Analysis on Multivariate Statistics Methods

Li Sun

Objective

The objective of this project is to study eleven vowel sounds with multivariate analysis methods.

Introduction

There are a total 6 questions in the project. First, we are going to conduct a principal component analysis, and decide K principal components to summarize the original training data. Then we need to use K components to conduct Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA), and use these rules to generate misclassification error rates based on training and test data. Second we will apply LDA and QDA on original training and test datasets. Moretherefore, we remove the classes that are most difficult to distinguish and do the comparison on the error rate between original datasets and dataset with classes removed. Finally, we perform hierarchical clustering methods and non hierarchical methods which include K-means and model-based cluster methods to conduct clustering analysis on these observations.

Data Description

There are two datasets: one is training data and the other one is test data. Both of them contain 11 classes and 10 predictors. There are 528 training observations in the training dataset and 462 test observations in the test dataset. The variable y is the class index for each observation, which is the target feature we are going to predict.

Procedure

- Data Preprocess: standardization

Before we start doing learning algorithms, we need to preprocess datasets. We are going to apply PCA for dimensionality reduction and clustering analysis. In PCA, we analyze the variances of the different features and try to find the directions that maximize the variance. In clustering analysis, we compare similarities between features based on distance measures. Hence standardization is very crucial. Due to this fact, the training and test datasets were standardized first.

- Data Analysis

The questions are displayed as fellows one by one.

Question 1

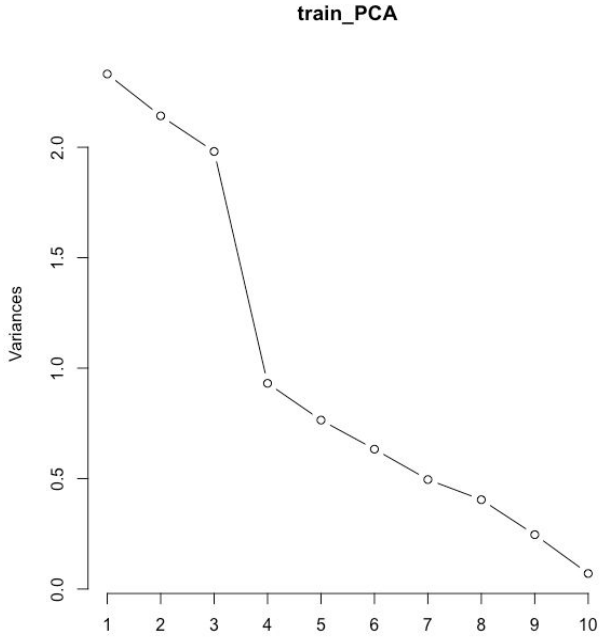
Conduct a principal component analysis for the training dataset. Illustrate the percent that each eigenvalue contributes to the total sample variance. How many principal components will be enough to explain at least 90% of the total sample variance in the training data?

Here I performed two methods, theoretical calculation and one of PCA functions prcomp(), to compute the percent of eigenvalue that contribute to the total sample variance and cumulative percentage.

In theoretical calculation, the eigenvalue and eigenvector for each component were computed. The contribution of each eigenvalue to the total sample variance were calculated based on $\lambda_i / \sum \lambda_i$. In prcomp() function, the contribution of each eigenvalue to the total sample variance are listed in the Proportion of Variance section. The results are 0.23, 0.21, 0.20, 0.09, 0.08, 0.06, 0.05, 0.04, 0.02, 0.01.

In order to find the number of principal components for the requirement, cumulative percentage of variances were calculated. From the cumulative percentage list, we can obtain that the percentage is up to 0.93 when k = 7. So 7 principal components are enough to explain at least 90% of the total sample variance in the training data. The PCA plot in Figure 1 provides details about K vs. variance.

Figure 1. PCA plot of K vs. Variance



Question 2

Suppose that you decide to use K principal components in part (1) to summarize the original training data. Get the scores of these K components for each observation, and then use these scores to conduct the linear discriminant analysis (LDA). What is the misclassification error rate based on the training dataset? Apply the obtained linear discriminant rule to the test dataset, what is the error rate?

7 principal components, which explain 92.79% of total sample variance in training data, were used to summarize the original training data. The scores of these 7 components for each observation were generated in theoretical calculation and prcomp() function coding as well. Both of them gave the same result. The partial scores for training data and test data are listed as follows.

```
> PCA_scores_Train[1:5,]
           PC1        PC2         PC3        PC4         PC5         PC6         PC7
[1,] 1.3130995 -1.6835992  0.19490255 -2.1529643 -0.02911887 -0.09154645  0.08425939
[2,] 1.4502410 -1.6814865  0.02887979 -1.6572374 -0.03710119  0.22856872 -0.27606983
[3,] 2.1036840 -1.0734109 -0.32691897  0.2289168 -1.05631736  0.82158433  0.58493986
[4,] 1.3281319 -0.7965712 -1.28360254 -0.2762440  0.59982614  0.35395581  1.56047190
[5,] 0.4025489  0.1019915 -1.25697825  0.6407106  1.75120891  0.42625892 -0.08543646
> PCA_scores_Test = (as.matrix(test_scaled)%*%train_PCA$rotation) [,1:k]
> PCA_scores_Test[1:5,]
             PC1        PC2        PC3         PC4        PC5        PC6        PC7
[1,]  4.21441400 -1.1290848 -0.7884809  1.55377088  0.1877084 -0.7433545 -1.0473732
[2,]  2.17455661 -1.2933227  0.6907593 -0.50186995  0.4466618 -1.0515826 -1.0352620
[3,]  1.21015421 -0.8665781  0.6488923 -0.03102003  0.1433750 -1.4962093  0.4710935
[4,]  0.84620387  0.2847478 -0.3504204  2.01270515 -0.1119862 -0.9989333  1.3781704
[5,] -0.09968289  1.1365274 -2.5511112  0.49458870 -0.5346641 -0.6267583  0.3317995
```

These PCA scores were input to conduct the LDA. The misclassification error rates based on LDA are 39.02% and 57.36% respectively for training data and test data.

Question 3

Repeat the work in part (2) and do the quadratic discriminant analysis (QDA). Does QDA give lower testing error?

The work in part (2) was repeated, and the PCA scores were utilized to conduct QDA. The misclassification error rates based on QDA are 7.95% and 56.06% respectively for training and test data. Compared with LDA, QDA has a smaller misclassification error rate on both training and test data. The misclassification error rates based on LDA and QDA by using PCA scores are listed in Table 1.

Table 1: Misclassification Error Rates Based on LDA and QDA by using PCA Scores

	LDA	*QDA*
Training Data	39.02%	7.95%
Test Data	57.36%	56.06%

Question 4

For the original training dataset, conduct linear discriminant analysis and quadratic discriminant analysis, and get the error rates. Apply these rules to the test data and get error rates. Summarize these error rates with those from parts (2) and (3) in a table and comment on the results.

The original training dataset was used to conduct LDA and QDA. In order to observe the standardization effects, both unstandardized and standardized datasets were applied on conduction of analysis algorithms, and their error rate was compared. The misclassification error rates based on LDA and QDA by using original datasets are listed in Table 2.

