Discriminant Analysis on Vowel Data

Hannah Pieper

3/5/21

This document can be run as an R notebook and is mostly self contained. The data used can be found at https://web.stanford.edu/~hastie/ElemStatLearn/data.html and consists observations associated to vowel sounds. There are 11 classes of vowels and 10 measured features of each observations. Thus document performs quadratic and linear discriminant analysis on this data.

Functions and Documentation

This document uses the following packages: dplyr, MASS, knitr, graphics.

The documentation of functions used in the latter portions of this document are included here.

QdiscriminantK()

Description: This function computes the quadratic discriminant function, $\delta_k(x)$ for class k evaluated at a particular observation x.

Inputs:
x - the feature vector for a particular observation
mu_k - the estimated mean for class k
sigma_k - the estimated covariance matrix for class k
pi_k - the estimated prior distribution for class k

mu_k and x must be row vectors for the dimensions of matrix multiplication to be compatible.

Output: the value of $\delta_k(x)$.

```
QdiscriminantK <- function(x, mu_k, sigma_k, pi_k){
  d_k <- -1/2*log(det(sigma_k))-1/2*t(x-mu_k)%*%ginv(sigma_k)%*%(x-mu_k) +log(pi_k)
  return(d_k)
}
```

get.class.data()

Description: This function computes the estimated class means, covariance matrices, pooled covariance matrix, and class priors.

Inputs: A data frame. In practice, we will input the data frame of training data. It is assumed that the rows are observations, the first column is the class and the remainder of the columns are the features.

Output: A list object. If there are K classes, let

$$\begin{aligned}
\text{priors} &= [\hat{\pi}_1, \ldots, \hat{\pi}_K] \\
\text{means} &= \begin{pmatrix} - & \hat{\mu}_1 & - \\ & \vdots & \\ - & \hat{\mu}_K & - \end{pmatrix} \\
\text{class.cov.list} &= \begin{pmatrix} \hat{\Sigma}_1 & \ldots & \hat{\Sigma}_K \end{pmatrix} \\
\text{pooled.sigma} &= \hat{\Sigma} = \frac{1}{N-K} \sum_{k=1}^{K} \sum_{g_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T
\end{aligned}$$

. Note that in practice $\hat{\Sigma}$ can be computed in the function pooled.cov().

```
get.class.data<-function(dataset){
  num.features<-ncol(dataset)-1
  N<-nrow(dataset)

  # We initialize the objects in which we store our class parameter data
  means<-matrix( nrow=11, ncol=num.features)
  priors<-rep(-1,11)
  class.cov.list<-list()

  # We compute the mean, prior and covariance estimates for each class and store them.
  for (i in 1:11){
    class_i<- dataset[dataset['y']==i,]

    Ni <- nrow(class_i)
    mu_i<-colSums(class_i[,-1])/Ni
    pi_i=Ni/N
    priors[i]=pi_i
    means[i,]=mu_i
    class.cov.list[[i]]<-cov(class_i[,-1])
  }

  # Compute the pooled covariance
  pooled.sigma<-pooled.cov(dataset)

  # Create a list object storing the priors, means, class covariance,
  # and pooled covariance
  data=list(priors, means, class.cov.list,pooled.sigma)
  return(data)
}
```

predict.class()

Description: This function computes the predicted class of a feature vector, x. This is done by computing $\text{argmax}_k \delta_k(x)$. Note that this function supports predictions with QDA and LDA.

Inputs:
x - the features of a particular observation as a row vector
type - enter 'Q' for the prediction via QDA and enter 'L' for the prediction via LDA
class.data - a list object of the form (priors, means, covariance matrices, pooled covariance matrix). In practice, this will be the list object returned from get.class.data().

Outputs: An integer corresponding to the predicted classification of the feature vector x.

```
predict.class <- function(x, type, class.data){
  # initialize vector to store values of the discriminant functions
  class.probs <- rep(-1,11)
  # compute the value of \delta_k(x) for each k
  for (i in 1:11){
    if (identical(type,'Q')){
      val<-QdiscriminantK(x,class.data[[2]][i,],class.data[[3]][[i]],class.data[[1]][i])}
    else if (identical(type,'L')){
      val<-LdiscriminantK(x,class.data[[2]][i,],class.data[[4]],class.data[[1]][i])
    }
    else{
      print('Incorrect analysis type. Choose "Q" for quadratic discriminant analysis
            and "L" for linear discriminant analysis')
    }
    # save the discriminant value
    class.probs[i]<-val
  }
  # choose the index for which \delta_k(x) is the largest
  predicted.class<-which.max(class.probs)
  return(predicted.class)
}
```

get.error()

Description: This function computes the overall misclassification rate and the misclassification rate by class via discriminant analysis for a data set.

