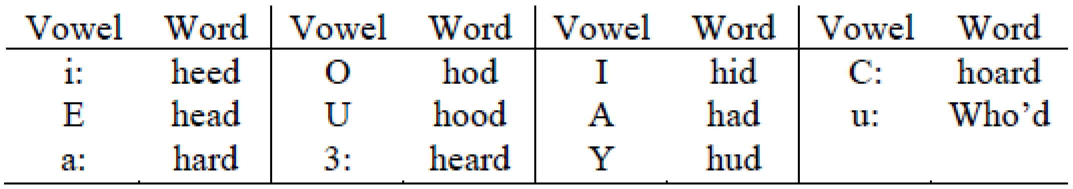
Introduction

This project aims to study the classification of eleven vowel sounds with multivariate analysis methods. The datasets used here are “*vowel-train*” and “*vowel-test*”. Both the training data and test data contain 11 classes and 10 predictors. The classes correspond to 11 vowel sounds, each contained in 11 different words. Here are the words, preceded by the symbols that represent them:



Vowel	Word	Vowel	Word	Vowel	Word	Vowel	Word
i:	heed	O	hod	I	hid	C:	hoard
E	head	U	hood	A	had	u:	Who’d
a:	hard	3:	heard	Y	hud		

In the training data set, each of eight speakers spoke each word six times. There are thus 528 training observations. In the test data set, each of seven speakers spoke each word six times. There are thus 462 test observations. The ten predictors (x.1,…,x.10) are derived from the digitized speech in a rather complicated way, but standard in the speech recognition world. The variable y is the class index for each observation. In this project, we will conduct a principal component analysis (PCA), linear discriminant analysis (LDA), quadratic discriminant analysis (QDA) and clustering analysis.

(1)

```
> library(MASS)
> library(plyr)
> library(mclust)
> library(stats)
> train <- read.table("…/vowel-train.txt",sep=",",header=T)
> test <- read.table("…/vowel-test.txt",sep=",",header=T)
> train <- train[,-1]
> test <- test[,-1]
> train$y <- as.factor(train$y)
> test$y <- as.factor(test$y)
> s <- cov(train[,-1]); print(s, digits = 3)
```

Covariance matrix for training data set:

```
         x.1     x.2      x.3     x.4      x.5     x.6      x.7     x.8      x.9     x.10
x.1   0.9177 -0.5725 -0.30617  0.0138 -0.11690  0.1503 -0.02311  0.1143  0.01096 -0.05099
x.2  -0.5725  1.3479  0.06227 -0.3832 -0.29450 -0.3319  0.12837  0.1065  0.06296 -0.08411
x.3  -0.3062  0.0623  0.54962  0.0778 -0.00621 -0.2573 -0.10049 -0.0570  0.08187  0.12861
x.4   0.0138 -0.3832  0.07777  0.5919 -0.04438  0.0607 -0.20565 -0.0256  0.13113  0.08685
x.5  -0.1169 -0.2945 -0.00621 -0.0444  0.52130  0.0541  0.00255 -0.2067 -0.23992  0.02131
x.6   0.1503 -0.3319 -0.25728  0.0607  0.05412  0.4206  0.01074  0.0878 -0.05878 -0.10927
x.7  -0.0231  0.1284 -0.10049 -0.2057  0.00255  0.0107  0.22968 -0.0184 -0.06696 -0.08250
x.8   0.1143  0.1065 -0.05700 -0.0256 -0.20670  0.0878 -0.01844  0.3547  0.05159 -0.10031
x.9   0.0110  0.0630  0.08187  0.1311 -0.23992 -0.0588 -0.06696  0.0516  0.38388 -0.00177
x.10 -0.0510 -0.0841  0.12861  0.0869  0.02131 -0.1093 -0.08250 -0.1003 -0.00177  0.31396
```

```
> print(eigen(s),digits=3)

eigen() decomposition
$values
 [1] 1.9987 1.1085 0.9068 0.5263 0.3202 0.2614 0.2045 0.1583 0.0997 0.0469

$vectors
         [,1]     [,2]    [,3]     [,4]      [,5]      [,6]     [,7]   [,8]   [,9]   [,10]
 [1,]  0.4802  0.48519  0.1859  0.51249  0.101286  0.095045 -0.23565  0.046  0.162 -0.3654
 [2,] -0.7788  0.27818  0.0399 -0.04738  0.254244  0.178779 -0.18279  0.109  0.172 -0.3756
 [3,] -0.1627 -0.49115  0.2207  0.31756 -0.598890 -0.021712 -0.03082 -0.125  0.299 -0.3433
 [4,]  0.2229 -0.35393  0.4304 -0.42648  0.342367  0.175486 -0.34300 -0.309 -0.120 -0.2867
 [5,]  0.1301 -0.30142 -0.5916 -0.00467 -0.000172  0.000453 -0.32765  0.454 -0.276 -0.3905
 [6,]  0.2481  0.18408 -0.1383 -0.57109 -0.162966 -0.006532  0.39353  0.110  0.472 -0.3783
 [7,] -0.0766  0.18715 -0.2320  0.05203 -0.040510 -0.356838  0.22154 -0.609 -0.429 -0.4112
 [8,] -0.0193  0.29748  0.2415 -0.20251 -0.547539  0.451892 -0.00166  0.157 -0.531 -0.0405
 [9,] -0.0431  0.00138  0.4915 -0.04946  0.055632 -0.655530  0.12402  0.489 -0.216 -0.1435
[10,]  0.0139 -0.26915  0.0835  0.28293  0.343691  0.407964  0.68247  0.148 -0.172 -0.1963
```