Compared with LDA, for unstandardized original data, QDA has a smaller misclassification error rate on both training and test data. However for standardized original data, the QDA has a higher misclassification error rate on test data.

Table 2: Misclassification Error Rates Based on LDA and QDA by Using Original Datasets

		Unstandardized Original Data	***Standardized Original Data***
LDA	***Training Data***	31.63%	31.63%
	Test Data	55.63%	54.76%
QDA	***Training Data***	1.13%	1.13%
	Test Data	52.81%	58.44%

- **Comments**

The misclassification error rates based on LDA and QDA by using PCA scores and standardized original data are listed in Table 3.

When analysis algorithms are considered, compared with LDA, the training performance of QDA is better. However, the testing performance is different while the input data are different. The testing performance is better by using PCA score, but it is worse by using standardized original data.

When input datasets are considered, compared with standardized original data, the LDA performs based by using PCA score is worse, but QDA testing performance by using PCA scores beats.

Based on the result, QDA has very low error rate on training data, but high error rate on test data, the model is overfitting. So LDA is a better choice. Also compared with PCA scores, the standardized original data gives lower error rate. Overall, the algorithm with LDA with standardized datasets are recommended here.

Table 3: Misclassification Error Rates Based on LDA and QDA
by Using PCA Scores and Standardized Original Datasets

		PCA Scores	***Standardized Original Data***
LDA	***Training Data***	39.02%	31.63%
	Test Data	57.36%	54.76%
QDA	***Training Data***	7.95%	1.13%
	Test Data	56.06%	58.44%

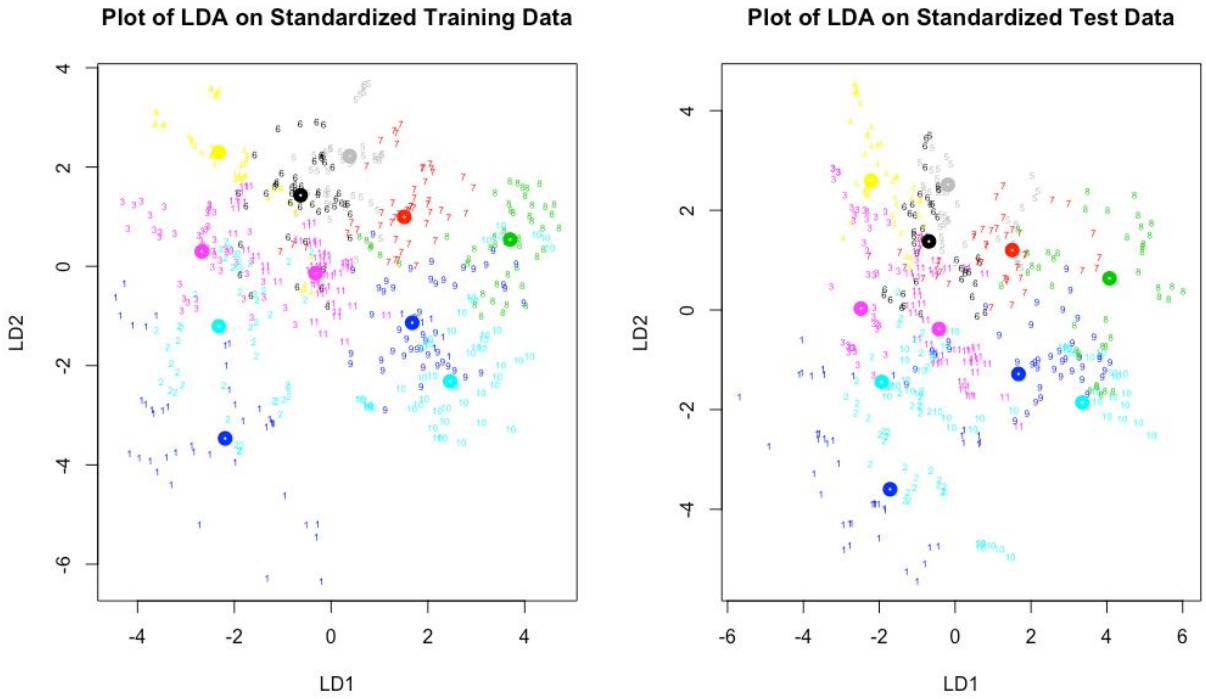
Question 5

What classes do you think are most difficult to distinguish from others? If these classes are removed, how does the test error rate for LDA and QDA change? Compare with results from part (4).

In order to dig out the classes that are most difficult distinguished, F1 score is utilized. F1 score is 2*((precision*recall)/(precision+recall)). It conveys the balance between precision and recall. The F1 score for each class in each prediction was checked and the classes with the smallest F1 was selected. For prediction by using PCA scores, the most difficult distinguished classes are 2 and 10. For prediction by using standardized original datasets, the most difficult distinguished classes are 2, 6 and 8.

Figure 2 provides the visualization of classification based on LDA for standardized original datasets. For training data and test data, the class with the lowest F1 is 6 (black numbers in left plot) and 2 (light blue numbers in right plot) respectively.

Figure 2: Plot of LDA on Standardized Original Datasets



Since the requirement is to do comparison between datasets with classes removal and original datasets, the classes 2, 6 and 8 were removed from the original datasets. The classification error rates are displayed on Table 4.

All of the error rates decrease compared with part (4), which means the better performance. Due to classes removal, there is information loss. For real life application, this impact should be considered.

Table 4: Misclassification Error Rates Based on LDA and QDA
by Using Datasets with Classes Removal and Standardized Original Datasets

		Datasets with Classes Removal	***Standardized Original Data***
LDA	***Training Data***	22.66%	31.63%
	Test Data	43.75%	54.76%
QDA	***Training Data***	0.26%	1.13%
	Test Data	47.62%	58.44%

Question 6

To simplify it, select observations from classes 1,3,6,10 and conduct clustering analysis on these observations. Try the hierarchical clustering methods, K-means method and model-based clustering method. Comment on the performance of these methods.

