Classification of Vowel Sounds

Final Project for Multivariate Data Analysis

Bita Nezamdoust

ID: 002332793

December 2019

Introduction

The aim of this project is to classify eleven vowel sounds using multivariate analysis methods. The data set consisting of a training and a test data set, contains 11 classes and 10 predictor variables. The classes correspond to 11 vowel sounds, each contained in 11 different English words. In the training data set, eight speakers spoke each word six times, which makes a total of 528 training observations. In the test data set, seven speakers spoke each word six times, so there are 462 test observations as well. The ten predictors are derived from the digitized speech using complicated standard methods in speech recognition. The dependent variable is the class index for each observation.

In this paper I employ a number of classification and clustering methods to study the classification of the vowel sounds. I use PCA to reduce the dimensionality, employ LDA and QDA for the purpose of classification and compare their performance. I then apply clustering methods such as K-means, Hierarchical clustering methods, and Model-based clustering methods on the data. I end the paper by offering a comparative analysis of the methods used.

Section 1: Principal Component Analysis

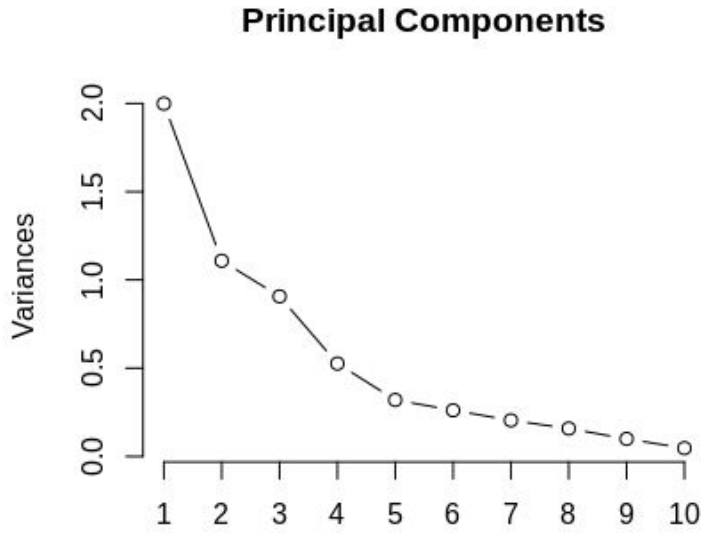
In Section 1, I use Principal Component Analysis (PCA) on the training data set to reduce the dimension of the independent variables while at the same time saving most of the useful information contained in the data.

```
train.data<-read.csv("/home/Multivar_Stat/vowel-train.csv", sep = ",")
test.data <- read.csv("/home/Multivar_Sta/vowel-test.csv", sep = ",")

X.train = train.data[,-c(1:2)]
Y.train = train.data[, 2]
X.test = test.data[,-c(1:2)]
Y.test = test.data[, 2]

pc.cov <- prcomp(X.train)
summary(pc.cov)

plot(pc.cov, type = "l", main = "Principal Components")
```

Output:

Importance of components:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10
Standard deviation	1.4138	1.0529	0.9523	0.72544	0.56583	0.51127	0.45226	0.3978	0.31576	0.21647
Proportion of Variance	0.3549	0.1968	0.1610	0.09345	0.05686	0.04642	0.03632	0.0281	0.01771	0.00832
Cumulative Proportion	0.3549	0.5518	0.7128	0.80627	0.86313	**0.90955**	0.94587	0.9740	0.99168	1.00000



The graph shows that the 1st PC explains most of the variability in the data, and the 2nd PC explains the second most, and so forth until the information becomes less valuable in the later PC's. The output reports that the proportion of variance explained by PC1 is 0.3549, or about 35%, PC2 explains about 20%, PC3 about 16% and so forth. The findings also indicate that the cumulative proportion of variability explained by PC1 through PC6 adds up to 0.90955, or about 91%, which means the first *six* principal components explain most of the variability in the data (above 90%) that is most of the useful information in the data.

Section 2: Linear Discriminant Analysis on Transformed Data

In this section, I employ Linear Discriminant Analysis (LDA) to classify the vowel sounds, on the transformed data by the six principal components that I have selected in section 1. First, I conduct LDA on the training data and calculate the misclassification error based on the training data to be 0.4053. I then apply the obtained linear discriminant rule to the transformed test data set and observe the misclassification error to increase to 0.5909. The code is as follows.

```
lda.fit = lda(data.frame(pc.cov$x[,c(1:6)]), Y.train)
Y.pred = predict(lda.fit, data.frame(pc.cov$x[,c(1:6)]))$class
sum(Y.pred!=Y.train)/length(Y.train) #misclassification error
# 0.405303

To apply the PCA rule from the training data on the test data
pc.train.data = scale(X.train,pc.cov$center,pc.cov$scale)%*%as.matrix(pc.cov$rotation)
pc.test.data = scale(X.test,pc.cov$center,pc.cov$scale)%*%as.matrix(pc.cov$rotation)

lda.fit = lda(pc.train.data[,c(1:6)], Y.train)
Y.pred.lda.pc = predict(lda.fit, pc.test.data[,c(1:6)])$class
sum(Y.pred.lda.pc!=Y.test)/length(Y.test)
# 0.5909091
```

I also created the confusion matrix for the classification of the test data by LDA on the transformed trained data, as can be seen below. I received an accuracy rate of 0.4090 or about 41 percent which I later compared with other methods. It is also interesting to note that accuracy can be obtained by subtracting 1 from the misclassification error. So: 1 - 0.59 = 0.41. Which is consistent with my calculated accuracy from the confusion matrix.

Confusion matrix

```
cm.lda = table(Y.test, Y.pred.lda.pc)
cm.lda
```

		Predicted Class										
		1	2	3	4	5	6	7	8	9	10	11
True class	1	29	10	0	0	0	0	0	0	2	1	0
	2	18	11	9	1	0	0	0	0	1	0	2
	3	0	11	14	17	0	0	0	0	0	0	0
	4	0	0	1	33	0	7	0	0	0	0	1
	5	0	0	0	17	12	3	10	0	0	0	0
	6	0	0	6	7	9	11	2	0	0	0	7
	7	0	0	3	2	21	5	6	3	2	0	0
	8	0	0	0	0	3	0	1	28	7	3	0
	9	0	0	2	0	0	0	5	7	13	10	5
	10	8	2	3	0	0	0	0	0	12	16	1
	11	0	0	6	5	1	5	1	0	8	0	16

Accuracy

```
sum(diag(cm.lda))/sum(cm.lda)
# 0.409090
```

Section 3: Quadratic Discriminant Analysis on Transformed Data

In this section, I apply Quadratic Discriminant Analysis (QDA), in place of LDA, on the transformed data. Similar to section 2, I first conduct QDA on the training data and calculate the misclassification error based on the training data to be 0.1231. I then apply the obtained quadratic discriminant rule to the transformed test data set and observe the misclassification error to increase to 0.4459. Both misclassification rates decreased, in comparison to the equivalent analysis done with LDA. The summary of the results will be presented in tabular form in the next section.

```
qda.fit = qda(pc.train.data[,c(1:6)], Y.train)
Y.pred = predict(qda.fit, pc.train.data[,c(1:6)])$class
sum(Y.pred!=Y.train)/length(Y.train)
# 0.1231061

qda.fit = qda(pc.train.data[,c(1:6)], Y.train)
Y.pred.qda.pc = predict(qda.fit, pc.test.data[,c(1:6)])$class
sum(Y.pred.qda.pc!=Y.test)/length(Y.test)
# 0.4458874
```

Similarly, I created the confusion matrix for the classification of the test data by QDA on the transformed trained data, as can be seen below, and received an accuracy rate of 0.5541 or about 55 percent.

