MATH 6350 Fall 2020 MSDS
Homework 4: K-means Clustering & Tree Classification

Sara Nafaryeh
Tony Nguyen
Thomas Su

All authors played an equal part

Introduction

In this report, we will continue to add to what we have performed in Homework 2 and Homework 3 by looking into improving Principal Component Analysis, PCA by applying the Kmeans clustering method. A brief reminder of what was done in Homework 2 and 3: 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. Next we used PCA, 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.

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 4, we will be more focused on using the K-means clustering on the 400-pixel features into k disjoint clusters. K-means clustering is an unsupervised clustering algorithm that partitions a data set into K distinct, non-overlapping clusters that are similar to one another. The main idea of the K-means

clustering is to minimize the distance within a cluster all while maximizing the distance between different clusters.

STEP 0: DATA Set up

A quick reminder of our data set up from 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

Once again, we make sure to standardize the 400 features. 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 K-means clustering 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: Principal Component Analysis

In this section, we will quickly go over our PCA analysis from Homework 3. 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 as seen in Figure 1 left. 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 1 left, 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%. From our Variance Explained vs r plot on the left, 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.

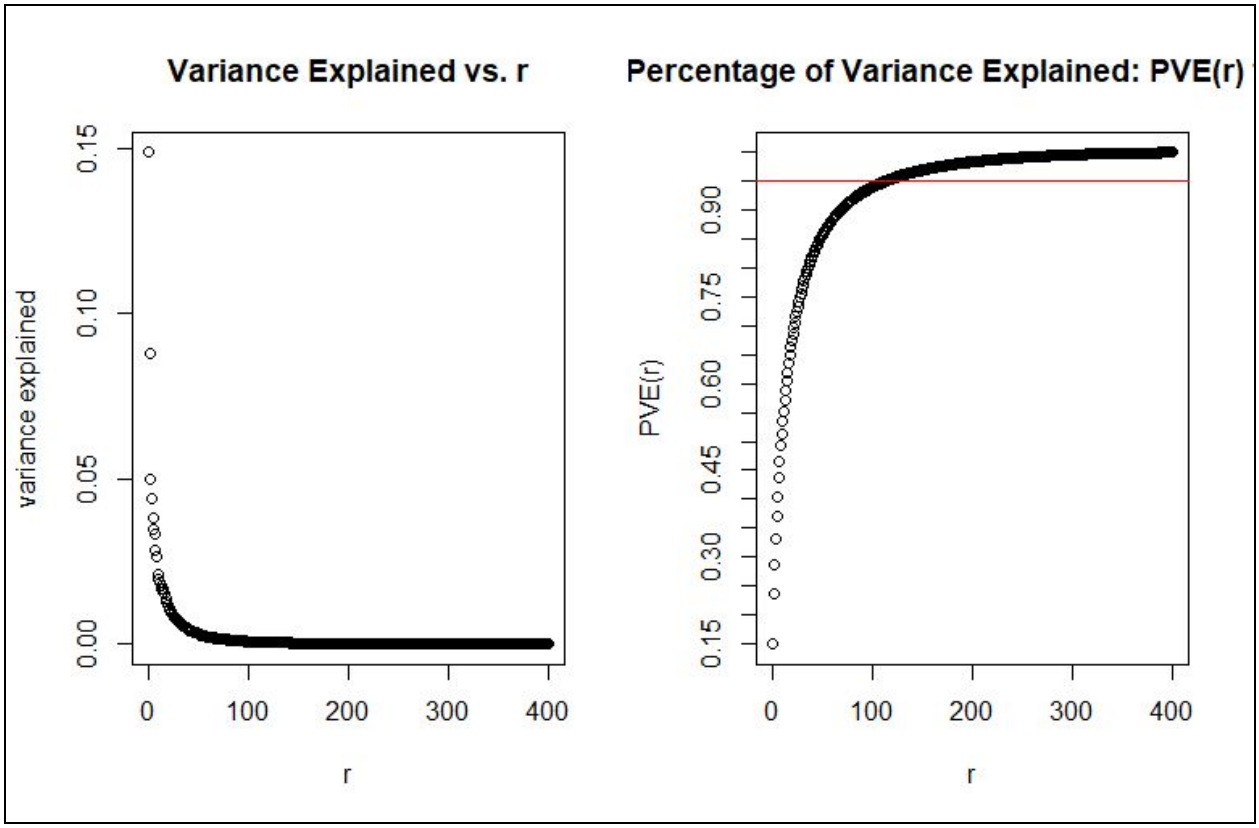


Figure 1. Left: A scree plot depicting the Variance Explained by each of the 400 principal components in our font DATA set. Right: Percentage of Variance Explained by every 400 principal components in the font DATA set. The plot of the PVE(r) vs r. X-axes from 0 to 400 features, Y-axes from 0.0 to 1.00 (percentage)

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 1 Right, we can see for our Percentage of Variance Explained (PVE(r)), overall 23.681% is being captured by r = 2 components and so on. The red horizontal line represents nearly 95% (PVE(r) = 95%) of the variance, we identified to be r = 112 dimensions (or about 28%). Therefore, we concluded that 95% of our data falls within 1 to 112 dimensions and computed the PCA: new features also known as principal components.