The 1st column in the eigenvectors output shows the coefficients of the linear combination that defines PC1, and so on. For example, the principal component function for PC2 is:

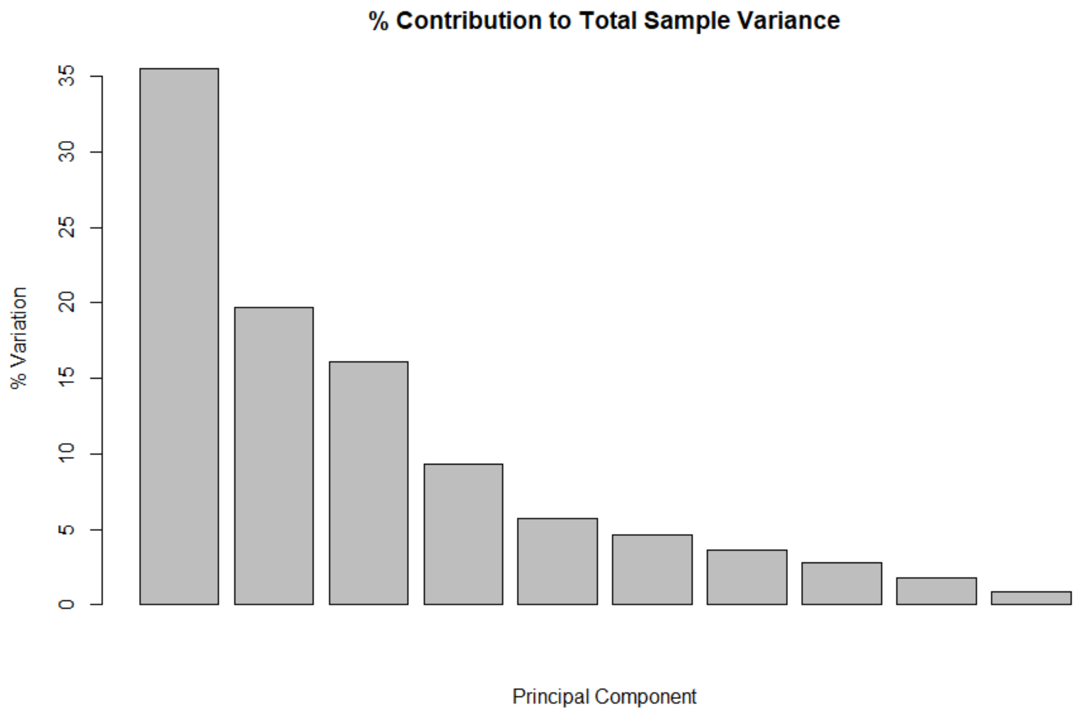
$$y_2 = 0.485x_1 + 0.278x_2 - 0.491x_3 - 0.354x_4 - 0.301x_5 + 0.184x_6 + 0.187x_7 + 0.297x_8 + 0.00138x_9 - 0.269x_{10}$$

A principal component analysis for the training data set is conducted based on the covariance matrix of the training data set. Because these variables are measured on the same unit and scales, we use the covariance matrix rather than the correlation matrix.

```
> pca <- prcomp(train[,-1])
> summary(pca) #First 6 principal components explain 90.955%


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
```