Confusion Matrix

```
cm.qda = table(Y.test, Y.pred.qda.pc)
cm.qda
```

		Predicted class										
		1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
True Class	1	30	12	0	0	0	0	0	0	0	0	0
	2	10	29	3	0	0	0	0	0	0	0	0
	3	0	12	22	4	0	0	4	0	0	0	0
	4	0	0	2	24	0	16	0	0	0	0	0
	5	0	0	0	0	11	21	10	0	0	0	0
	6	0	0	0	1	6	32	0	0	1	0	2
	7	0	0	0	0	2	7	33	0	0	0	0
	8	0	0	0	0	0	0	21	20	0	1	0
	9	0	0	0	1	0	1	3	14	16	3	4
	10	13	4	3	0	0	0	0	0	7	15	0
	11	0	0	0	1	0	5	3	0	9	0	24

Accuracy

```
sum(diag(cm.qda))/sum(cm.qda)
# 0.55411
```

Comparison with the similar accuracy measure obtained for LDA method (to be 0.4049), it is evident that the classification accuracy improved by 15 percent when QDA was employed.

Section 4: LDA and QDA on Original Data

I repeat the procedure in Sections 2 and 3 on the original data before transformation by PCA. All misclassification error rates found in Sections 2, 3 and 4 are summarized in Table 1.

LDA:

```
lda.fit = lda(X.train, Y.train)
Y.pred = predict(lda.fit, X.train)$class
sum(Y.pred!=Y.train)/length(Y.train)
# 0.3162879

lda.fit = lda(X.train, Y.train)
```

```
Y.pred.lda = predict(lda.fit, X.test)$class
sum(Y.pred.lda!=Y.test)/length(Y.test)
# 0.5562771
```

QDA:

```
qda.fit = qda(X.train, Y.train)
Y.pred = predict(qda.fit, X.train)$class
sum(Y.pred!=Y.train)/length(Y.train)
# 0.01136364

qda.fit = qda(X.train, Y.train)
Y.pred.qda = predict(qda.fit, X.test)$class
sum(Y.pred.qda!=Y.test)/length(Y.test)
# 0.5281385
```

Table 1: Summary table of misclassification error using LDA and QDA classification methods.

Classification Method	Data Classified based on	Misclassification error
LDA on Transformed training data	Training data	0.4053
QDA on Transformed training data	Training data	0.1231
LDA on Original training data	Training data	0.3163
QDA on Original training data	Training data	0.0114
LDA on Transformed training data	Testing data	0.5910
QDA on Transformed training data	Testing data	0.4459
LDA on Original training data	Testing data	0.5563
QDA on Original training data	Testing data	0.5281

The table indicates that, among the classification rules that are built on the training data and applied for classifying the training data, QDA on the original data has the lowest misclassification error of 0.0114. Among the classification rules that are built on the training data and applied for classifying the test data, QDA on the transformed data by PCA results in the lowest misclassification error, i.e. 0.4459. Since the models are obtained based on the training data (whether transformed or original training data), the best indication of the model's

performances is when we find the misclassification error of their classification of test data. So according to the models' classification based on test data, it can be concluded that 'QDA on the transformed data, with misclassification error of 0.4459 (and accuracy of 55%) has the best performance.

Section 5: Identifying Hard-to-distinguish Classes

In this section, I attempt to identify hard-to-distinguish classes and repeat the classification methods in their absence. By the analysis that I provide shortly, I found that classes 3, 5, 8, 9 and 10 are harder to classify than the other six. I reached this conclusion by examining the confusion matrix of the best classification that had already been achieved. According to Table 1, QDA technique performed on the transformed data by PCA resulted is the lowest misclassification error. So I examined the confusion matrix associated with it (drawn in Section 3), to see which classes were classified less accurately than the others. The overall accuracy of the classification was found to be over 50 percent, (55.4%), so as a rule of thumb, I considered any class whose accuracy was less than 50 percent to be hard-to-classify in comparison to the others. Therefore, given that there are 42 items in each of the 11 classes in the training data, I took 0.5*42 = 22 as the threshold for satisfactory classification and identified classes who possessed 22 or less number of correct classifications as harder-to-classify compared to others.

```
which(diag(cm.qda)<=22)
# 3  5  8  9  10
```

I then excluded the resulting classes from the data sets and repeated the methods. The results are summarized in Table 2.

```
#Repeat Section 4 for the reduced data:
# New data: Classes 3, 5, 8, 9, 10 removed
indices = Y.train == 1 | Y.train == 2|Y.train == 4| Y.train == 6| Y.train ==
7 | Y.train == 11
Y.train1 = Y.train[indices]
X.train1 = X.train[indices,]

indices = Y.test == 1 | Y.test == 2| Y.test == 4| Y.test == 6| Y.test == 7|
Y.test == 11
Y.test1 = Y.test[indices]
```

```
X.test1 = X.test[indices,]

#####LDA#####
lda.fit = lda(X.train1, Y.train1)
Y.pred = predict(lda.fit, X.train1)$class
sum(Y.pred!=Y.train1)/length(Y.train1)
# [1]  0.2326389

lda.fit = lda(X.train1, Y.train1)
Y.pred.lda = predict(lda.fit, X.test1)$class
sum(Y.pred.lda!=Y.test1)/length(Y.test1)
# [1]  0.3730159
```

Confusion Matrix

```
cm.lda_new = table(Y.test1, Y.pred.lda)
cm.lda_new
```

		Predicted class					
		1	4	6	7	8	11
True class	1	31	11	0	0	0	0
	2	19	17	0	3	0	3
	4	0	0	31	8	0	3
	6	0	0	11	21	1	9
	7	0	1	1	13	26	1
	11	0	0	0	9	1	32

Accuracy

```
sum(diag(cm.lda_new))/sum(cm.lda_new)
#  0.626984

#####QDA####
qda.fit = qda(X.train1, Y.train1)
Y.pred = predict(qda.fit, X.train1)$class
sum(Y.pred!=Y.train1)/length(Y.train1)
# [1] 0.01041667

qda.fit = qda(X.train1, Y.train1)
Y.pred.qda = predict(qda.fit, X.test1)$class
sum(Y.pred.qda!=Y.test1)/length(Y.test1)
# [1]  0.3412698
```

Confusion Matrix

```
cm.qda_new = table(Y.test1, Y.pred.qda)
cm.qda_new
```

		Predicted class					
		1	4	6	7	8	11

	1	38	4	0	0	0	0
	2	18	24	0	0	0	0
True class	4	0	4	14	17	4	3
	6	0	0	1	22	14	5
	7	0	0	0	2	35	5
	11	0	1	4	2	2	33

Accuracy

```
sum(apply(cm.qda_new, 1, max))/sum(cm.qda_new)
#  0.658730
```

Table 2: Summary table of misclassification error using LDA and QDA classification methods on the reduced data with six classes.