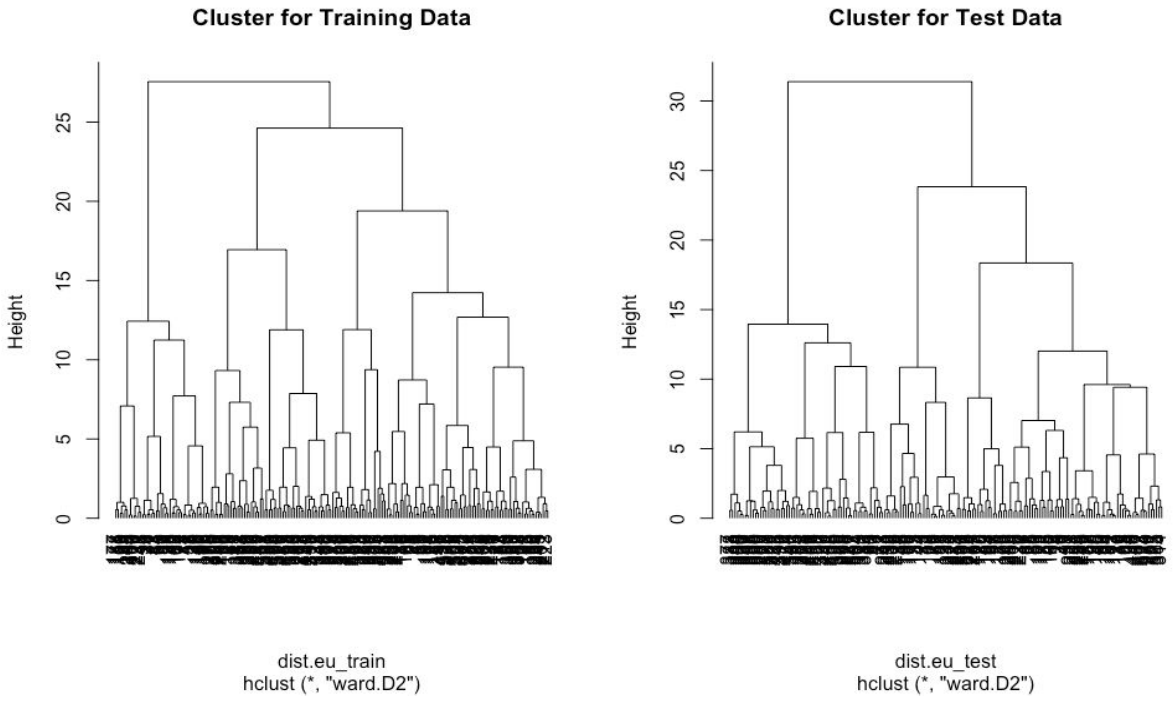
Cluster analysis is an exploratory analysis that tries to identify homogeneous groups of cases if the grouping is not previously known. It is an unsupervised classification method. It does not make any distinction between dependent and independent variables. It is unnecessary to split the dataset into training and test datasets. Due to the fact that two separated datasets are already provided, the average error rate is generated according to the proportion of each dataset. Here average error rage = (7/13*rate(training) + 6/13*rate(test)).

We have a prior knowledge on the number of classes, which is 4. All the clustering methods were implemented to group data into 4 clusters.

- Hierarchical Clustering Methods

Ward's hierarchical clustering method was picked first intuitively since its advantage of minimizing the loss of informing from joining two groups. Figure 3 displays the plot of clustering.

Figure 3: Plot of Ward's Hierarchical Clustering Method



But because by using Ward's the classification error rate 71.88% for training data is too high, Agglomerative hierarchical clustering methods were implemented too. Three methods: complete linkage, single linkage and average linkage were used here.

As previous work, the classification error rates were computed both on unstandardized and standardized datasets. The error rates are listed in Table 5. From the results in the table, there is no reliable method standing out. Even though the ward's hierarchical clustering and single linkage methods have the lowest average rate, 65.08% is still very high. At the same time, it seems that standardized technique does not improve the performance of our models.

Table 5: Classification Error Rate Based on Hierarchical Clustering Methods

		Unstandardized Original Data		***Standardized Original Data***	
			Average rate		***Average rate***
Ward's	***Training Data***	25%	48.90%	71.88%	65.08%
	Test Data	76.79%		57.15%	
Complete	***Training Data***	68.75%	65.04%	65.63%	67.48%
	Test Data	60.71%		69.64%	
Single	***Training Data***	78.13%	65.15%	71.88%	65.08%
	Test Data	50%		57.15%	
Average	***Training Data***	87.8%	70.35%	84.38%	77.57%
	Test Data	50%		69.64%	

- Non Hierarchical Clustering Methods
    - K-means method

K-means method minimizes the total within the classes variance and is attempted to find centroids that are good representatives. The clustering results will be dependent upon the initial partition or initial selection of seed points. To check the stability of the cluster, it is desirable to rerun the algorithm with a new initial partition. So in this project, the algorithm was rerun 10000 times, and the error rates were averaged. The classification error rate is listed in Table 6. The error rates are high on K-means methods.

Table 6: Classification Error Rate Based on K-means Methods after 10000 Times Rerun

	Unstandardized Original Data		***Standardized Original Data***	
		Average rate		***Average rate***
Training Data	75.00%	75.06%	87.50%	81.75%
Test Data	75.12%		75.05%	

- Model-Based clustering method

Model-based clustering method applies assumption on statistical models. It is assumed that the data are generated by a mixture of probability distributions in which each component represents a different cluster. The classification error rates based on Model-based clustering method are listed in Table 7.

Table 7: Classification Error Rate Based on Model-Based Clustering Method

	Unstandardized Original Data		***Standardized Original Data***	
		Average rate		***Average rate***
Training Data	78.13%	76.69%	78.13%	76.69%
Test Data	75%		75%	

- Comments

In the above clustering analysis on standardized datasets, all the error rates are greater than 65.0%. The ward's hierarchical clustering and single linkage methods have the lowest average rate 65.08%.

At the same time, it seems that standardized technique does not improve the performance of our clustering models.

Conclusion

Question 7

- *What is the final conclusion based on the previous methods.*

1. In this project, the discriminant analysis provides higher prediction accuracy than clustering analysis. If the standardized data is utilized, the algorithm with LDA with standardized datasets are recommended here.
2. Since there is labeled target feature, unsupervised algorithm is NOT recommended.
3. Even though standardization does not produce big improvement performance in clustering methods in our project, feature scaling and normalization are very important in data analysis and machine learning. As a rule of thumb, when in doubt, just standardize the data.
4. In our project, one of the reasons that the standardization does not impact big is that the features in the original data do not have big difference ranges.
5. PCA is an unsupervised linear dimensionality reduction algorithm. When we need to tackle the curse of dimensionality, PCA is desired.
6. LDA requires a normal distribution, and QDA assumes each class has its own covariance. So QDA has weaker power on condition requirements. But when the number of predictors is large the number of estimated parameters becomes very large which can lead to a high variance. Therefore, To decide which one is an appropriate method depends.
7. Removal of hard distinguished classes will improve the classification performance, but will suffer from information loss as well.
8. One reason why error rates are not low in this project is that the sample sizes (528 and 462) are not large enough. The other reason is because of multiclass.