Below lists each eigenvalue and their percentage contribution to the total sample variance:

```
> contribution <- round(pca$sdev^2/sum(pca$sdev^2)*100, 2)
> cumulative_contrib <- cumsum(contribution)
> cbind(pca$sdev^2,contribution,cumulative_contrib)


                  contribution cumulative_contrib
 [1,] 1.99872683         35.49              35.49
 [2,] 1.10852912         19.69              55.18
 [3,] 0.90679436         16.10              71.28
 [4,] 0.52626516          9.35              80.63
 [5,] 0.32016650          5.69              86.32
 [6,] 0.26140035          4.64              90.96
 [7,] 0.20453666          3.63              94.59
 [8,] 0.15825270          2.81              97.40
 [9,] 0.09970595          1.77              99.17
[10,] 0.04685914          0.83             100.00


> barplot(contribution, main = "% Contribution to Total Sample Variance", ylab="% Variati
on", xlab="Principal Component")
```



We can see that 6 principal components are enough to explain at least 90% (90.955%) of the total sample variance in the training data. Therefore 6 components should be retained.

(2)

In this step, we use 6 principal components to summarize the original training dataset.
We get the scores of these 6 components for each observation and then use these scores to conduct the linear discriminant analysis and apply the obtained linear discriminant rule to the test dataset.

Scores of K = 6 components for first six observations:

```
> head(pca$x)[,1:6]

            PC1        PC2        PC3       PC4        PC5         PC6
[1,] 1.44682444  0.6524854 -0.1622952 1.6011797 -0.3194216 -0.23249282
[2,] 1.50295973  0.3238943 -0.1284640 1.3654673 -0.3096981 -0.15155894
[3,] 1.52199174 -1.1916511 -0.2594978 0.2220821  0.1442669 -0.59951027
[4,] 0.70678442 -1.1059725  0.5905678 0.5652180  0.7153274  0.07815277
[5,] 0.13668803 -0.7318058  0.9856730 0.1505875  0.7210824  0.68371804
[6,] 0.01172622 -0.5741844  0.8176760 0.1808969  0.6069952  0.69829920
```

Confusion matrix from LDA of training data set:

```
> new <- data.frame(train$y, pca$x[,1:6])
> ld <- lda(train.y ~ ., data=new)
> ld_train <- predict(ld, new[,-1])
> table(train$y, ld_train$class, dnn=c("Actual Class", "Predicted Class"))
```

```
             Predicted Class
Actual Class  1  2  3  4  5  6  7  8  9 10 11
          1  31 11  0  0  0  0  0  0  0  6  0
          2  15 14 13  4  0  1  0  0  0  0  1
          3   0  2 44  1  0  1  0  0  0  0  0
          4   0  0  3 34  0  8  0  0  0  0  3
          5   0  0  0  1 19  9 17  0  0  0  2
          6   0  2  3  8 12 18  1  0  0  0  4
          7   0  0  0  0 11  3 24  5  5  0  0
          8   0  0  0  0  0  0  6 36  0  6  0
          9   0  0  0  0  0  2  2  7 20 12  5
          10  0  0  0  0  0  0  0  6  4 36  2
          11  0  0  2  3  0  1  0  0  4  0 38
> sum(ld_train$class!=train$y)/length(train$y)
```

The misclassification error rate = 0.405303

After applying the obtained linear discriminant rule to the test data set, we get the resulting confusion matrix:

```
> new_test <- predict(pca, test[,-1])
> new_test <- data.frame(new_test[,1:6])
> ld_test <- predict(ld, new_test)
> table(test$y, ld_test$class, dnn=c("Actual Class", "Predicted Class"))

             Predicted Class
Actual Class  1  2  3  4  5  6  7  8  9 10 11
          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
```

Its misclassification error rate = 0.5909091

(3)

We repeat the work done in (2) and conduct a quadratic discriminant analysis on the training data

```
> qd <- qda(train.y ~ ., data=new)
> qd_train  <- predict(qd, new[,-1])
> table(train$y, qd_train$class, dnn=c("Actual Class", "Predicted Class"))

             Predicted Class
Actual Class  1  2  3  4  5  6  7  8  9 10 11
          1  40  8  0  0  0  0  0  0  0  0  0
          2   0 42  6  0  0  0  0  0  0  0  0
          3   1  4 43  0  0  0  0  0  0  0  0
          4   0  0  0 43  0  2  0  0  0  0  3
          5   0  0  0  1 44  2  1  0  0  0  0
          6   0  0  2  3  8 33  0  0  0  0  2
          7   0  0  0  0  1  0 45  0  2  0  0
          8   0  0  0  0  0  0  0 48  0  0  0
          9   0  0  0  0  0  0  4  4 37  1  2
          10  0  0  0  0  0  0  0  0  3 45  0
```