Classification Method	Data Classified based on	Misclassification error
LDA on Original training data	Training data	0.2326
LDA on Original training data	Test data	0.3730
QDA on Original training data	Training data	0.1041
QDA on Original training data	Test data	0.3413

Comparison of Table 1 and Table 2 reveals that excluding the hard-to-classify classes noticeably decreased the misclassification error, as the error dropped from 0.5563 to 0.3730 for LDA classifying test data and similarly for QDA, from 0.5281 to 0.3413. So the models improve if we only classifying classes 1, 2, 4, 6, 7 and 11.

Section 6: Clustering Analysis

In this section, I conduct clustering analysis and for the sake of simplicity and better accuracy, I only consider classes 1, 3, 6 and 10. I use a number of clustering methods, namely, K-means, Hierarchical clustering methods, and Model-based clustering methods as I discuss in the following. To analyze the performance, I calculate the Silhouette score for every model and compare them in the end. I also make the confusion matrix and calculate the accuracy for the

best clustering model for the purpose of comparison with the best classification method which will be discussed in the conclusion section.

K-means

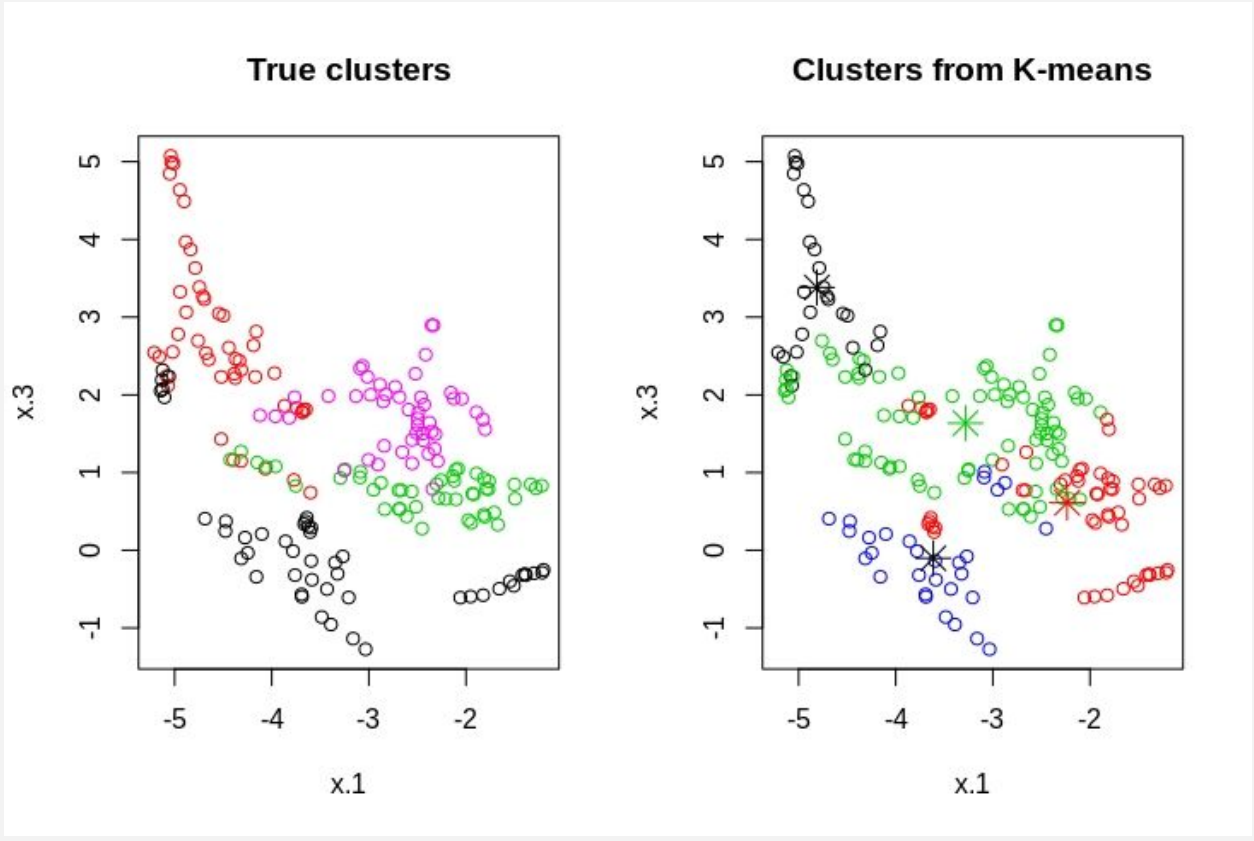
K-means is an unsupervised clustering technique that I use, by denoting k = 4 as the predefined number of clusters.

```
## New data: Classes 1, 3, 6, 10
indices2 = Y.train == 1 | Y.train == 3| Y.train == 6| Y.train == 10
Y.train2 = Y.train[indices2]
X.train2 = X.train[indices2,]

indices2 = Y.test == 1 | Y.test == 3| Y.test == 6| Y.test == 10
X.test2 = X.test[indices2,]
Y.test2 = Y.test[indices2]


###### K-means
library(stats)
km = kmeans(X.train2, 4)$km
```

Output

```
K-means clustering with 4 clusters of sizes 49, 48, 41, 54

Cluster means:
     x.1    x.2    x.3    x.4     x.5    x.6     x.7    x.8     x.9    x.10
1 -3.4102 0.2738 -0.0187 1.228  0.6293 0.5672 -0.1183 -0.4852 -0.6230  0.3409
2 -2.2117 0.5386 -1.4320 0.7391 -0.1448 1.4675 -0.0112  0.8105 -0.5137 -0.4731
3 -4.7899 2.9635  0.1098 0.4805 -0.0227 0.2701  0.0999  0.2524 -0.2213 -0.1995
4 -2.8401 1.7206 -0.4396 0.3812 -0.6693 0.5064  0.1461  0.2916 -0.0733  0.0845

Clustering vector:
  1   3   6  10  12  14  17  21  23  25  28  32  34  36  39  43  45  47  50
54  56  58  61  65  67  69  72
  2   2   4   2   2   2   4   2   2   2   4   2   2   2   4   2   2   2   4
2   2   2   4   2   1   2   4
 76  78  80  83  87  89  91  94  98 100 102 105 109 111 113 116 120 122 124
127 131 133 135 138 142 144 146
  3   1   2   4   3   1   2   4   3   1   2   4   3   1   2   4   3   1   2
4   3   2   2   4   4   2   2
149 153 155 157 160 164 166 168 171 175 177 179 182 186 188 190 193 197 199
201 204 208 210 212 215 219 221
  4   4   2   2   4   3   2   2   4   3   2   2   4   3   2   2   4   3   2
2   4   4   2   2   4   4   2
```

```
223 226 230 232 234 237 241 243 245 248 252 254 256 259 263 265 267 270 274
276 278 281 285 287 289 292 296
  2   4   4   2   2   4   4   2   2   4   4   2   2   4   3   1   1   4   3
1   1   4   3   1   1   4   3
298 300 303 307 309 311 314 318 320 322 325 329 331 333 336 340 342 344 347
351 353 355 358 362 364 366 369
  1   1   4   3   1   1   4   3   1   1   1   3   1   1   4   3   1   1   4
3   1   1   4   3   1   1   4
373 375 377 380 384 386 388 391 395 397 399 402 406 408 410 413 417 419 421
424 428 430 432 435 439 441 443
  3   1   1   4   3   1   1   4   3   1   1   4   3   1   1   4   3   1   1
4   3   1   1   4   3   1   1
446 450 452 454 457 461 463 465 468 472 474 476 479 483 485 487 490 494 496
498 501 505 507 509 512 516 518
  4   3   1   1   4   3   3   1   4   3   3   1   4   3   3   1   4   3   3
1   4   3   3   1   4   3   3
520 523 527
  1   4   3

Within cluster sum of squares by cluster:
[1] 137.7804 137.2747 178.0557 135.8140
 (between_SS / total_SS =  50.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss" "betweenss"    "size"       "iter"         "ifault"


par(mfrow = c(1, 2))
plot(X.train2[c("x.1", "x.2")], col = Y.train2, main = "True clusters", ylab
= "x.3")
plot(X.train2[c("x.1", "x.2")], col = c(1, 3, 6, 10), main = "Clusters from
K-means", ylab = "x.3")
```