Appendent

Question 1

- **Prepare dataset**

```
> # import dataset
> vowel_train=read.table('~/GSU/data/vowel-train.txt',header = T,sep = ',')
> train = vowel_train[,3:12]
> vowel_test=read.table('~/GSU/data/vowel-test.txt',header = T,sep = ',')
> test = vowel_test[,3:12]
```

- **Illustrate the percent that each eigenvalue contributes to the total sample variance**
  - Using the theoretical method

```
> # standardize datasets
> train_scaled = scale(train)
> test_scaled = scale(test)
> # calculate the Principle Component, Proportion of Variance and Cumulative Proportion
> eigenvalue=eigen(cov(train_scaled))$values
> PC =eigen(cov(train_scaled))$vectors
> colnames(PC)=c("PC1","PC2","PC3","PC4","PC5","PC6","PC7","PC8","PC9","PC10")
> row.names(PC)=colnames(train_scaled)
```

```
> PV = CP = 0
> for (i in 1:ncol(train_scaled)){
+   PV[i]=eigenvalue[i]/sum(eigenvalue); CP[i]=sum(eigenvalue[1:i])/sum(eigenvalue)}
> # Principle component
> PC
             PC1         PC2         PC3         PC4         PC5         PC6         PC7
PC8         PC9        PC10
x.1   0.37909964 -0.24932896 -0.18219086  0.62962032 -0.10618937 -0.27606925  0.20560323
-0.054140710 -0.09269720  0.47245760
x.2  -0.22715993 0.55560125 -0.14514476 -0.12861878 -0.06895509  0.32992778 0.35334533
-0.005890131 -0.08703377 0.59667930
x.3 -0.52647855 -0.04252772  0.02574849 -0.11202202 -0.16869468 -0.56081924 -0.41894051
-0.008652676 -0.29861327  0.31633493
x.4  -0.13880889 -0.52477379 -0.16308526 -0.29418866  0.21009282  0.17180117 0.04062774
-0.633885382 0.21434506  0.26294687
x.5   0.06927726 -0.13144617  0.61591153 -0.21889038 -0.03124299 -0.20740231  0.18468152
0.296905825 0.52338156  0.33238851
x.6   0.49091434 -0.20521679 -0.02142017 -0.39613293  0.04738741  0.27240964 -0.37139825
0.313872271 -0.38913604  0.30850797
x.7   0.23709815 0.46061251  0.16361933  0.20741158 0.35659718 -0.01985142 -0.56624216
-0.332495234  0.27247780 0.16984317
x.8   0.19287425 0.10553563 -0.48946542 -0.21081702 -0.61251623 -0.09543594 -0.20991099
0.012325651  0.48888414 -0.01162574
x.9  -0.20917712 -0.08337887 -0.49910899  0.05357916 0.57430433 -0.08351121 -0.05269179
0.515033132  0.29344924  0.07977964
x.10 -0.35393071 -0.24960063  0.15236931  0.43905712 -0.27565793  0.58068978 -0.34088044
0.180320666 0.15491688  0.10267837
> # Proportion of Variance
> PV
 [1] 0.233168251 0.214209523 0.198103010 0.093128860 0.076478137 0.063306115 0.049545302
0.040431175 0.024605298 0.007024331
> # Cumulative Proportion
> CP
 [1] 0.2331683 0.4473778 0.6454808 0.7386096 0.8150878 0.8783939 0.9279392 0.9683704
0.9929757 1.0000000
```

- Call the PCA function directly: prcomp()

```
# Using function prcomp() to find the info about PCA
> train_PCA = prcomp(train_scaled, scale= T)
> summary(train_PCA)
Importance of components:
                          PC1    PC2   PC3     PC4     PC5    PC6     PC7     PC8    PC9    PC10
Standard deviation     1.5270 1.4636 1.4075 0.96503 0.87452 0.79565 0.70388 0.63586 0.49604 0.26503
Proportion of Variance 0.2332 0.2142 0.1981 0.09313 0.07648 0.06331 0.04955 0.04043 0.02461 0.00702
Cumulative Proportion  0.2332 0.4474 0.6455 0.73861 0.81509 0.87839 0.92794 0.96837 0.99298 1.00000
> plot(train_PCA, type = 'l')
```

Question 2

- **Get the scores of these K components for each observation**
  - Using the theoretical method

```
## Using theoretical method
> k=7
> # scaled dataset have ZERO MEAN
> PCA_scores_Train=(train_scaled%*%PC)[,1:k]
> PCA_scores_Train[1:5,]
           PC1        PC2         PC3        PC4         PC5         PC6         PC7
[1,] 1.3130995 -1.6835992  0.19490255 -2.1529643 -0.02911887 -0.09154645 0.08425939
[2,] 1.4502410 -1.6814865  0.02887979 -1.6572374 -0.03710119  0.22856872 -0.27606983
[3,] 2.1036840 -1.0734109 -0.32691897 0.2289168 -1.05631736 0.82158433  0.58493986
[4,] 1.3281319 -0.7965712 -1.28360254 -0.2762440 0.59982614  0.35395581  1.56047190
[5,] 0.4025489  0.1019915 -1.25697825  0.6407106 1.75120891  0.42625892 -0.08543646
> PCA_scores_Test = (test_scaled%*%PC)[,1:k]
> PCA_scores_Test[1:5,]
             PC1        PC2        PC3         PC4        PC5        PC6        PC7
[1,]  4.21441400 -1.1290848 -0.7884809  1.55377088 0.1877084 -0.7433545 -1.0473732
[2,]  2.17455661 -1.2933227  0.6907593 -0.50186995  0.4466618 -1.0515826 -1.0352620
[3,]  1.21015421 -0.8665781  0.6488923 -0.03102003  0.1433750 -1.4962093  0.4710935
[4,]  0.84620387 0.2847478 -0.3504204  2.01270515 -0.1119862 -0.9989333  1.3781704
[5,] -0.09968289  1.1365274 -2.5511112  0.49458870 -0.5346641 -0.6267583  0.3317995
```

- Using the information from prcomp() function

```
> ## Using the information from prcomp() function
> k=7
> train_PCA = prcomp(train, scale= T)
> PCA_scores_Train=train_PCA$x[,1:k]
> PCA_scores_Train[1:5,]
           PC1        PC2         PC3        PC4         PC5         PC6         PC7
[1,] 1.3130995 -1.6835992  0.19490255 -2.1529643 -0.02911887 -0.09154645  0.08425939
[2,] 1.4502410 -1.6814865  0.02887979 -1.6572374 -0.03710119  0.22856872 -0.27606983
[3,] 2.1036840 -1.0734109 -0.32691897  0.2289168 -1.05631736  0.82158433  0.58493986
[4,] 1.3281319 -0.7965712 -1.28360254 -0.2762440  0.59982614  0.35395581  1.56047190
[5,] 0.4025489  0.1019915 -1.25697825  0.6407106  1.75120891  0.42625892 -0.08543646
> PCA_scores_Test = (as.matrix(test_scaled)%*%train_PCA$rotation) [,1:k]
> PCA_scores_Test[1:5,]
             PC1        PC2        PC3         PC4        PC5        PC6        PC7
[1,]  4.21441400 -1.1290848 -0.7884809  1.55377088  0.1877084 -0.7433545 -1.0473732
[2,]  2.17455661 -1.2933227  0.6907593 -0.50186995  0.4466618 -1.0515826 -1.0352620
[3,]  1.21015421 -0.8665781  0.6488923 -0.03102003  0.1433750 -1.4962093  0.4710935
[4,]  0.84620387  0.2847478 -0.3504204  2.01270515 -0.1119862 -0.9989333  1.3781704
[5,] -0.09968289  1.1365274 -2.5511112  0.49458870 -0.5346641 -0.6267583  0.3317995
```

