give different results for different types of datasets. Therefore, it is important to understand how to use and interpret them.

R Codes

```
rm(list=ls()) # to remove all objects from a specified environment
cat("\f") # to clean console
```

# STEP 0: Data Setup

```
library(readr)
setwd("~/Library/Mobile Documents/com~apple~CloudDocs/Study Files (Graduate)/MATH
6350/Homework/HW03/fonts/")
# We selected COMIC.csv, BOOKMAN.csv, MONOTYPE.csv.
COMIC <- data.frame(read_csv("COMIC.csv"))
BOOKMAN <- data.frame(read_csv("BOOKMAN.csv"))
MONOTYPE <- data.frame(read_csv("MONOTYPE.csv"))

c.names = names(COMIC) # All three font files shares same categories of columns

c.discard = match(c("fontVariant", "m_label", "orientation", "m_top", "m_left", "originalH",
                "originalW", "h", "w"),c.names) # discard 9 columns listed
# na.omit() function is to omit all rows that contain NA values
comic = na.omit(COMIC[,-c.discard])
bookman = na.omit(BOOKMAN[,-c.discard])
monotype = na.omit(MONOTYPE[,-c.discard])

CL1 = comic[comic[,match("strength", names(comic))] == 0.4 &
          comic[,match("italic", names(comic))] == 0,]
CL2 = bookman[bookman[,match("strength", names(bookman))] == 0.4 &
            bookman[,match("italic", names(bookman))] == 0,]
CL3 = monotype[bookman[,match("strength", names(monotype))] == 0.4 &
             bookman[,match("italic", names(monotype))] == 0,]
DATA = rbind(CL1, CL2, CL3)
str(DATA)
```

# STEP 1: Standardization

```
true_set = DATA[,1]
start_col = match("r0c0",names(DATA))
end_col = ncol(DATA)
X = DATA[,start_col: end_col]
m = sapply(X, mean) # same as apply(X, MARGIN = 2, mean)
s = sapply(X, sd)
SDATA = scale(X)
```

# STEP 2: Correlation Matrix Computation

```
R = cor(X)
# sigma = cov(X) # variance-covariance matrix "sigma"
# R=cov2cor(sigma) # either way to find correlation matrix
# all.equal(cor(X), cov2cor(sigma))
```