Silhouette Score

```
ss.km <- silhouette(km$cluster, dist(X.train2))
mean(ss.km[, 3])
# 0.2694399
```

Confusion Matrix

```
km.test = kmeans(X.test2, 4)
cm.km.test = table(Y.test2, km.test$cluster)
cm.km.test
```

		Predicted class			
		1	2	3	4
True class	1	26	0	16	0
	3	0	25	17	0
	6	0	40	2	0
	10	6	0	12	24

```
sum(apply(cm.km.test, 1, max))/sum(cm.km.test)
# 0.6845238
```

The silhouette score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. The Silhouette score obtained for k-means on the reduced data in 0.2693. A value close to 1 implies that the instance is close to its right cluster, whereas, a value close to -1 means that the value is assigned to a wrong one, so the value found in this analysis is more than half-way close to its accurate class. A side-by-side plot of K-means clustering and the original clusters for classes 1 and 3 are also provided which reveals a rather accurate clustering of the classes.

Agglomerative hierarchical method

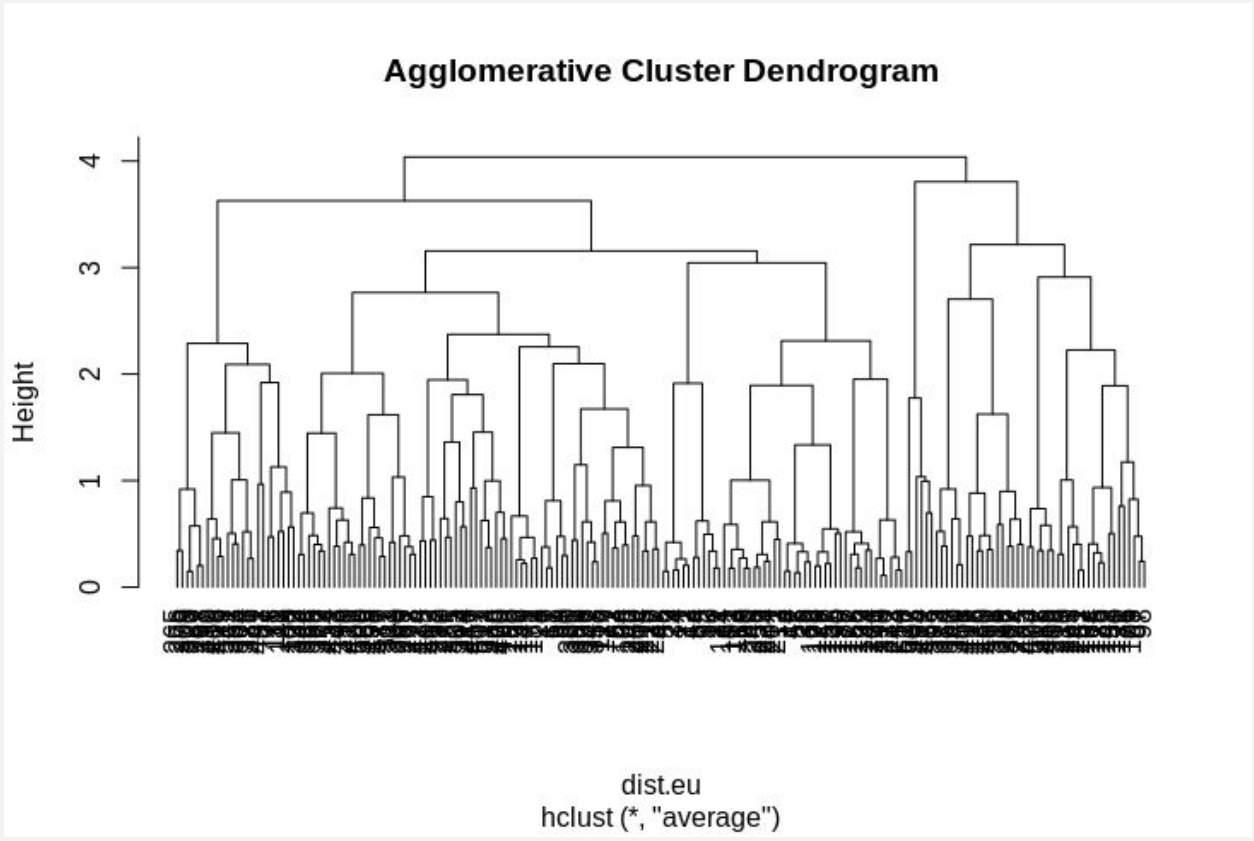
I use complete-linkage, single-linkage and average-linkage agglomerative hierarchical methods and calculate the Silhouette score to evaluate the best performance.