Y1(n), Y2(n),.., Yr(n) for each case in n and r = 112.

```
> #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
```

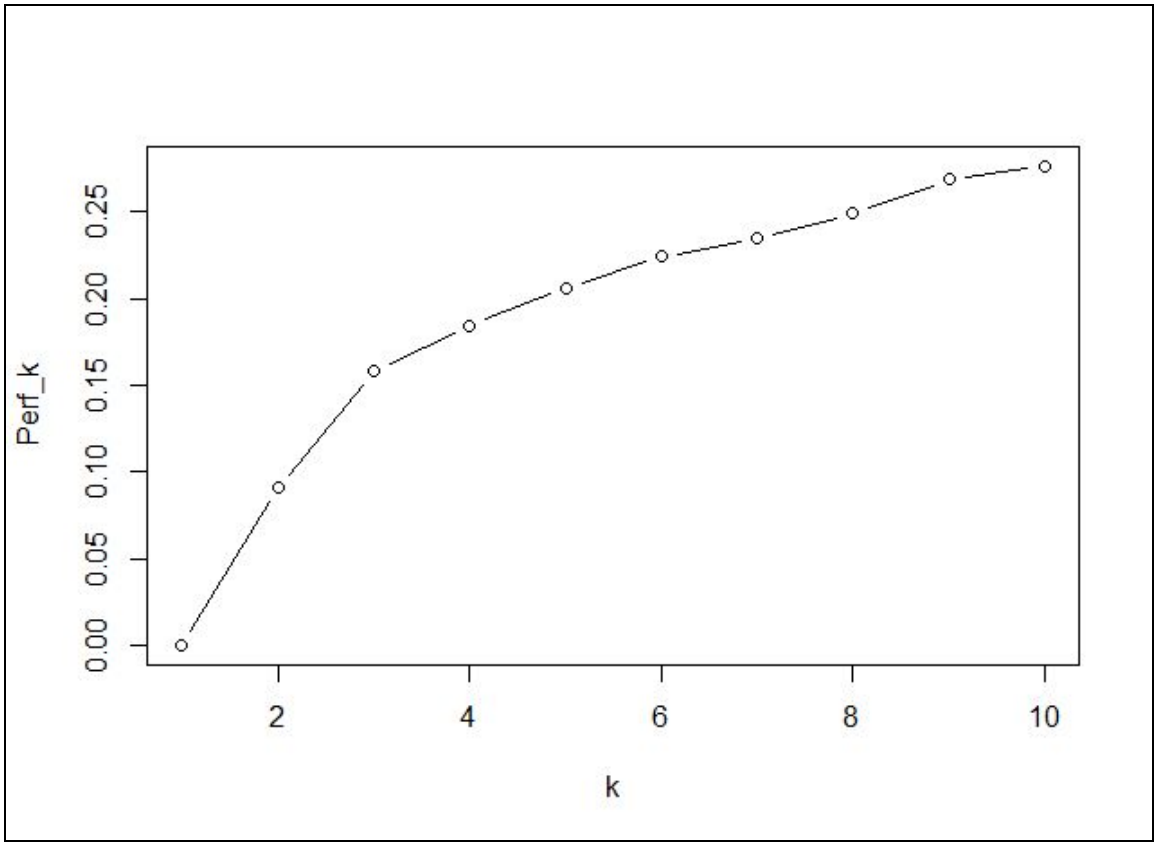
That being said:

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)

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 in Homework 3 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.

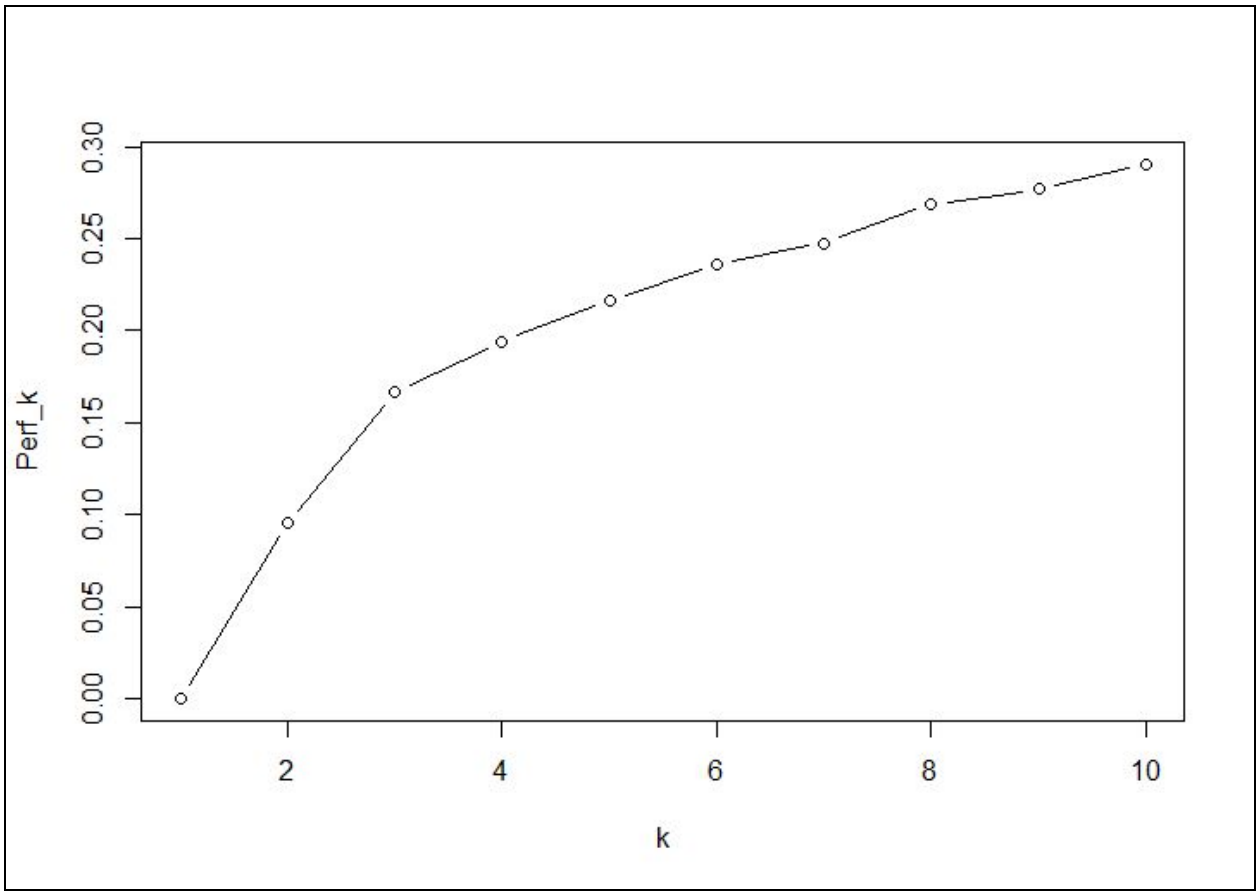
(Question 1: K-means Clustering)