Its misclassification error rate = 0.1231061

After applying the obtained quadratic discriminant rule to the test data set, we get the resulting confusion matrix:

```
> qd_test <- predict(qd, new_test)
> table(test$y, qd_test$class, dnn=c("Actual Class", "Predicted Class"))

            Predicted Class
Actual Class  1  2  3  4  5  6  7  8  9 10 11
          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
```

Its misclassification error rate = 0.4458874

Quadratic discriminant analysis gives a lower testing error rate, as the difference between the testing error rate for linear discriminant analysis and QDA is +0.1450216, therefore quadratic discriminant analysis should be used instead of LDA.

(4)

Here, we conduct an LDA and QDA on the original training data set, i.e. without using any principal components to summarize the data set.

LDA for Training data:

```
> lda_4 <- lda(y~., data=train)
> lda_train <- predict(lda_4,train[,-1])
> table(train$y, lda_train$class, dnn=c("Actual Class", "Predicted Class"))

            Predicted Class
Actual Class  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
```

Error rate = 0.3162879

Applied to test data:

```
> lda_test <- predict(lda_4, test[,-1])
> table(test$y, lda_test$class, dnn=c("Actual Class", "Predicted Class"))

            Predicted Class
Actual Class  1  2  3  4  5  6  7  8  9 10 11
          1  28 10  1  0  0  0  0  0  3  0  0
          2  23 16  2  0  0  1  0  0  0  0  0
          3   0 11 16 11  0  4  0  0  0  0  0
          4   0  0  2 33  0  6  0  0  0  0  1
          5   0  0  0  1  7 22  9  0  0  0  3
          6   0  0  5  3  8 19  1  0  0  0  6
          7   0  0  1  0  9 12 11  4  4  0  1
          8   0  0  0  0  1  0  2 23  8  8  0
          9   0  2  0  0  0  0  0  6 15 14  5
          10  8  1  5  0  0  0  0  0  9 13  6
          11  0  1  2  0  0 11  1  0  2  1 24
```

Error rate = 0.5562771

QDA for Training data:

```
> qda_4 <- qda(y~., data=train)
> qda_train <- predict(qda_4,train[,-1])
> table(train$y, qda_train$class, dnn=c("Actual Class", "Predicted Class"))

            Predicted Class
Actual Class  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
```

Error rate = 0.01136364

Applied to test data:

```
> qda_test <- predict(qda_4, test[,-1])
> table(test$y, qda_test$class, dnn=c("Actual Class", "Predicted Class"))

            Predicted Class
Actual Class  1  2  3  4  5  6  7  8  9 10 11
          1  37  4  0  0  0  0  0  0  1  0  0
          2  18 22  1  0  0  0  0  0  1  0  0
          3   9 13 12  5  0  2  0  0  1  0  0
          4   0  2  3 12  5 17  2  0  0  0  1
          5   0  0  0  0 16  7 19  0  0  0  0
          6   0  0  0  1  0 22 14  0  0  0  5
          7   0  0  0  0 11  1 22  0  3  0  5
          8   0  0  0  0  0  0 15  6 21  0  0
          9   0  0  0  0  0  0  3  1 38  0  0
          10  2  4  0  0  0  0  4  0 21 11  0
          11  0  1  0  2  0  1  2  0 15  1 20
```

Error rate = 0.5281385

The table below summarizes these error rates with those from (2) and (3):

	Training	Testing
LDA – part (2)	0.405303	0.5909091
QDA – part (3)	0.1231061	0.4458874
LDA – part (4)	0.3162879	0.5562771
QDA – part (4)	0.0113636	0.5281385

We can see that for all training and testing data sets, QDA has a lower error rate (higher accuracy) than LDA. It may because LDA assumes a common covariance matrix while QDA assumes that each class has its own covariance matrix. In this case, QDA might be more appropriate.

(5)

To simplify the analysis, we select only observations from classes {1,3,6,10} and conduct clustering analysis on these observations. We will try the hierarchical clustering method (single, complete, and average), *k*-means method and model-based clustering method for both the training and testing data sets.