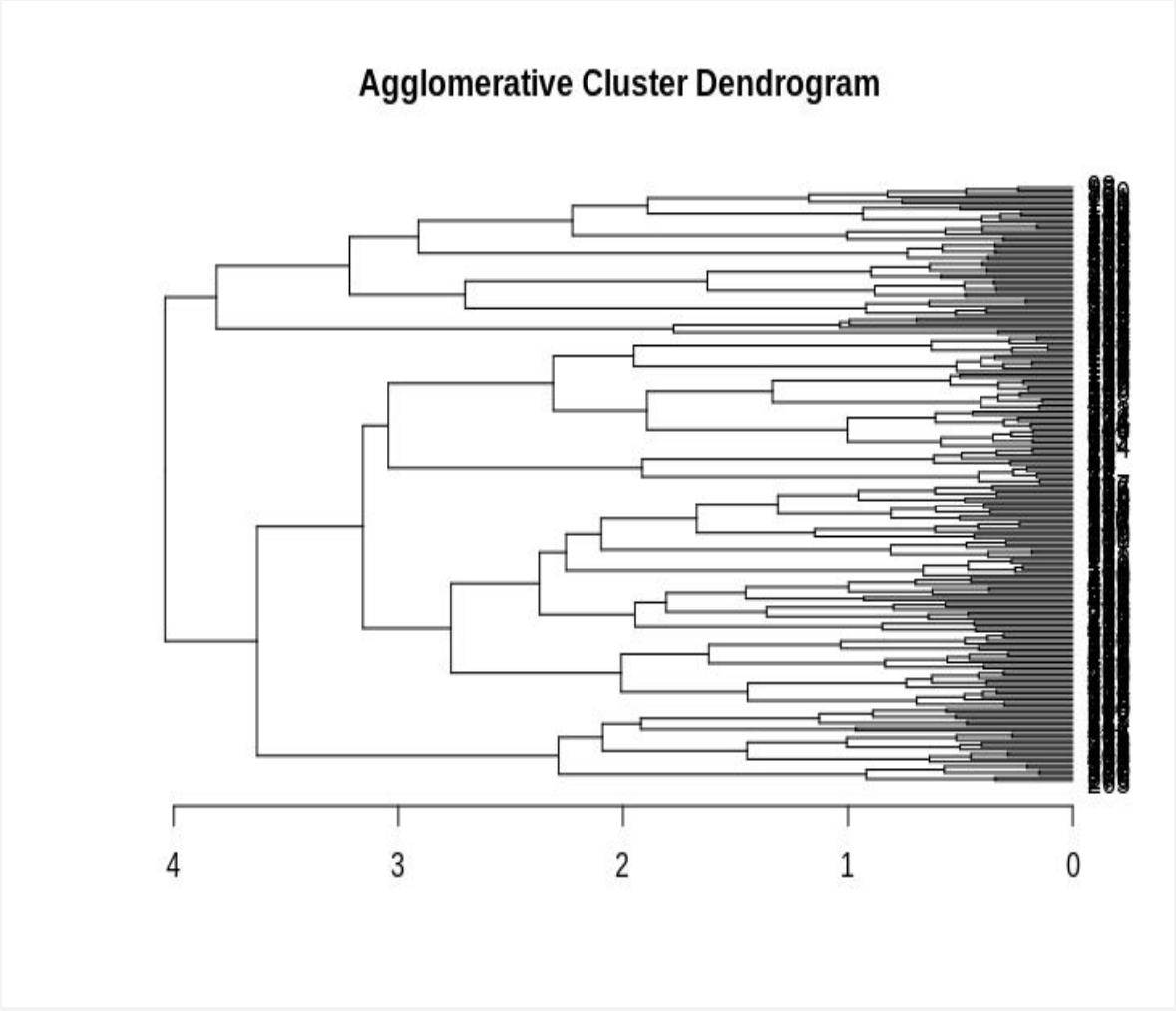
```
#Aglomorotive Hierarchial
Complete-linkage
    dist.eu = dist(X.train2, "euclidean")
    hc.eu.c = hclust(dist.eu, "complete")
    hc.eu.c$merge
    hc.eu.c$height

    hA.cluster = cutree(hc.eu.c, k = 4)
    hA.cluster

Silhouette Score
    hA.cluster = cutree(hc.eu.c, k = 4)

    library(cluster)
    ss.hA <- silhouette(hA.cluster, dist(X.train2))
    mean(ss.hA[, 3])
    # 0.1925788

Single-linkage
    dist.eu = dist(X.train2, "euclidean")
    hc.eu.c = hclust(dist.eu, "single")
    hc.eu.c$merge
    hc.eu.c$height

Silhouette Score
    hA.cluster = cutree(hc.eu.c, k = 4)
    hA.cluster
```

```
    ss.hA <- silhouette(hA.cluster, dist(X.train2))
    mean(ss.hA[, 3])
    # 0.2318668

Average-linkage
    dist.eu = dist(X.train2, "euclidean")
    hc.eu.c = hclust(dist.eu, "average")
    hc.eu.c$merge
    hc.eu.c$height

Silhouette Score
    hA.cluster = cutree(hc.eu.c, k = 4)
    ss.hA <- silhouette(hA.cluster, dist(X.train2))
    mean(ss.hA[, 3])
    # 0.2382757

    hA.cluster
Output:
    1   3   6  10  12  14  17  21  23  25  28  32  34  36  39  43  45  47  50
      1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
     54  56  58  61  65  67  69  72  76  78  80  83  87  89  91  94  98 100 102
      1   1   1   1   1   2   1   1   3   2   1   1   3   2   1   1   3   2   1
    105 109 111 113 116 120 122 124 127 131 133 135 138 142 144 146 149 153 155
      1   3   2   1   1   3   2   1   1   3   1   1   1   3   1   1   1   3   1
    157 160 164 166 168 171 175 177 179 182 186 188 190 193 197 199 201 204 208
      1   1   3   1   1   1   3   1   1   1   3   1   1   1   3   1   1   1   3
    210 212 215 219 221 223 226 230 232 234 237 241 243 245 248 252 254 256 259
      1   1   1   3   1   1   1   3   1   1   1   3   1   1   1   3   1   1   1
    263 265 267 270 274 276 278 281 285 287 289 292 296 298 300 303 307 309 311
      3   2   1   1   3   2   1   1   3   2   1   1   3   2   1   1   3   2   1
    314 318 320 322 325 329 331 333 336 340 342 344 347 351 353 355 358 362 364
      1   3   2   1   1   3   2   1   1   3   2   1   1   3   2   1   1   3   2
    366 369 373 375 377 380 384 386 388 391 395 397 399 402 406 408 410 413 417
      1   1   3   2   1   1   3   2   1   1   3   2   1   1   3   2   1   1   3
    419 421 424 428 430 432 435 439 441 443 446 450 452 454 457 461 463 465 468
      2   1   1   3   2   1   1   3   2   1   1   3   2   1   1   3   3   1   1
    472 474 476 479 483 485 487 490 494 496 498 501 505 507 509 512 516 518 520
      4   3   1   1   4   3   1   1   4   3   1   1   4   3   1   1   4   3   1
    523 527
      1   4
```

Average-linkage hierarchical model has the highest Silhouette score (0.2383), so I select that as the best of the three linkage used. I plot the average-linkage agglomerative hierarchical clustering below.

```
plot(hc.eu.c, hang = -1)
plot(as.dendrogram(hc.eu.c), horiz = T, main = "Agglomerative Cluster
Dendrogram")
```