The K-means clustering method is the simplest and most commonly used clustering method for splitting a dataset into a set of k groups. When clustering observations, we want similar observations in the same group, while different observations in dissimilar groups. There is no response variable therefore this method is known as unsupervised and seeks to find relationships between the total number of observations. Using our standardized data features defined as ***SDATA***, we apply k-means clustering: ***kmeans(SDATA, k, nstart = 50),*** to partition the 1931 cases into k disjoint clusters, where k = 1 to 10. And "repeat" the k-means function 50 times. For each k, we compute the Reduction of Variance, Perf(k), and plot the following curve:



Perf_k vs k curve using our original standardized data of 400 features defined as SDATA

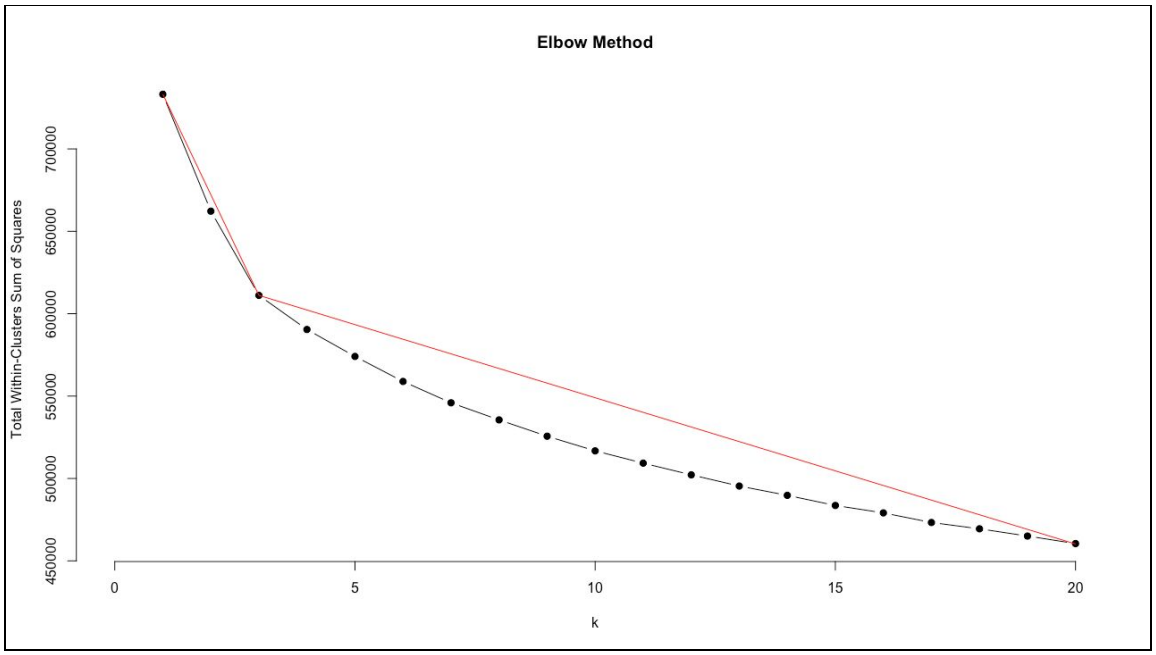
We also explore our reduced model, features r = 112 out of our 400, we obtained using PCA in Homework 3. Defined as ***SX***, we apply k-means clustering (as we had done above with SDATA): ***kmeans(SX, k, nstart = 50),*** to partition the 1931 cases into k disjoint clusters, where k = 1 to 10. Once again we “repeat” the k-means function 50 times. For each k, we compute the Reduction of Variance, Perf(k), and plot the following curve:



Perf_k vs k curve using our original standardized data of 400 features defined as SX

We observe that between the two curves plotted, there doesn't appear to be much of a difference and is quite interesting. Running our PCA reduced new features, we notice it runs faster and hopes it can even give better results than using the full model SDATA.