- **Conduct the linear discriminant analysis (LDA) model**

```
> library(MASS)
> PCA_Train = data.frame(cbind(vowel_train['y'], PCA_scores_Train))
> PCA_Train$y  = as.factor(PCA_Train$y)
> PCA_Test = data.frame(cbind(vowel_test['y'], PCA_scores_Test))
> PCA_Test$y = as.factor(PCA_Test$y)
> ### Use these scores to conduct the linear discriminant analysis (LDA)
> lda0 = lda(y~., PCA_Train)
```

- **Calculate the misclassification error rate of LDA**

```
> # Calculate the misclassification error rate
```

```
> mean(PCA_Train$y  != predict(lda0, PCA_Train)$class)
[1] 0.3901515
> mean(PCA_Test$y != predict(lda0, PCA_Test)$class)
[1] 0.5735931
```

- **Set up the Confusion Matrices**

```
> ### Set up the Confusion Matrices
> PCA.lda_Train_table = table(PCA_Train$y, predict(lda0, PCA_Train)$class); PCA.lda_Train_table
    1  2  3  4  5  6  7  8  9 10 11
 1 31 11  0  0  0  0  0  0  0  6  0
 2 13 16 11  0  0  7  0  0  0  0  1
 3  0  4 38  4  0  0  0  0  0  0  2
 4  0  0  0 40  0  7  0  0  0  0  1
 5  0  0  0  0 19  8 17  0  0  0  4
 6  0  2  0  4 11 25  0  0  0  0  6
 7  0  0  1  0 11  2 20  8  5  1  0
 8  0  0  0  0  0  0  6 36  3  3  0
 9  0  0  0  0  1  1  3  6 22 13  2
 10 0  0  0  0  0  0  0  7  3 38  0
 11 0  0  4  1  0  5  0  0  1  0 37
> PCA.lda_Test_table = table(PCA_Test$y, predict(lda0, PCA_Test)$class); PCA.lda_Test_table
    1  2  3  4  5  6  7  8  9 10 11
 1 26 10  0  0  0  0  0  0  2  4  0
 2 23  8  5  0  0  1  0  0  0  0  5
 3  0  6 17 13  0  6  0  0  0  0  0
 4  0  0  0 36  0  6  0  0  0  0  0
 5  0  0  0 10 10 12 10  0  0  0  0
 6  0  0  6 12  7 11  2  0  1  0  3
 7  0  0  3  1 15  5  8  3  3  0  4
 8  0  0  0  0  2  0  0 27 10  3  0
 9  0  2  0  0  0  0  2 11 15  8  4
 10 8  2  6  0  0  0  0  2  4 16  4
 11 0  0  5  0  0  4  1  0  9  0 23
```

Question 3

- **Prepare dataset**

```
> library(MASS)
> PCA_Train = data.frame(cbind(vowel_train['y'], PCA_scores_Train))
> PCA_Train$y  = as.factor(PCA_Train$y)
> PCA_Test = data.frame(cbind(vowel_test['y'], PCA_scores_Test))
> PCA_Test$y = as.factor(PCA_Test$y)
```

- **Conduct the Quadratic Discriminant Analysis (QDA)**

```
> qda0 = qda(y~., PCA_Train)
> mean(PCA_Train$y  != predict(qda0, PCA_Train)$class)
[1] 0.07954545
> mean(PCA_Test$y != predict(qda0, PCA_Test)$class)
[1] 0.5606061
```

- **Set up the Confusion Matrices**

```
> PCA.qda_Train_table = table(PCA_Train$y, predict(qda0, PCA_Train)$class); PCA.qda_Train_table

    1  2  3  4  5  6  7  8  9 10 11
 1 42  6  0  0  0  0  0  0  0  0  0
 2  0 46  2  0  0  0  0  0  0  0  0
 3  0  7 41  0  0  0  0  0  0  0  0
 4  0  0  0 45  0  2  0  0  0  0  1
 5  0  0  0  2 42  4  0  0  0  0  0
 6  1  1  0  2  1 41  0  0  0  0  2
 7  0  0  0  0  0  0 47  0  1  0  0
 8  0  0  0  0  0  0  0 48  0  0  0
 9  1  0  0  0  0  1  2  1 42  1  0
 10 0  0  0  0  0  0  0  0  1 47  0
 11 0  0  0  0  0  1  0  0  2  0 45
> PCA.qda_Test_table = table(PCA_Test$y, predict(qda0, PCA_Test)$class); PCA.qda_Train_table

    1  2  3  4  5  6  7  8  9 10 11
 1 42  6  0  0  0  0  0  0  0  0  0
 2  0 46  2  0  0  0  0  0  0  0  0
 3  0  7 41  0  0  0  0  0  0  0  0
 4  0  0  0 45  0  2  0  0  0  0  1
 5  0  0  0  2 42  4  0  0  0  0  0
 6  1  1  0  2  1 41  0  0  0  0  2
 7  0  0  0  0  0  0 47  0  1  0  0
 8  0  0  0  0  0  0  0 48  0  0  0
 9  1  0  0  0  0  1  2  1 42  1  0
 10 0  0  0  0  0  0  0  0  1 47  0
 11 0  0  0  0  0  1  0  0  2  0 45
```

Question 4

- **Prepare dataset**

```
> Train1 = cbind(vowel_train['y'], train_scaled)
> Test1 = cbind(vowel_test['y'], test_scaled)
> Train1$y = as.factor(Train1$y)
> Test1$y = as.factor(Test1$y)
```

- **Linear Discriminant Analysis (LDA)**
  - Conduct the Linear Discriminant Analysis (LDA)

```
lda1 = lda(y~., Train1)
> mean(Train1$y  != predict(lda1, Train1)$class)  #[1] 0.3162879
[1] 0.3162879
> mean(Test1$y != predict(lda1, Test1)$class) # [1] 0.5562771
[1] 0.547619
```

  - Set up Confusion Matrices

```
> Full.lda_Train_table=table(Train1$y, predict(lda1, Train1)$class);Full.lda_Train_table
    1  2  3  4  5  6  7  8  9 10 11
```

```
  1  32  8  2  0  0  0  0  0  1  5  0
  2  10 28  9  0  0  0  0  0  0  0  1
  3   0  4 42  0  0  0  0  0  0  0  2
  4   0  0  0 36  0  6  0  0  0  0  6
  5   0  0  0  0 33  6  8  0  0  0  1
  6   0  2  0  1 11 23  3  0  0  0  8
  7   0  0  0  0  8  1 33  0  3  0  3
  8   0  0  0  0  0  0  6 34  1  7  0
  9   0  0  0  0  0  0  3  5 29 11  0
  10  0  0  0  0  0  0  0  6  9 33  0
  11  0  0  2  5  0  2  0  0  1  0 38
> Full.lda_Test_table=table(Test1$y, predict(lda1, Test1)$class);Full.lda_Test_table
     1  2  3  4  5  6  7  8  9 10 11
  1  30  8  1  0  0  0  0  0  2  1  0
  2  25 12  3  0  0  0  0  0  0  0  2
  3   0 11 12 19  0  0  0  0  0  0  0
  4   0  0  2 34  0  6  0  0  0  0  0
  5   0  0  0  6 12 12 10  0  0  0  2
  6   0  0  6  7  6 20  2  0  0  0  1
  7   0  0  1  1  8 12 13  4  2  0  1
  8   0  0  0  0  1  0  3 23  8  7  0
  9   0  2  0  0  0  0  2  9 14 10  5
  10  9  1  6  0  0  0  0  1  8 12  5
  11  0  1  2  0  0  8  1  0  2  1 27
```