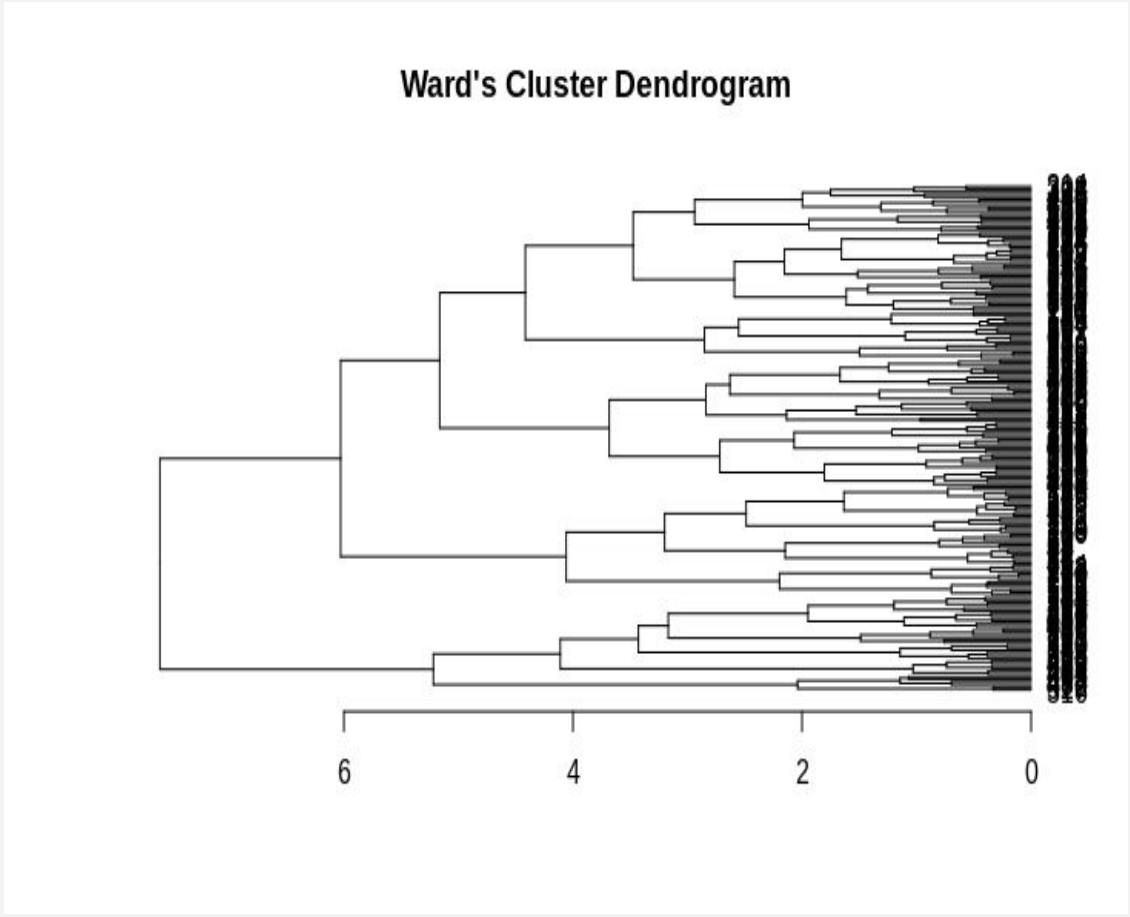
Ward's hierarchical method

I also use Ward's hierarchical method and provide a comparison of the methods in the following.

```
#Ward's hierarchichal
dist.eu = dist(X.train2, "euclidean")
hc.eu.w = hclust(dist.eu, "ward.D")
```

Silhouette Score

```
hW.cluster = cutree(hc.eu.w, k = 4)

ss.hW <- silhouette(hW.cluster, dist(X.train2))
mean(ss.hW[, 3])
# 0.2556523


plot(as.dendrogram(hc.eu.c), horiz = T)
```



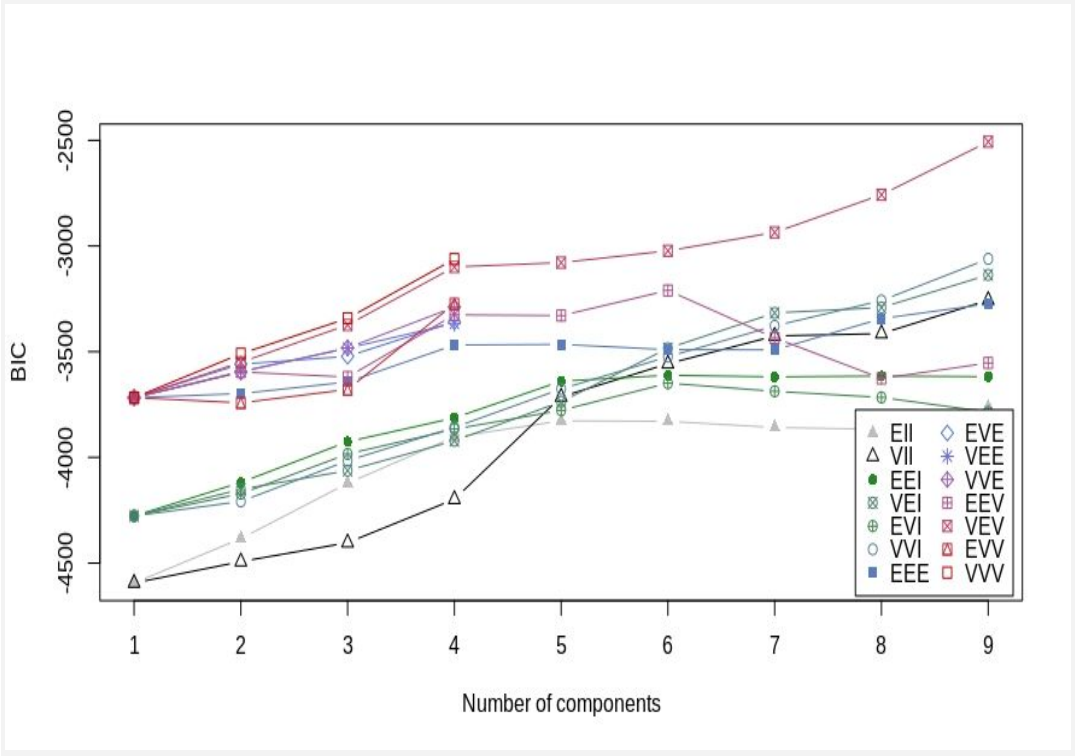
The Silhouette score for Ward's method is 0.2557 which is slightly higher than the score for average-linkage agglomerative method that is 0.2383 and indicates slightly better performance by Ward's method. However, both Silhouette scores are less than that of the K-means which was 0.2694. Therefore, I conclude that K-means, so far, has better clustering outcomes than hierarchical methods.