Moving forward, we apply the “elbow method” explained in Homework 3. This method helps to visually show us which is the best k value. It is a graph that has a steep decreasing curve followed by a bend and continues to a straight horizontal line. We are interested in where the bend is. We find our “best” value k* for the number of clusters k to be 3 as shown in the graph below:

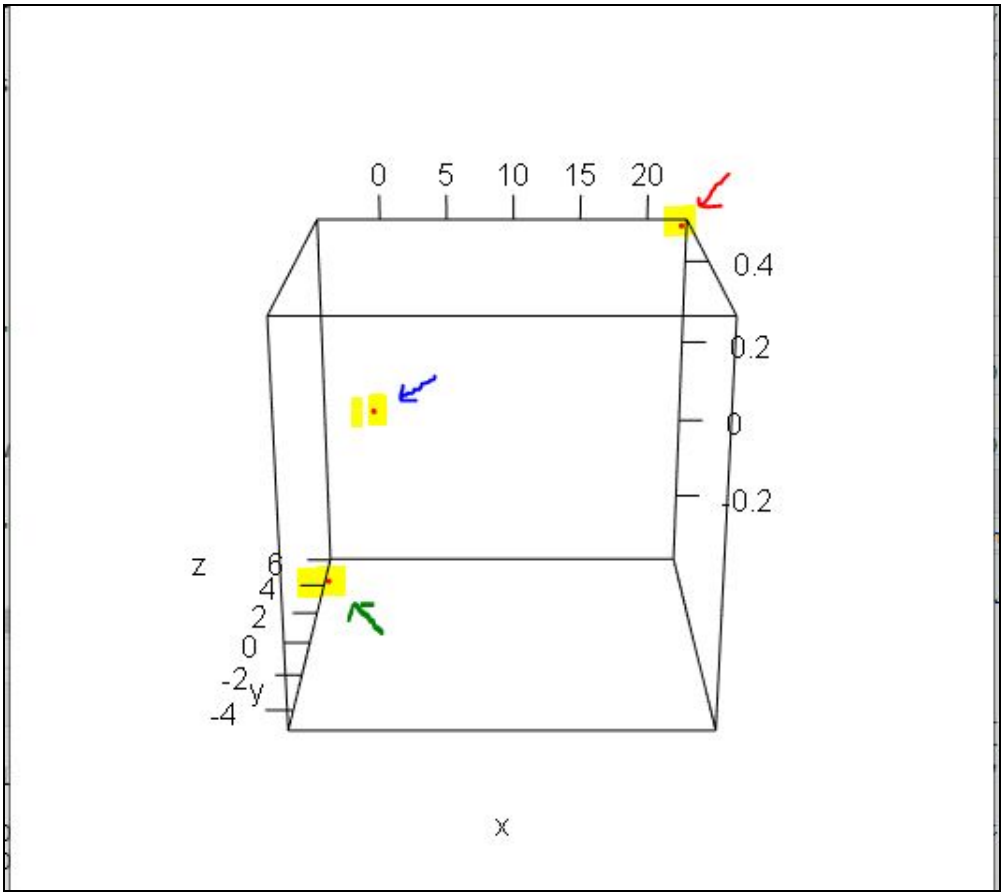


(Question 2: K-means Clustering)

After selecting the “best” value k* at 3 clusters, we can get started on plotting our 3D graph of the clustering. First, we compute a k x N matrix listing the N=1931 coordinates of cluster centers CENT1, …, CENT3 defined as **SX_FULL_kbest**. This is the first 3 vectors of **SX_FULL = X x Q** matrix where X is our 400 features and Q is our 400 eigenvectors from homework3. Using the PCA results we compute the 3D vectors, gathering the first 3 principal components of CENT1 ...CENT3, we get the 3x3 matrix listing the 3 vectors:

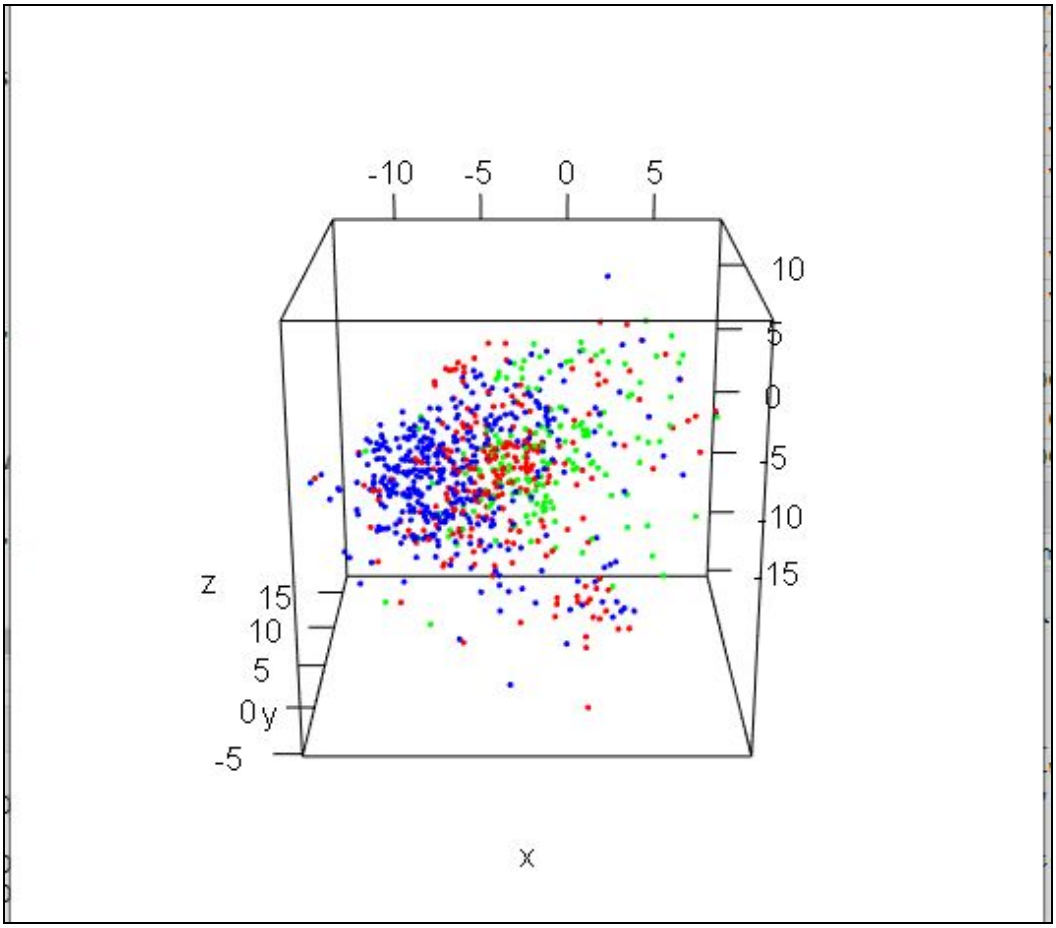
	[,1]	[,2]	[,3]
1	1.407422	-4.924437	0.3098136
2	-4.273792	3.950941	-0.3600967
3	22.552347	5.903588	0.4882885