- **Quadratic Discriminant Analysis (QDA)**
  - Conduct the Quadratic Discriminant Analysis (QDA)

```
> qda1 = qda(y~., Train1)
> mean(Train1$y  != predict(qda1, Train1)$class)  # [1] 0.01136364
[1] 0.01136364
> mean(Test1$y != predict(qda1, Test1)$class) # [1] 0.5281385
[1] 0.5844156
```

  - Set up Confusion Matrices

```
> Full.qda_Train_table=table(Train1$y, predict(qda1, Train1)$class);Full.qda_Train_table
      1  2  3  4  5  6  7  8  9 10 11
  1  48  0  0  0  0  0  0  0  0  0  0
  2   0 48  0  0  0  0  0  0  0  0  0
  3   0  1 47  0  0  0  0  0  0  0  0
  4   0  0  0 48  0  0  0  0  0  0  0
  5   0  0  0  0 47  1  0  0  0  0  0
  6   1  0  0  0  0 45  0  0  0  0  2
  7   0  0  0  0  0  0 48  0  0  0  0
  8   0  0  0  0  0  0  0 48  0  0  0
  9   0  0  0  0  0  0  0  0 48  0  0
  10  0  0  0  0  0  0  0  0  1 47  0
  11  0  0  0  0  0  0  0  0  0  0 48
> Full.qda_Test_table = table(Test1$y, predict(qda1, Test1)$class);Full.qda_Test_table
    1  2  3  4  5  6  7  8  9 10 11
  1  31  6  2  0  0  0  0  0  1  2  0
  2  16 26  0  0  0  0  0  0  0  0  0
```

```
  3  13 14 10  0  0  3  1  0  1  0  0
  4   5  3  3 13  3 10  1  0  0  0  4
  5   0  0  0  0 10 10 22  0  0  0  0
  6   0  0  0  0  0 21 12  0  3  0  6
  7   0  0  0  0  7  1 24  0  5  0  5
  8   0  0  0  0  0  0 16  5 21  0  0
  9   1  0  2  0  0  0  1  8 27  3  0
  10  2  5  4  0  0  0  5  0 16 10  0
  11  1  0  0  1  0  3  2  0 20  0 15
```

Question 5

- **Using F1- measure to evaluate the classification rate of each class**

```
> F1 = function(table){
+   table = as.matrix(table)
+   precision = diag(table) / apply(table, 2, sum)
+   recall = diag(table) / apply(table, 1, sum)
+   f1 = 2 * precision * recall / (precision + recall)
+   return(round(f1,2))
+ }
> F1(PCA.lda_Train_table)
   1    2    3    4    5    6    7    8    9   10   11
0.67 0.40 0.75 0.82 0.42 0.49 0.43 0.69 0.54 0.70 0.73
> F1(PCA.lda_Test_table)
   1    2    3    4    5    6    7    8    9   10   11
0.53 0.23 0.40 0.63 0.26 0.25 0.25 0.64 0.35 0.44 0.54
> F1(PCA.qda_Train_table)
   1    2    3    4    5    6    7    8    9   10   11
0.91 0.85 0.90 0.93 0.92 0.85 0.97 0.99 0.89 0.98 0.94
> F1(PCA.qda_Test_table)
   1    2    3    4    5    6    7    8    9   10   11
0.48 0.34 0.49 0.66 0.23 0.53 0.38 0.53 0.22 0.19 0.69
> F1(Full.lda_Train_table)
   1    2    3    4    5    6    7    8    9   10   11
0.71 0.62 0.82 0.80 0.66 0.53 0.65 0.73 0.63 0.63 0.71
> F1(Full.lda_Test_table)
   1    2    3    4    5    6    7    8    9   10   11
0.57 0.31 0.32 0.62 0.35 0.40 0.36 0.58 0.36 0.33 0.64
> F1(Full.qda_Train_table)
   1    2    3    4    5    6    7    8    9   10   11
0.99 0.99 0.99 1.00 0.99 0.96 1.00 1.00 0.99 0.99 0.98
> F1(Full.qda_Test_table)
   1    2    3    4    5    6    7    8    9   10   11
0.56 0.54 0.32 0.46 0.32 0.47 0.38 0.18 0.40 0.35 0.42
> which.min(F1(PCA.lda_Train_table))  # 2
2
> which.min(F1(PCA.lda_Test_table))   # 2
2
> which.min(F1(PCA.qda_Train_table))  # 2
2
> which.min(F1(PCA.qda_Test_table))   # 10
10
> which.min(F1(Full.lda_Train_table))  # 6
```

```
6
> which.min(F1(Full.lda_Test_table))   # 2
2
> which.min(F1(Full.qda_Train_table))  # 6
6
> which.min(F1(Full.qda_Test_table))   # 8
8
```

- **Visualize the LDA plots**

```
> train.ld2 = predict(train.lda, dimen = 2)
> train.y = vowel_train[,2]
> # calculate of center of each cluster
> centers = matrix(rep(0, 22), ncol=2)
> for (i in 1:11){
+   centers[i, 1:2] = colMeans(train.ld2$x[train.ld2$class==i,])
+ }
> plot(train.ld2$x, type = 'n', xlab = 'LD1', ylab = 'LD2')
> text(train.ld2$x, labels = as.character(train.y), col = 11+as.integer(train.y),cex = 0.5)
> points(centers, col = 11+as.integer(train.y), pch = 1, cex = 1,lwd = 5)
```

```
> test.y = vowel_test[,2]
> # calculate of center of each cluster
> centers = matrix(rep(0, 22), ncol=2)
> for (i in 1:11){
+   centers[i, 1:2] = colMeans(test.ld2$x[test.ld2$class==i,])
+ }
> plot(test.ld2$x, type = 'n', xlab = 'LD1', ylab = 'LD2',main = 'Plot of LDA on Test Data')
> text(test.ld2$x, labels = as.character(test.y), col = 11+as.integer(test.y),cex = 0.5)
> points(centers, col = 11+as.integer(test.y), pch = 1, cex = 1,lwd = 5)
```

- **Remove the classes that are most difficult to distinguish from others**

```
> ### select observation 2,6,8
> # convert y back from factor to int before remove observations in case of factor level issue
> Train1$y = as.numeric(as.character(Train1$y))
> Test1$y = as.numeric(as.character(Test1$y))
> newTrain = Train1[!(Train1$y == 2 | Train1$y == 6 | Train1$y == 8),]
> newTest = Test1[!(Test1$y == 2 | Test1$y == 6 | Test1$y == 8),]
> newTrain$y = as.factor(newTrain$y)
> newTest$y = as.factor(newTest$y)
```