Model-based methods

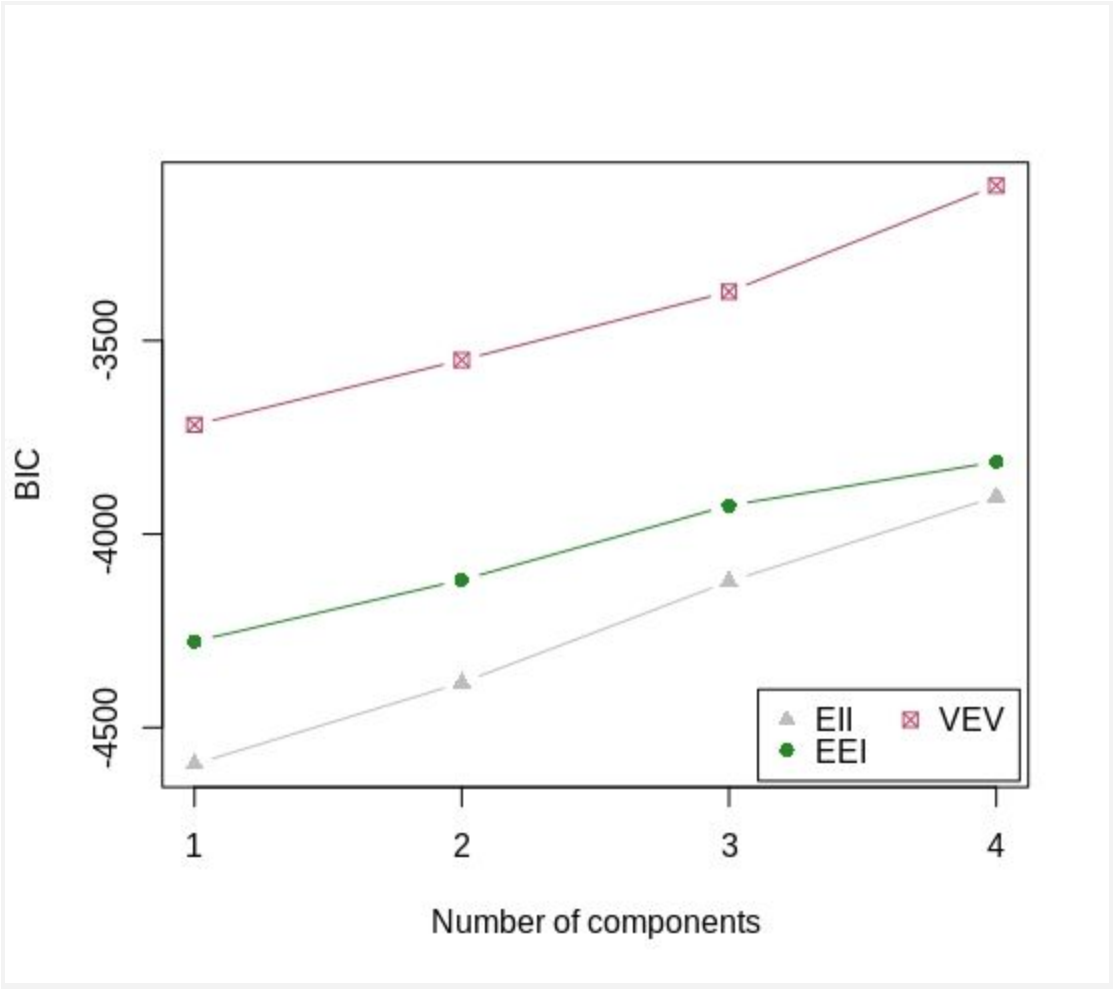
Lastly, model-based clustering technique is used and the appropriate models were selected.

```
##Model-based clustering method
library(mclust)

bic = mclustBIC(X.train2)

bic
```

```
Bayesian Information Criterion (BIC):
         EII       VII       EEI       VEI       EVI       VVI       EEE       EVE       VEE       VVE
1 -4593.968 -4593.968 -4276.959 -4276.959 -4276.959 -4276.959 -3717.485 -3717.485 -3717.485 -3717.485
2 -4383.848 -4492.939 -4118.311 -4150.872 -4171.684 -4208.259 -3698.143 -3558.795 -3593.621 -3596.272
3 -4122.015 -4403.208 -3925.840 -4062.354 -3984.003 -4015.845 -3643.379 -3522.158 -3483.425 -3481.879
4 -3903.197 -4198.143 -3812.997 -3921.615 -3866.220 -3857.845 -3468.339 -3358.364 -3365.478 -3288.637
5 -3826.948 -3713.456 -3639.490 -3736.562 -3777.321 -3676.357 -3465.485        NA        NA        NA
6 -3829.289 -3557.007 -3611.784 -3483.823 -3648.958 -3523.811 -3490.420        NA        NA        NA
7 -3858.691 -3425.471 -3618.969 -3316.600 -3687.510 -3377.946 -3491.336        NA        NA        NA
8 -3867.116 -3414.450 -3615.264 -3291.121 -3716.471 -3258.226 -3343.473        NA        NA        NA
9 -3761.518 -3256.179 -3618.070 -3136.800 -3785.827 -3060.960 -3273.175        NA        NA        NA

         EEV       VEV       EVV       VVV
1 -3717.485 -3717.485 -3717.485 -3717.485
2 -3595.872 -3550.052 -3741.911 -3509.919
3 -3620.048 -3373.475 -3678.724 -3342.168
4 -3325.944 -3099.093 -3273.794 -3061.914
5 -3329.275 -3078.684        NA        NA
6 -3210.201 -3022.798        NA        NA
7 -3435.062 -2935.756        NA        NA
8 -3626.461 -2757.509        NA        NA
9 -3553.500 -2506.888        NA        NA

Top 3 models based on the BIC criterion:
    VEV,9     VEV,8     VEV,7
-2506.888 -2757.509 -2935.756
plot(bic)
```



```
bic1 = mclustBIC(X.train2, G = seq(from=1, to = 4, by = 1),
modelNames=c("EII", "EEI", "VEV"))
plot(bic1)
```




```
mc <- Mclust(X.train2, modelNames = "EEE", G = seq(from=1, to = 4, by = 1))
mc

mc$parameters[1]
# $pro
# [1] 0.5061356 0.1250007 0.2121202 0.1567434

mc$parameters[2]
# $mean
#         [,1]       [,2]       [,3]        [,4]
# x.1  -3.38936229 -3.7642010 -2.4252736 -3.47332315
# x.2   1.97558605 -0.2859090  0.8912642  1.07217066
# x.3  -0.76298401  0.2603258 -0.5356020  0.02717875
# x.4   0.61652088  1.1493330  0.1296061  1.43437302
# x.5  -0.31937492  1.0148275 -0.2494988  0.12146913
```

```
# x.6   0.84976834  1.1489546  0.7208394 -0.09485301
# x.7   0.09912332  0.2010817  0.1629645 -0.51306861
# x.8   0.41902874 -0.6521647  0.5846732 -0.25434272
# x.9  -0.47132131 -0.7717499  0.1411157 -0.32025685
# x.10 -0.19687625  0.1005858 -0.2335314  0.55192451

mc$parameters[3]$variance$sigma
# , , 1
#
# x.1         x.2          x.3         x.4         x.5
# x.1   1.093000052 -0.71531737 -0.502685005  0.08705640 -0.10304205
# x.2  -0.715317372  0.94774973  0.369373879 -0.24346595  0.02927767
# x.3  -0.502685005  0.36937388  0.571877253 -0.05557655 -0.03086917
# x.4   0.087056405 -0.24346595 -0.055576548  0.27965660 -0.06205131
# x.5  -0.103042048  0.02927767 -0.030869174 -0.06205131  0.22306906
# x.6   0.283183943 -0.42616579 -0.258691315  0.15080765 -0.01566427
# x.7  -0.034773834  0.06581358 -0.039013767 -0.06231277  0.01420160
# x.8   0.112155888 -0.18688189 -0.009167003  0.07819787  0.01531764
# x.9  -0.186218812  0.15566314  0.105078053  0.10811581 -0.09998637
# x.10  0.009669443  0.04557435  0.008647089 -0.05049904 -0.08358152
# x.6          x.7          x.8         x.9         x.10
# x.1   0.28318394 -0.034773834  0.112155888 -0.18621881  0.009669443
# x.2  -0.42616579  0.065813578 -0.186881889  0.15566314  0.045574349
# x.3  -0.25869132 -0.039013767 -0.009167003  0.10507805  0.008647089
# x.4   0.15080765 -0.062312772  0.078197866  0.10811581 -0.050499039
# x.5  -0.01566427  0.014201597  0.015317638 -0.09998637 -0.083581519
# x.6   0.36200996 -0.058288760  0.131468285 -0.06770064 -0.044825065
# x.7  -0.05828876  0.134929828 -0.054898931 -0.03248827  0.004466042
# x.8   0.13146828 -0.054898931  0.204266445 -0.06638797 -0.070074968
# x.9  -0.06770064 -0.032488273 -0.066387967  0.30871323  0.031997512
# x.10 -0.04482506  0.004466042 -0.070074968  0.03199751  0.256255045
#
# , , 2
#
# x.1         x.2          x.3         x.4         x.5
# x.1   1.093000052 -0.71531737 -0.502685005  0.08705640 -0.10304205
# x.2  -0.715317372  0.94774973  0.369373879 -0.24346595  0.02927767
# x.3  -0.502685005  0.36937388  0.571877253 -0.05557655 -0.03086917
# x.4   0.087056405 -0.24346595 -0.055576548  0.27965660 -0.06205131
# x.5  -0.103042048  0.02927767 -0.030869174 -0.06205131  0.22306906
# x.6   0.283183943 -0.42616579 -0.258691315  0.15080765 -0.01566427
# x.7  -0.034773834  0.06581358 -0.039013767 -0.06231277  0.01420160
# x.8   0.112155888 -0.18688189 -0.009167003  0.07819787  0.01531764
# x.9  -0.186218812  0.15566314  0.105078053  0.10811581 -0.09998637
# x.10  0.009669443  0.04557435  0.008647089 -0.05049904 -0.08358152
# x.6          x.7          x.8         x.9         x.10
# x.1   0.28318394 -0.034773834  0.112155888 -0.18621881  0.009669443
# x.2  -0.42616579  0.065813578 -0.186881889  0.15566314  0.045574349
```

```
# x.3  -0.25869132 -0.039013767 -0.009167003  0.10507805  0.008647089
# x.4   0.15080765 -0.062312772  0.078197866  0.10811581 -0.050499039
# x.5  -0.01566427  0.014201597  0.015317638 -0.09998637 -0.083581519
# x.6   0.36200996 -0.058288760  0.131468285 -0.06770064 -0.044825065
# x.7  -0.05828876  0.134929828 -0.054898931 -0.03248827  0.004466042
# x.8   0.13146828 -0.054898931  0.204266445 -0.06638797 -0.070074968
# x.9  -0.06770064 -0.032488273 -0.066387967  0.30871323  0.031997512
# x.10 -0.04482506  0.004466042 -0.070074968  0.03199751  0.256255045
#
# , , 3
#
# x.1         x.2          x.3         x.4         x.5
# x.1   1.093000052 -0.71531737 -0.502685005  0.08705640 -0.10304205
# x.2  -0.715317372  0.94774973  0.369373879 -0.24346595  0.02927767
# x.3  -0.502685005  0.36937388  0.571877253 -0.05557655 -0.03086917
# x.4   0.087056405 -0.24346595 -0.055576548  0.27965660 -0.06205131
# x.5  -0.103042048  0.02927767 -0.030869174 -0.06205131  0.22306906
# x.6   0.283183943 -0.42616579 -0.258691315  0.15080765 -0.01566427
# x.7  -0.034773834  0.06581358 -0.039013767 -0.06231277  0.01420160
# x.8   0.112155888 -0.18688189 -0.009167003  0.07819787  0.01531764
# x.9  -0.186218812  0.15566314  0.105078053  0.10811581 -0.09998637
# x.10  0.009669443  0.04557435  0.008647089 -0.05049904 -0.08358152
# x.6          x.7          x.8         x.9         x.10
# x.1   0.28318394 -0.034773834  0.112155888 -0.18621881  0.009669443
# x.2  -0.42616579  0.065813578 -0.186881889  0.15566314  0.045574349
# x.3  -0.25869132 -0.039013767 -0.009167003  0.10507805  0.008647089
# x.4   0.15080765 -0.062312772  0.078197866  0.10811581 -0.050499039
# x.5  -0.01566427  0.014201597  0.015317638 -0.09998637 -0.083581519
# x.6   0.36200996 -0.058288760  0.131468285 -0.06770064 -0.044825065
# x.7  -0.05828876  0.134929828 -0.054898931 -0.03248827  0.004466042
# x.8   0.13146828 -0.054898931  0.204266445 -0.06638797 -0.070074968
# x.9  -0.06770064 -0.032488273 -0.066387967  0.30871323  0.031997512
# x.10 -0.04482506  0.004466042 -0.070074968  0.03199751  0.256255045
#
# , , 4
#
# x.1         x.2          x.3         x.4         x.5
# x.1   1.093000052 -0.71531737 -0.502685005  0.08705640 -0.10304205
# x.2  -0.715317372  0.94774973  0.369373879 -0.24346595  0.02927767
# x.3  -0.502685005  0.36937388  0.571877253 -0.05557655 -0.03086917
# x.4   0.087056405 -0.24346595 -0.055576548  0.27965660 -0.06205131
# x.5  -0.103042048  0.02927767 -0.030869174 -0.06205131  0.22306906
# x.6   0.283183943 -0.42616579 -0.258691315  0.15080765 -0.01566427
# x.7  -0.034773834  0.06581358 -0.039013767 -0.06231277  0.01420160
# x.8   0.112155888 -0.18688189 -0.009167003  0.07819787  0.01531764
# x.9  -0.186218812  0.15566314  0.105078053  0.10811581 -0.09998637
# x.10  0.009669443  0.04557435  0.008647089 -0.05049904 -0.08358152
# x.6          x.7          x.8         x.9         x.10
```

```
# x.1   0.28318394 -0.034773834  0.112155888 -0.18621881  0.009669443
# x.2  -0.42616579  0.065813578 -0.186881889  0.15566314  0.045574349
# x.3  -0.25869132 -0.039013767 -0.009167003  0.10507805  0.008647089
# x.4   0.15080765 -0.062312772  0.078197866  0.10811581 -0.050499039
# x.5  -0.01566427  0.014201597  0.015317638 -0.09998637 -0.083581519
# x.6   0.36200996 -0.058288760  0.131468285 -0.06770064 -0.044825065
# x.7  -0.05828876  0.134929828 -0.054898931 -0.03248827  0.004466042
# x.8   0.13146828 -0.054898931  0.204266445 -0.06638797 -0.070074968
# x.9  -0.06770064 -0.032488273 -0.066387967  0.30871323  0.031997512
# x.10 -0.04482506  0.004466042 -0.070074968  0.03199751  0.256255045
```