According to our 3x3 matrix, the columns represent the x,y,z coordinates, while the rows represent the 3 clusters. For visual ease, the 3 vectors c1, c2, c3 coordinates are emphasized by the respected color arrows in the graph below: 1 = Bule, 2 = Green, and 3 = Red.



Defining bigCLU as the cluster with the largest number of cases, t = maximum size of the kmeans data. Now we can compute the 3-dimensional vectors v1,v2,v3 gathering the first 3 principal components of all the cases belonging to the CLU(t). The following 3D graph below displays the t = 925 the largest size of the 3 cluster vectors in 3 colors of our 3 classes:

Red, Blue, Lime


-10
-5
0
5
10
5
0
5
10
15
z
15
10
5
0 y
-5
x

(Question 3) - **Gini Index**

Using our clusters k* = 3, and our 3 classes of the fonts: Class1=BOOKMAN, Class2: COMIC, and Class3: MONOTYPE; we will now explore the Gini indexes for each of our 3 clusters. Also known as the Gini impurity, the Gini index measures total variance across the K classes, varying between values of 0 and 1. For this reason, the Gini index is referred to as a measure of node purity. A small value indicates that a node contains predominantly observations from a single class, therefore it is considered pure.

The Gini index is defined as the following, where $\hat{p}_{mk}$ represents the proportion of training observation in the *mth* region that is from the *kth* class.

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

And in our R-code it is defined as ***gin(CLU_j)*** for each of the 3 clusters:

```
gin(CLU_1) = 0.248305
gin(CLU_2) = 0.2495601
gin(CLU_3) = 0.05828209
```

We can observe that class 3 has a small Gini index value, close to 0, therefore considered to have high purity. Now we consider the Gini impurity. A measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Our Gini impurity of the clusters CLU1, CLU2, and CLU3 is defined, ***IMP(k*)*** = **0.55614**. It comes out to be the sum of our 3 Gini index clusters we calculated above.

Moving on, we calculate the matrix of frequencies ***FREQ_j*** = A(m,j) /Sj, where A(m,j) is the size of the three font classes intersections of CLUj coordinate center and Sj is the size of the CLUj. The matrix frequency shows the proportions of the fonts for each of our 3 clusters.

```
> FREQ_j = matrix(FREQ_j, nrow = 3)
> print(FREQ_j) # frequency: dimension of number of classes * kbest
          [,1]      [,2]      [,3]
[1,] 0.3972912 0.2940541 0.3583333
[2,] 0.4097065 0.2108108 0.3250000
[3,] 0.1930023 0.4951351 0.3166667
```

We can move onto computing the class index TOP(j) for each of ou 3 clusters such that A(TOP(j),j) = max(A(1,j),A(2,j), A(3,j) where j = 1,2,3

The maximum of the three frequencies of the three font classes are as follow:

```
> A_mj = matrix(A_mj, nrow = 3, byrow = TRUE)
> print(A_mj)
     [,1] [,2] [,3]
[1,]  352  272   43
[2,]  363  195   39
[3,]  171  458   38
```

This represents which class from most to least frequent in the cluster.

In our case, A_mj column 1 represents our most frequent class cluster to be COMIC, followed by MONOTYPE, and finally last being BOOKMAN. We can confirm this using the Confusion matrix in the next question. We can also note that in the case of A_mj, row 1 represents CLASS2: BOOKMAN, row 2 represents CLASS1: COMIC, and row 3 represents our CLASS3: Monotype of our true_set.

We can confirm this from **STEP 0: DATA Set up** The respected row sizes for CLASS1, CLASS2, and CLASS3 are named ***n1, n2, n3*** = 597, 667, 667

(Question 4: Confusion Matrix)

In this section, we use the result of the k-means to help us identify each case(n) to be assigned to a specific cluster CLUj(n). Looking at the coordinate number n in the vector of our k-means function. We define predictor, Pred(n) for class of case(n): Pred(n) = Top(j(n)). Note that the predictor for each case is not true all the time because we already know the true class of the function. Computing the 3x3 confusion matrix, we are comparing the performance of actual assignments vs predictions. Our CONF matrix of the predictor Pred(n):

```
> CONF = prop.table(table_matrix, margin  = 1)
> print(CONF)
           predict
true_set       BOOKMAN      COMIC   MONOTYPE
  BOOKMAN   0.06446777 0.52773613 0.40779610
  COMIC     0.06532663 0.60804020 0.32663317
  MONOTYPE  0.05697151 0.25637181 0.68665667
```

Once again, we observe the larger values we care about along the diagonal of the matrix. Prediction accuracy for the true set of three classes is between 60-69% which is acceptable. Comparing the CONF matrix to our FREQ, frequency matrices A_mj matrix we can see that we get the same matrix since we have k* = 3. Looking at our A_mg matrix, if we divide each value by its row sum we will be left with our CONF matrix. For example for the max frequency of class COMIC:

352/(352+272+43) = 0.5277 the predicted value of COMIC and true_set of BOOKMAN.