# STEP 3: Eigenvalues and Eigenvectors

```
D = eigen(R)$values
# or D = (summary(pca)$sdev)^2
Q = eigen(R)$vectors
```

```r
#first 6 eigenvalues
D = eigen(R)$values; D[1:6]
# first 6x6 eigenvectors
Q = eigen(R)$vectors; Q[1:6,1:6]


# STEP 4: Principal Component Analysis
layout(matrix(c(1:3), 1, 3))

# plot of the eigenvalues vs ordinal component r
plot(1:400, D, ylab = "eigenvalues L_r", xlab = "r", main = "Eigenvalues vs. r")
#pca = princomp(X,cor=TRUE,scores=TRUE)
#summary(pca)
#plot(pca, type = "line")

#The variance explained by each principal component is obtained by squaring and then
#plot of the variance explained vs r
plot(1:400, D[1:400]/sum(D), xlab = "r", ylab = "variance explained", main = "Variance Explained
vs. r")

# compute the proportion of variance explained by each principal component and then
# plot of the PVE(r) vs r
plot(1:400, cumsum(D)/sum(D), xlab = "r", ylab = "PVE(r)", main = "Percentage of Variance
Explained: PVE(r) vs. r", yaxt = "n")
axis(2, at = seq(0, 1, 0.05))
abline(a = 0.95, b = 0, col = "red")

#transpose of the matrix W^T, ==> principal components Y1(n)...Yp(n)
#Y1(n) = W11 * X1(n) + W21* X2(n) + … + Wp1 * Xp(n)
Qt = t(eigen(R)$vector)[1:6,1:6];Qt

# STEP 5 apply KNN using only the "r" (112) new features
r = order(abs(cumsum(D)/sum(D) - 0.95), decreasing = FALSE)[1]
V = Q[, 1:r]
X = as.matrix(scale(X))
Y = data.frame(true_set, X%*%V)
CL1 = Y[Y[,match("true_set", names(Y))] == "COMIC",]
CL2 = Y[Y[,match("true_set", names(Y))] == "BOOKMAN",]
CL3 = Y[Y[,match("true_set", names(Y))] == "MONOTYPE",]

n1 = nrow(CL1)
n2 = nrow(CL2)
n3 = nrow(CL3)
N = sum(n1, n2, n3)
N == nrow(DATA) # verification

SX = Y[,-1]

r1 = round(n1 * 0.2) # a set testCL1 of r1 test cases
s1 = n1 - r1 # a set trainCL1 of s1 training cases
```

```
r2 = round(n2 * 0.2)
s2 = n2 - r2
r3 = round(n3 * 0.2)
s3 = n3 - r3

trainCL1 = CL1[1:s1,]
trainCL2 = CL2[1:s2,]
trainCL3 = CL3[1:s3,]
testCL1 = CL1[(s1+1):n1,]
testCL2 = CL2[(s2+1):n2,]
testCL3 = CL3[(s3+1):n3,]

TRAINSET_TARGET = rbind(trainCL1, trainCL2, trainCL3)[,1] # true train set
TESTSET_TARGET = rbind(testCL1, testCL2, testCL3)[,1] # true test set

TRAINSET = rbind(SX[1:s1,], SX[(n1+1):(n1+s2),], SX[(n1+n2+1):(n1+n2+s3),])
TESTSET = rbind(SX[(s1+1):n1,], SX[(n1+s2+1):(n1+n2),], SX[(n1+n2+s3+1):(n1+n2+n3),])

library(class)
test.pred = function(k) {
  set.seed(1)
  test.pred = knn(train = TESTSET, test = TRAINSET, cl = TESTSET_TARGET, k = k)
  return(test.pred)
}
test.table = function(k) {
  test.pred = test.pred(k)
  test.table = table(TRAINSET_TARGET, test.pred)
  return(test.table)
}
testconf = function(k) {
  test.table = test.table(k)
  testconf = prop.table(test.table, margin = 1)
  return(testconf)
}
testperf = function(k) {
  test.table = test.table(k)
  testperf = sum(diag(test.table))/sum(test.table)
  return(testperf)
}
train.pred = function(k) {
  set.seed(1)
  train.pred = knn(train = TRAINSET, test = TESTSET, cl = TRAINSET_TARGET, k = k)
  return(train.pred)
}
train.table = function(k) {
  train.pred = train.pred(k)
  train.table = table(TESTSET_TARGET, train.pred)
  return(train.table)
}
trainconf = function(k) {
  train.table = train.table(k)
  trainconf = prop.table(train.table, margin  = 1)
  return(trainconf)
}
trainperf = function(k) {
  train.table = train.table(k)
  trainperf = sum(diag(train.table))/sum(train.table)
  return(trainperf)
}

layout(matrix(c(1:2), 1, 2))
k = seq(1,50,1)
testperf_k = mapply(testperf, k)
trainperf_k = mapply(trainperf, k)
plot(k, testperf_k, type = "b", col = "red", xlab = "k",
     ylab = "accuracy", main = "k versus accuracy",
     xlim = c(0, 50),ylim = c(0.5, 0.85))
lines(k, trainperf_k, type = "b", col = "blue")
legend(0, 0.85, c("testperf", "trainperf"), lwd = 2,
       col = c("red", "blue"), pch = 1, cex = 0.6)

set.seed(1)
plot(k, sapply(k, function(k) {kmeans(SX, k, iter.max = 50)$tot.withinss}),
     type = "b", pch = 19, col = "black", frame = FALSE,
     xlim = c(0, 50),
     xlab = "k", ylab = "Total Within-Clusters Sum of Squares",
     main = "Elbow Method")

kbest = 9
test.table(kbest)
testconf(kbest)
testperf(kbest)
train.table(kbest)
trainconf(kbest)
trainperf(kbest)

layout(matrix(c(1:6), 2, 3))
Y = Y[,-1]
# plot(Y[,1], Y[,2], xlab = "Y1", ylab = "Y2")
# plot(Y[,1], Y[,3], xlab = "Y1", ylab = "Y3")
# plot(Y[,1], Y[,4], xlab = "Y1", ylab = "Y4")
# plot(Y[,2], Y[,3], xlab = "Y2", ylab = "Y3")
# plot(Y[,2], Y[,4], xlab = "Y2", ylab = "Y4")
# plot(Y[,3], Y[,4], xlab = "Y3", ylab = "Y4")

layout(matrix(c(1:6), 2, 3))
P1 <- data.frame(x=Y[,1], y=Y[,2], z=true_set)
attach(P1); plot(x, y, xlab = "Y1", ylab = "Y2", col=c("red","blue","green")[z]); detach(P1)
legend("topright", inset=c(-0.02,0), legend = levels(P1$z), col=c("red","blue","green"), pch=1,
title="Classes",cex = 0.45)

P2 <- data.frame(x=Y[,1], y=Y[,3], z=true_set)
```

```
attach(P2); plot(x, y, xlab = "Y1", ylab = "Y3", col=c("red","blue","green")[z]); detach(P2)
legend("topright", inset=c(-0.02,0), legend = levels(P2$z), col=c("red","blue","green"), pch=1,
title="Classes",cex = 0.45)

P3 <- data.frame(x=Y[,1], y=Y[,4], z=true_set)
attach(P3); plot(x, y, xlab = "Y1", ylab = "Y4", col=c("red","blue","green")[z]); detach(P3)
legend("topright", inset=c(-0.02,0), legend = levels(P3$z), col=c("red","blue","green"), pch=1,
title="Classes",cex = 0.45)

P4 <- data.frame(x=Y[,2], y=Y[,3], z=true_set)
attach(P4); plot(x, y, xlab = "Y2", ylab = "Y3", col=c("red","blue","green")[z]); detach(P4)
legend("topright", inset=c(-0.02,0), legend = levels(P4$z), col=c("red","blue","green"), pch=1,
title="Classes",cex = 0.45)

P5 <- data.frame(x=Y[,2], y=Y[,4], z=true_set)
attach(P5); plot(x, y, xlab = "Y2", ylab = "Y4", col=c("red","blue","green")[z]); detach(P5)
legend("topright", inset=c(-0.02,0), legend = levels(P5$z), col=c("red","blue","green"), pch=1,
title="Classes",cex = 0.45)

P6 <- data.frame(x=Y[,3], y=Y[,4], z=true_set)
attach(P6); plot(x, y, xlab = "Y3", ylab = "Y4", col=c("red","blue","green")[z]); detach(P6)
legend("topright", inset=c(-0.02,0), legend = levels(P6$z), col=c("red","blue","green"), pch=1,
title="Classes",cex = 0.45)
```