Silhouette Score

```
mb.cluster = mc$classification
ss.mb <- silhouette(mb.cluster, dist(X.train2))
mean(ss.mb[, 3])
# 0.1346217
```

The Silhouette score for model-based techniques is 0.1346 which is noticeably less than the other clustering methods used. Comparing all three major categories of clustering, K-means provided the best clustering, with the highest Silhouette score of 0.2693. I also calculated the accuracy of K-means methods which is found to be 0.6845, or about 68 percent, which is a comparatively high score. In the next section I compare this performance with the classification methods.

Section 7: Conclusion

For the purpose of classifying the 11 class that we had, among the classification method used that have been trained on the training data sets and tested on the test data sets, it can be concluded that 'QDA performed on the PCA-transformed data, with misclassification error of 0.4459 (and accuracy of 55%) has the best performance.

In addition, I managed to improve the models by excluding the hard-to-classify classes and only classifying classes 1, 2, 4, 6, 7 and 11 in section 5. By comparing section 4 with section 5, it was found that the misclassification error decreased, as the error dropped from 0.5563 to 0.3730 for LDA classifying test data, and similarly for QDA, from 0.5281 to 0.3413.

Among the clustering methods used to classify classes 1, 3, 6, 10, K-means offered the best Silhouette score of 0.2693 which was higher than the same value obtained by other clustering methods, namely hierarchical methods and model-based methods.

Lastly, I built the confusion matrix for the best classification method and best clustering method, so that I could compare them together. The best classification method was QDA and the best clustering method was K-means. With a subset of classes selected (classes 1, 3, 6, 20), K-means obtained the accuracy of 0.6845 or 68.5%. QDA on PCA-transformed data classifying all 11 classes, obtained accuracy of 0.5541 or 55.4%. However, QDA on original data classifying a subset of classes selected (1, 2, 4, 6, 7, 11), obtained accuracy of 0.6587 or 65.9%. So, it can be concluded that K-means with four classes offered the best performance in predicting the accurate classes.