363/(363+195+39) = 0.6080 the predicted value of COMIC and true_set COMIC.

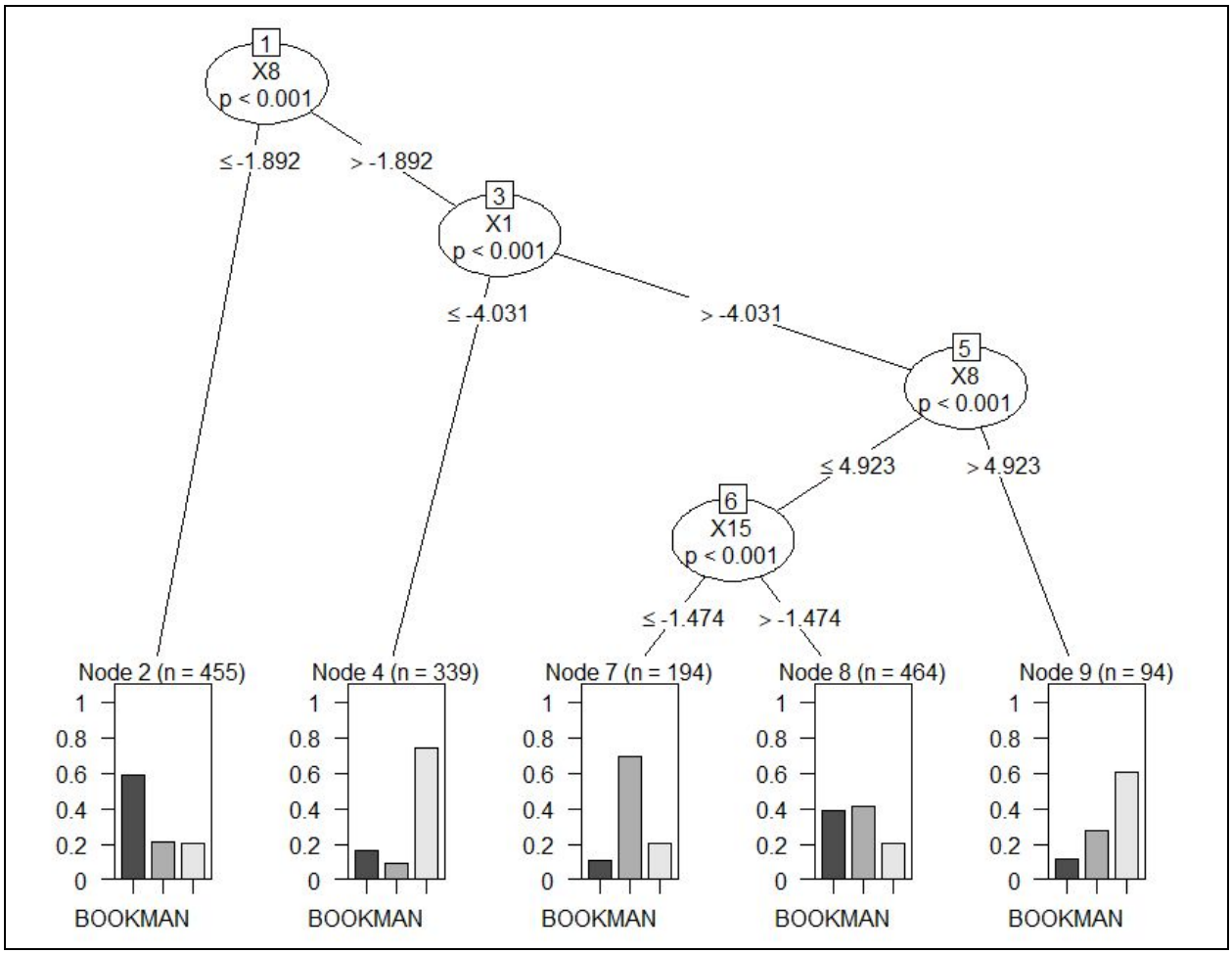
171/(171+458+38) = 0.2563 the predicted value of COMIC and true_set MONOTYPE

Therefore, we look at the class that has the maximum frequency in the cluster j(n) where j(n) is listed in the standard output out$cluster of the k-means function. As we see in our A_mj matrix from the previous question we see that the maximum frequency is COMIC

To get a visual understanding, we decided to explore the decision tree of our data using the R package:*party*. Defined as

tree_cut=ctree(as.factor(TRAINSET_TARGET)~., data = TRAINSET, controls = ctree_control(mincriterion = 0.99, minsplit = 500))

We prune our tree at a *mincriterion* confidence level of 99% (confidence the variable is significant) and *minsplit* of branches to split into 2 when the sample size is 500. By controlling these parameters, it helps restrict the growth of the tree making it less complicated as we can see in the tree below:



Our pruned tree is more compacted with 9 nodes (way less compared to our full tree with 49 nodes). Out of our 400 variables used, the eighth component X8 is our most important to categorize our 3 classes of fonts as can be seen at the root of the

nodes, followed by the first component X1 and the fifteenth component X15. Looking at the lower part of our tree we see the terminal node, also known as the leaves, we can read the probability of our three classes. For example, at Node 2 (n =455) we observe that TRAINSET has a 60% chance to fall under the Bookman font. Note the leaves go by, Bookman, Comic, and Monotype. Following the branches we see node 4 has about 75% chance of Monotype, Node 7 = 70% Comic, Node 8 appears to be almost tied with bookman and Comic, and finally, Node 9 is 60% Monotype.