Training:

```
> clust_train <- data.frame(train[train$y %in% c(1,3,6,10),])
> lel <- c(1,2,3,4); clust_train$y <- factor(clust_train$y,labels=lel)
> rownames(clust_train) <- seq(length=nrow(clust_train))
> sub <- clust_train[,-1]

> # Hierarchical:
> dist_sub <- dist(sub)
> # Single:
> singl_sub <-  hclust(dist_sub, method = 'single')
> singl_clusterCut <- cutree(singl_sub, 4)
> mean(singl_clusterCut!=clust_train$y)
[1] 0.78125
> # Complete:
> comp_sub <-  hclust(dist_sub, method = 'complete')
> comp_clusterCut <- cutree(comp_sub, 4)
> mean(comp_clusterCut!=clust_train$y)
[1] 0.6875
> # Average:
> avg_sub <-  hclust(dist_sub, method = 'average')
> avg_clusterCut <- cutree(avg_sub, 4)
> mean(avg_clusterCut!=clust_train$y)
[1] 0.875

> # K-Means:
> cl <- kmeans(sub, 4)
> table(cl$cluster,clust_train$y)

     1  2  3  4
  1 24 24  1  0
  2 18 24  0  6
  3  6  0  0 35
  4  0  0 47  7

> mean(cl$cluster != clust_train$y)
[1] 0.7135417
> # Model-based:
> mc <- Mclust(sub, 4)
fitting …
  |======================================================================| 100%
> table(mc$classification,clust_train$y)
```

```
     1  2  3  4
  1 24 12 30 48
  2 24  0  0  0
  3  0 12 18  0
  4  0 24  0  0
> mean(mc$classification != clust_train$y)
[1] 0.78125
```

Testing:

```
> clust_test <- data.frame(test[test$y %in% c(1,3,6,10),])
> lel <- c(1,2,3,4); clust_test$y <- factor(clust_test$y,labels=lel)
> rownames(clust_test) <- seq(length=nrow(clust_test))
> test_sub <- clust_test[,-1]

> # Hierarchical:
> dist_test.sub <- dist(test_sub)
> # Single
> ts_single.sub <- hclust(dist_test.sub, method = ‘single’)
> ts_single.clusterCut <- cutree(ts_single.sub, 4)
> mean(ts_single.clusterCut!=clust_test$y)
[1] 0.5
> # Complete:
> ts_complete.sub <-  hclust(dist_test.sub, method = ‘complete’)
> ts_complete.clusterCut <- cutree(ts_complete.sub, 4)
> mean(ts_complete.clusterCut!=clust_test$y)
[1] 0.6071429
> # Average:
> ts_avg.sub <-  hclust(dist_test.sub, method = ‘average’)
> ts_avg.clusterCut <- cutree(ts_avg.sub, 4)
> mean(ts_avg.clusterCut!=clust_test$y)
[1] 0.5

> # k-Means:
> ts_cl <- kmeans(test_sub, 4)
> table(ts_cl$cluster,clust_test$y)

     1  2  3  4
  1  0  0  0 26
  2 26  0  0  4
  3  0 25 40  0
  4 16 17  2 12
> mean(ts_cl$cluster != clust_test$y)
[1] 0.6904762

# Model-based:
> ts_mc <- Mclust(test_sub, 4)
fitting …
  |===========================================================================| 100%
> table(ts_mc$classification,clust_test$y)

     1  2  3  4
  1 24  0  0  6
  2  0 12 36 18
  3 18 18  6 18
  4  0 12  0  0
> mean(ts_mc$classification != clust_test$y)
[1] 0.75
```

The table below summarizes the performance results for each clustering method:

		Hierarchical		*k*-means	Model-based
	Single	Complete	Average		
Training	0.7813	0.6875	0.8750	0.7135	0.7813
Testing	0.5000	0.6071	0.5000	0.6905	0.7500

We can see that there really isn't any significant difference among the different clustering methods. An important thing to note is that in *k*-means clustering, the results change each time after running the algorithm because we start with a random choice of clusters. The results are reproducible, however, for the hierarchical and model-based clustering methods.

(6)

Final Notes:

Principal component analysis is a very good method to use when dealing with high-dimensional data, as it reduces both the risk of overfitting and computational complexity. It's also a well-established mathematical technique for reducing the dimensionality of data while keeping as much variation as possible.