- **Linear Discriminant Analysis (LDA)**

```
> lda1.1 = lda(y~., newTrain)
> mean(newTrain$y  != predict(lda1.1, newTrain)$class)  #[1] 0.3162879 -> [1] 0.25
[1] 0.2265625
> mean(newTest$y != predict(lda1.1, newTest)$class) # [1] 0.5562771 -> [1] 0.4910714
[1] 0.4375
> table(newTrain$y, predict(lda1.1, newTrain)$class)
    1   3  4  5  7  9 10 11
  1 36  3  0  0  0  0  5  4
  3  0 47  1  0  0  0  0  0
  4  0  0 40  2  0  0  0  6
  5  0  0  0 35  9  0  0  4
```

```
  7   0  1  0  8 32  4  0  3
  9   0  0  0  0  9 26 13  0
  10  0  0  0  0  0  7 41  0
  11  0  3  4  0  0  1  0 40
 > table(newTest$y, predict(lda1.1, newTest)$class)
      1  3  4  5  7  9 10 11
  1  35  7  0  0  0  0  0  0
  3   0 21 19  0  0  0  0  2
  4   0  1 35  6  0  0  0  0
  5   0  0  8 24  9  0  0  1
  7   0  3  3 15 17  3  0  1
  9   0  2  0  0  3 19 13  5
  10 12  6  0  0  0  9 11  4
  11  0  9  3  0  1  1  1 27
```

- **Quadratic Discriminant Analysis (QDA)**

```
> qda1.1 = qda(y~., newTrain)
> mean(newTrain$y  != predict(qda1.1, newTrain)$class)  # [1] 0.01136364 ->[1] 0.005208333
[1] 0.002604167
> mean(newTest$y != predict(qda1.1, newTest)$class) # [1] 0.5281385->[1] 0.4107143
[1] 0.4761905
> table(newTrain$y, predict(qda1.1, newTrain)$class)
      1  3  4  5  7  9 10 11
  1  48  0  0  0  0  0  0  0
  3   0 48  0  0  0  0  0  0
  4   0  0 48  0  0  0  0  0
  5   0  0  0 48  0  0  0  0
  7   0  0  0  0 48  0  0  0
  9   0  0  0  0  0 48  0  0
  10  0  0  0  0  0  1 47  0
  11  0  0  0  0  0  0  0 48
  > table(newTest$y, predict(qda1.1, newTest)$class)
      1  3  4  5  7  9 10 11
  1  37  2  0  0  0  1  2  0
  3  14 17  1  0  6  4  0  0
  4   5  5 19  6  3  0  0  4
  5   0  0  0 14 28  0  0  0
  7   0  0  0  7 24  5  0  6
  9   1  2  0  0  2 33  4  0
  10  2  4  0  0  5 16 15  0
  11  1  0  1  0  2 21  0 17
```

Question 6

- **Select observations from classes 1,3,6,10**

```
> ### select observation 1, 3, 6, 10
> simpleTrain = vowel_train[,2:12][(vowel_train$y == 1 | vowel_train$y == 3 | vowel_train$y == 6 |
vowel_train$y == 10),]
```

```
> simpleTest = vowel_test[,2:12][(vowel_test$y == 1 | vowel_test$y == 3 | vowel_test$y == 6 |
vowel_test$y == 10),]
> simpleTrain['yy'] = rep(c(1,2,3,4),48)
> simpleTest['yy'] = rep(c(1,2,3,4), 42)
```

- **Try hierarchical cluster methods (ward's)**
  - On training and test datasets

```
> ### ward's hierarchical clustering methods
> dist.eu_train = dist(simpleTrain[,2:11],'euclidean')
> hc.eu.w_train = hclust(dist.eu_train, 'ward.D2')
> plot(hc.eu.w_train, hang = -1)
> simpleTrain['wardClusters']=cutree(hc.eu.w_train, k= 4)
> mean(simpleTrain[,12] != simpleTrain[,13])
[1] 0.71875
> dist.eu_test = dist(simpleTest[,2:11],'euclidean')
> hc.eu.w_test = hclust(dist.eu_test, 'ward.D2')
> plot(hc.eu.w_test, hang = -1)
> simpleTest['wardClusters']=cutree(hc.eu.w_test, k= 4)
> mean(simpleTest[,12] != simpleTest[,13])
[1] 0.5714286
```

  - Confusion Matrices

```
> table(simpleTrain[,12], simpleTrain[,13])
     1  2  3  4
  1 18  0 30  0
  2 12 12 24  0
  3  6 42  0  0
  4  6 18  0 24
> table(simpleTest[,12], simpleTest[,13])
     1  2  3  4
  1 24  0 18  0
  2  0 24 18  0
  3  0 36  6  0
  4  6  0 18 18
```

- **Try K-means method**
  - On training and test dataset

```
> ite = 10000
> nn = array(rep(0, ite))
> for (i in 1: ite) {
+   simpleTrain.km = kmeans(simpleTrain[,2:11],4)
+   simpleTrain['kmCluster'] = simpleTrain.km$cluster
+   nn[i]= mean(simpleTrain[,12] != simpleTrain[,14])}
> mean(nn)
[1] 0.875
> nn = array(rep(0, ite))
> for (i in 1: ite) {
+   simpleTest.km = kmeans(simpleTest[,2:11],4)
+   simpleTest['kmCluster'] = simpleTest.km$cluster
```

```
+   nn[i]=mean(simpleTest[,12] != simpleTest[,14])}
> mean(nn)
[1] 0.7505339
```

- Confusion Matrices for Error Rate = 0.65625(training data) and 0.57142(test data)

```
> table(simpleTrain[,12], simpleTrain[,14])
    1  2  3  4
  1 18 24  6  0
  2 24  6 18  0
  3  8  0  0 40
  4  6  0  0 42
> table(simpleTest[,12], simpleTest[,14])
    1  2  3  4
  1 24  0 18  0
  2  0 24 18  0
  3  0 30  6  6
  4  6  0 18 18
```

- **Try Model-Based clustering method**
  - On training and test dataset

```
> library(mclust)
> simpleTrain.mc = Mclust(simpleTrain[,-c(1,12,13,14)], G = 4)
> simpleTrain['mcCluster'] = simpleTrain.mc$classification
> mean(simpleTrain[,12] != simpleTrain[,15])
[1] 0.78125
> simpleTest.mc = Mclust(simpleTest[,-c(1,12,13,14)], G = 4)
> simpleTest['mcCluster'] = simpleTest.mc$classification
> mean(simpleTest[,12] != simpleTest[,15])   # [1] 0.75
[1] 0.75
```

  - Confusion Matrices

```
> table(simpleTrain[,12], simpleTrain[,15])
    1  2  3  4
  1 24 24  0  0
  2 12  0 12 24
  3 30  0 18  0
  4 48  0  0  0
> table(simpleTest[,12], simpleTest[,15])
    1  2  3  4
  1 24  0 18  0
  2  0 12 18 12
  3  0 36  6  0
  4  6 18 18  0
```

The following codes are based on the unstandardized datasets

Question 4

- **Prepare dataset**

```
> Train1 = vowel_train[,2:12]
```

```
> Test1 = vowel_test[,2:12]
> Train1$y = as.factor(Train1$y)
> Test1$y = as.factor(Test1$y)
```