Conclusion

In this homework, we used clustering to build a predictor. The quality of the predictor depended on how constant the clusters are. After computing clustering and classification using the k-means function, we identified our best $k^* = 3$ clusters. Of course, k-means clustering is not the only way to gather a collection of data points based on their similarities. There are many other methods such as random forest that can be explored in future reports that can help improve our cluster.

R-codes

```
# STEP : Data Setup
rm(list=ls()) # to remove all objects from a specified environment
cat("\f") # to clean console
library(readr)

# 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])
# seperate into classes
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,]
#using 4bind to bind only classes of thoes rows
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: Principal Component Analysis
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

#first 6 eigenvalues
```

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


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)...Yr(n)
#Y1(n) = Q11 * X1(n) + Q21* X2(n) + … + Qr1 * Xr(n)
Qt = t(eigen(R)$vector)[1:6,1:6]; Qt

#-------------------------------------------------------------------------------------------

## Question 1  # K-means Clustering
names(k_out1)
# Reminder SDATA is our standardize 400 feature data set
set.seed(123)
k_out1 <- kmeans(SDATA,1,50)
k_out2 <- kmeans(SDATA,2,50)
k_out3 <- kmeans(SDATA,3,50)
k_out4 <- kmeans(SDATA,4,50)
k_out5 <- kmeans(SDATA,5,50)
k_out6 <- kmeans(SDATA,6,50)
k_out7 <- kmeans(SDATA,7,50)
k_out8 <- kmeans(SDATA,8,50)
k_out9 <- kmeans(SDATA,9,50)
k_out10 <- kmeans(SDATA,10,50)
################################
SWS1 = sum(k_out1$withinss)
SWS2 = sum(k_out2$withinss)
SWS3 = sum(k_out3$withinss)
SWS4 = sum(k_out4$withinss)
SWS5 = sum(k_out5$withinss)
```

```
SWS6 = sum(k_out6$withinss)
SWS7 = sum(k_out7$withinss)
SWS8 = sum(k_out8$withinss)
SWS9 = sum(k_out9$withinss)
SWS10 = sum(k_out10$withinss)

Perf_1 <- 1-(SWS1/SWS1)
Perf_2 <- 1-(SWS2/SWS1)
Perf_3 <- 1-(SWS3/SWS1)
Perf_4 <- 1-(SWS4/SWS1)
Perf_5 <- 1-(SWS5/SWS1)
Perf_6 <- 1-(SWS6/SWS1)
Perf_7 <- 1-(SWS7/SWS1)
Perf_8 <- 1-(SWS8/SWS1)
Perf_9 <- 1-(SWS9/SWS1)
Perf_10 <- 1-(SWS10/SWS1)
k <- c(1:10)
Perf_k <- data.frame(Perf_1,Perf_2,Perf_3,Perf_4,Perf_5,
            Perf_6,Perf_7,Perf_8,Perf_9,Perf_10)
plot(k,Perf_k, type = "b")

# SX = Y[,-1]  # reminder, SX is the reduced model using PCA
k_out1 <- kmeans(SX,1,50)
k_out2 <- kmeans(SX,2,50)
k_out3 <- kmeans(SX,3,50)
k_out4 <- kmeans(SX,4,50)
k_out5 <- kmeans(SX,5,50)
k_out6 <- kmeans(SX,6,50)
k_out7 <- kmeans(SX,7,50)
k_out8 <- kmeans(SX,8,50)
k_out9 <- kmeans(SX,9,50)
k_out10 <- kmeans(SX,10,50)

SWS1 = sum(k_out1$withinss)
SWS2 = sum(k_out2$withinss)
SWS3 = sum(k_out3$withinss)
SWS4 = sum(k_out4$withinss)
SWS5 = sum(k_out5$withinss)
SWS6 = sum(k_out6$withinss)
SWS7 = sum(k_out7$withinss)
SWS8 = sum(k_out8$withinss)
SWS9 = sum(k_out9$withinss)
SWS10 = sum(k_out10$withinss)

Perf_1 <- 1-(SWS1/SWS1)
Perf_2 <- 1-(SWS2/SWS1)
Perf_3 <- 1-(SWS3/SWS1)
Perf_4 <- 1-(SWS4/SWS1)
Perf_5 <- 1-(SWS5/SWS1)
Perf_6 <- 1-(SWS6/SWS1)
Perf_7 <- 1-(SWS7/SWS1)
```

```
Perf_8 <- 1-(SWS8/SWS1)
Perf_9 <- 1-(SWS9/SWS1)
Perf_10 <- 1-(SWS10/SWS1)

k <- c(1:10)
Perf_k <- data.frame(Perf_1,Perf_2,Perf_3,Perf_4,Perf_5,
            Perf_6,Perf_7,Perf_8,Perf_9,Perf_10)
plot(k,Perf_k, type = "b")
```