- **Linear Discriminant Analysis (LDA)**
  - Conduct the Linear Discriminant Analysis (LDA) model

```
lda1 = lda(y~., Train1)
> mean(Train1$y  != predict(lda1, Train1)$class)  #[1] 0.3162879
[1] 0.3162879
> mean(Test1$y != predict(lda1, Test1)$class) # [1] 0.5562771
[1] 0.5562771
```

- **Quadratic Discriminant Analysis (QDA)**

```
> qda1 = qda(y~., Train1)
> mean(Train1$y  != predict(qda1, Train1)$class)  # [1] 0.01136364
[1] 0.01136364
> mean(Test1$y != predict(qda1, Test1)$class) # [1] 0.5281385
[1] 0.5281385
```

Question 5

- **Using F1- measure to evaluate the classification accuracy of each class**

```
> F1 = function(table){
+   table = as.matrix(table)
+   precision = diag(table) / apply(table, 2, sum)
+   recall = diag(table) / apply(table, 1, sum)
+   f1 = 2 * precision * recall / (precision + recall)
+   return(round(f1,2))}
> which.min(F1(PCA.lda_Train_table))  # 2
2
> which.min(F1(PCA.lda_Test_table))   # 2
2
> which.min(F1(PCA.qda_Train_table))  # 2
2
> which.min(F1(PCA.qda_Test_table))   # 10
10
> which.min(F1(Full.lda_Train_table))  # 6
6
> which.min(F1(Full.lda_Test_table))   # 5
5
> which.min(F1(Full.qda_Train_table))  # 6
6
> which.min(F1(Full.qda_Test_table))   # 8
8
```

- **Visualize the LDA plots**

```
> train.ld2 = predict(train.lda, dimen = 2)
> train.y = vowel_train[,2]
> # calculate of center of each cluster
> centers = matrix(rep(0, 22), ncol=2)
> for (i in 1:11){
+   centers[i, 1:2] = colMeans(train.ld2$x[train.ld2$class==i,])
+ }
```

```
> plot(train.ld2$x, type = 'n', xlab = 'LD1', ylab = 'LD2')
> text(train.ld2$x, labels = as.character(train.y), col = 11+as.integer(train.y),cex = 0.5)
> points(centers, col = 11+as.integer(train.y), pch = 1, cex = 1,lwd = 5)
```

```
> test.y = vowel_test[,2]
> # calculate of center of each cluster
> centers = matrix(rep(0, 22), ncol=2)
> for (i in 1:11){
+   centers[i, 1:2] = colMeans(test.ld2$x[test.ld2$class==i,])
+ }
> plot(test.ld2$x, type = 'n', xlab = 'LD1', ylab = 'LD2',main = 'Plot of LDA on Test Data')
> text(test.ld2$x, labels = as.character(test.y), col = 11+as.integer(test.y),cex = 0.5)
> points(centers, col = 11+as.integer(test.y), pch = 1, cex = 1,lwd = 5)
```

- **Remove the classes that are most difficult to distinguish from others**

```
> ### select observation 5,6,8
> newTrain = vowel_train[,2:12][!(vowel_train$y == 5 | vowel_train$y ==6 | vowel_train$y == 8),]
> newTest = vowel_test[,2:12][!(vowel_test$y == 5 | vowel_test$y == 6 | vowel_test$y == 8),]
> newTrain$y = as.factor(newTrain$y)
> newTest$y = as.factor(newTest$y)
```

- **Linear Discriminant Analysis (LDA)**

```
> lda1.1 = lda(y~., newTrain)
> mean(newTrain$y  != predict(lda1.1, newTrain)$class)  #[1] 0.3162879 -> [1] 0.25
[1] 0.25
> mean(newTest$y != predict(lda1.1, newTest)$class) # [1] 0.5562771 -> [1] 0.4910714
[1] 0.4910714
```

- **Quadratic Discriminant Analysis (QDA)**

```
> qda1.1 = qda(y~., newTrain)
> mean(newTrain$y  != predict(qda1.1, newTrain)$class)  # [1] 0.01136364 ->[1] 0.005208333
[1] 0.005208333
> mean(newTest$y != predict(qda1.1, newTest)$class) # [1] 0.5281385->[1] 0.4107143
[1] 0.4107143
```

Question 6

- **Select observations from classes 1,3,6,10**

```
> ### select observation 1, 3, 6, 10
> simpleTrain = vowel_train[,2:12][(vowel_train$y == 1 | vowel_train$y == 3 | vowel_train$y == 6 |
vowel_train$y == 10),]
> simpleTest = vowel_test[,2:12][(vowel_test$y == 1 | vowel_test$y == 3 | vowel_test$y == 6 |
vowel_test$y == 10),]
> simpleTrain['yy'] = rep(c(1,2,3,4),48)
> simpleTest['yy'] = rep(c(1,2,3,4), 42)
```

- **Try hierarchical cluster methods (ward's)**

```
> ### ward's hierarchical clustering methods
```

```
> dist.eu_train = dist(simpleTrain[,2:11],'euclidean')
> hc.eu.w_train = hclust(dist.eu_train, 'ward.D2')
> plot(hc.eu.w_train, hang = -1)
> simpleTrain['wardClusters']=cutree(hc.eu.w_train, k= 4)
> mean(simpleTrain[,12] != simpleTrain[,13])
[1] 0.25
> dist.eu_test = dist(simpleTest[,2:11],'euclidean')
> hc.eu.w_test = hclust(dist.eu_test, 'ward.D2')
> plot(hc.eu.w_test, hang = -1)
> simpleTest['wardClusters']=cutree(hc.eu.w_test, k= 4)
> mean(simpleTest[,12] != simpleTest[,13])
[1] 0.767857
```

- **Try K-means method**

```
> ite = 10000
> nn = array(rep(0, ite))
> for (i in 1: ite) {
+   simpleTrain.km = kmeans(simpleTrain[,2:11],4)
+   simpleTrain['kmCluster'] = simpleTrain.km$cluster
+   nn[i]= mean(simpleTrain[,12] != simpleTrain[,14])}
> mean(nn)
[1] 0.7499474
> nn = array(rep(0, ite))
> for (i in 1: ite) {
+   simpleTest.km = kmeans(simpleTest[,2:11],4)
+   simpleTest['kmCluster'] = simpleTest.km$cluster
+   nn[i]=mean(simpleTest[,12] != simpleTest[,14])}
> mean(nn)
[1] 0.7512036
```

- **Try Model-Based clustering method**

```
> library(mclust)
> simpleTrain.mc = Mclust(simpleTrain[,-c(1,12,13,14)], G = 4)
> simpleTrain['mcCluster'] = simpleTrain.mc$classification
> mean(simpleTrain[,12] != simpleTrain[,15])
[1] 0.78125
> simpleTest.mc = Mclust(simpleTest[,-c(1,12,13,14)], G = 4)
> simpleTest['mcCluster'] = simpleTest.mc$classification
> mean(simpleTest[,12] != simpleTest[,15])   # [1] 0.75
[1] 0.75
```