```
# Question 1 (continued): Elbow Method for choosing the best k
k1 = c(1:20)
plot(k1, sapply(k1, function(k) {kmeans(x = SX, centers = k, nstart = 50, iter.max = 50)$tot.withinss}),
    type = "b", pch = 19, col = "black", frame = FALSE,
    xlim = c(0, max(k1)),
    xlab = "k", ylab = "Total Within-Clusters Sum of Squares",
    main = "Elbow Method")
segments(min(k1), kmeans(x = SX, centers = min(k1), nstart = 50, iter.max = 50)$tot.withinss,
        3, kmeans(x = SX, centers = 3, nstart = 50, iter.max = 50)$tot.withinss,
        col = "red")
segments(3, kmeans(x = SX, centers = 3, nstart = 50, iter.max = 50)$tot.withinss,
        max(k1), kmeans(x = SX, centers = max(k1), nstart = 50, iter.max = 50)$tot.withinss,
        col = "red")

#-----------------------------------------------------------------------------------------------------
# Question 2: K-Means Clustering
kbest = 3

set.seed(123)
KM = kmeans(SX, kbest, nstart = 50, iter.max = 50)

centers = KM$centers
size = KM$size
cluster = KM$cluster # a vector of nrow(DATA) x 1

c = centers[,1:3] #dimension of kbest x 3
print(c)
library(rgl)
plot3d(c, xlab = "x", ylab = "y", zlab = "z", col = "red")

t = max(size)

v = SX[which(cluster == which(size == t)),1:3]
colors = c("red", "green", "blue")
colors = colors[as.numeric(as.factor(true_set))][which(cluster == which(size == t))]
plot3d(v, xlab = "x", ylab = "y", zlab = "z", col = colors)
#-----------------------------------------------------------------------------------------------------
# Question 3: Gini Index
# class 1: BOOKMAN; class 2: COMIC; class3: MONOTYPE
s = function(j) {
  s = length(cluster[cluster==j])
```

```
  return(s)
}

f = function(c) { #class frequencies
  f_class = s(c)/length(cluster)
  return(f_class)
}

for (i in 1:kbest){ # gini indexes for each cluster
  cat(paste0("gin(CLU_",i,") ="), f(i)*(1-f(i)), "\n")
}

gini = function(kbest = 3) { # Impurity IMP(kbest) for clustering CLU_1,...,CLU_k
  gini = 0
  for (i in 1:kbest){
    gini = gini + f(i)*(1-f(i))
  }
  return(gini)
}
cat("Impurity IMP(Kbest) =", gini(kbest = kbest),"\n")

A = function(m, j) {
  A = length(intersect(which(as.integer(as.factor(true_set))==m),which(cluster==j)))
  return(A)
}

fm = function(j) {
  rbind(A(m = 1, j)/s(j), A(m = 2, j)/s(j), A(m = 3, j)/s(j))
}

FREQ_j = c()
for (k in 1:kbest) {
  FREQ_j = append(FREQ_j, fm(k))
}
FREQ_j = matrix(FREQ_j, nrow = 3)
print(FREQ_j) # frequency: dimension of number of classes * kbest

A_mj = c()
for (m in 1:3) {
  for (j in 1: kbest) {
    A_mj = append(A_mj, A(m, j))
  }
}
A_mj = matrix(A_mj, nrow = 3, byrow = TRUE)
print(A_mj)

TOP = function(j) {
  top = which(A_mj[,j]==max(A(1, j), A(2, j), A(3, j)))
  return(top)
}
#-----------------------------------------------------------------------------------------------
```

Question 4: Confusion matrix / Decision Tree (Continued...)

```
# Question 4: Decision Tree (Continued...)
j = function(n) { # standard output out$cluster of the kmeans function
  j = cluster[n]
  return(j)
}
Pred = function(n) {
  Pred = TOP(j(n))
  return(Pred)
}

pred = c()
for (i in 1:nrow(SX)) {
  pred = append(pred, Pred(i), after = length(pred))
}
pred = replace(pred, which(pred==1), "BOOKMAN")
pred = replace(pred, which(pred==2), "COMIC")
pred = replace(pred, which(pred==3), "MONOTYPE")
table_matrix = table(true_set, predict = pred[1:nrow(SX)])
CONF = prop.table(table_matrix, margin  = 1)
print(CONF)
#-------------------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------------------

layout(matrix(c(1), 1, 1))
## Decision Tree with party's package
library(party)
tree_original = ctree(as.factor(TRAINSET_TARGET)~., data = TRAINSET)
tree_cut = ctree(as.factor(TRAINSET_TARGET)~., data = TRAINSET,
                 # to make the tree smaller and less complicated by controlling some parameters
                 # mincriterion is the confidence level
                 # minsplit means the min sample size when the branch will split into two
                 controls = ctree_control(mincriterion = 0.99, minsplit = 500))
plot(tree_original)
plot(tree_cut)
# Predict
predict(tree_original, TESTSET)

## Decision Tree with rpart's package
library(rpart)
tree_1 = rpart(as.factor(TRAINSET_TARGET)~., TRAINSET, method = "class")
library(rpart.plot)
rpart.plot(tree_1)
# Predict
predict(tree_1, TESTSET) #probability of all in test dataset

# Misclassification error for train data
tab1 = table(TRAINSET_TARGET, trainPred = predict(tree_original))
print(tab1)
1-sum(diag(tab1))/sum(tab1)

# Misclassification error with test data
testPred = predict(tree_original, newdata = TESTSET)
tab2 = table(TESTSET_TARGET, testPred)
print(tab2)
1-sum(diag(tab2))/sum(tab2)
```