Inputs:
data set - the data frame on which we wish to run either QDA or LDA. It is assumed that the rows are observations, the first column is the class and the remaining columns are the features.
type - enter 'Q' for the prediction via QDA and enter 'L' for the prediction via LDA
class.data - a list object of the form (priors, means, covariance matrices, pooled covariance matrix). In practice, this will be the list object returned from get.class.data().

Outputs: A list object of the form (class.error, overall.error). The object class.error is a vector containing the misclassification rate for class i in the ith entry. The object overall.error is a scalar that describes the proportion of misclassified data.

```
get.error <- function(dataset, type, class.data){
    N<-ncol(dataset)
    obs<-nrow(dataset)

    # initialize the vector of class errors and the total number of misclassified
    # observations
    class.error<-rep(-1,11)
    num.total.incorrect<-0

    # perform L/Q DA on each class
    for (j in 1:11){
      class.j <- as.matrix(dataset[dataset['y']==j,])
      features.j<-class.j[,2:N]
      truth_j<-class.j[,1]
      N.j<-nrow(class.j)

      # initialize vector of predictions for all the observations in class j
```

```
    predictions_j<-rep(-1,N.j)

    # Compute and store the predictions for all the observations in class j
    for (i in 1:N.j){
      predictions_j[i]<-as.vector(predict.class(features.j[i,],type, class.data))
    }
    # record the number of incorrectly classified observations in class j
    num.class.incorrect<-sum((truth_j-predictions_j)!=0)
    # record the proportion of incorrectly classified observations in class j
    class.error[j]<-num.class.incorrect/N.j
    # add the number of incorrectly classified observations in class j to the
    # total number of incorrect observations
    num.total.incorrect<-num.total.incorrect+num.class.incorrect
  }
  # compute proportion of incorrectly classified observations in entire data set
  overall.error<-num.total.incorrect/obs
  # create a list object containing the vector of class errors and the scalar
  # overall error rate
  error.list<-list(class.error, overall.error)
  return(error.list)
}
```

aug.data.set()

Description: This function creates a data set whose feature columns are augmented with quadratic features.

Inputs:
terms - a vector of the form c($i_1, ..., i_n$). The integers i_j correspond to the features that will be used to form the quadratic terms
data set - a data frame in which the rows consist of observations, the first column consists of the classifications and the remaining columns consist of the features.

Output: A augmented data frame. This data frame has the same number of rows as the input and the classification column and original feature columns are unchanged. An additional $n(n+1)/2$ feature columns are appended to the right of the original data frame and are of the form $i_j i_k$ with $j, k \in [1, \ldots, n]$. For a given observation, the quantity corresponding to the new features $i_j i_k$ is created by multiplying the corresponding observations for the original features i_j and i_k.

```
aug.data.set<-function(terms, dataset){
  # copy the original data set
  aug.data=dataset
  N=length(terms)

  for(i in 1:N){
    for (j in i:N){
      # create the new column x_ix_j. We add +1 in the indices because the first
      # entry of the data frame is the classification
      new<-dataset[,terms[i]+1]*dataset[,terms[j]+1]
      # append the new column to the augmented data frame
      aug.data[,ncol(aug.data)+1]<-new
      # append the name 'x.ix.j' to the list of column names
      colnames(aug.data)[ncol(aug.data)]<-paste0("x.",terms[i], "x.",terms[j])
    }
  }
  return(aug.data)
```

```
}
```

LdiscriminantK()

Description: This function computes the value of the linear discriminant function for class k at a specified value of x; $\delta_k(x)$.

Inputs:
x - the feature vector for a particular observation
mu_k - the estimated mean for class k
sigma - the estimated pooled covariance matrix (can be computed with pooled.cov())
pi_k - the estimated prior distribution for class k

mu_k and x must be row vectors for the dimensions of matrix multiplication to be compatible.

Output: the value of $\delta_k(x)$.

```
LdiscriminantK <- function(x, mu_k, sigma_k, pi_k){
  d_k <- t(x)%*%ginv(sigma_k)%*%mu_k-1/2*t(mu_k)%*%ginv(sigma_k)%*%(mu_k) +log(pi_k)
  return(d_k)
}
```

pooled.cov()

The built-in function cov() constructs the estimated covariance matrix on the columns of a given matrix. In particular, the estimated covariance of the ith class of a data set is given by

$$\hat{\Sigma}_i = \text{cov}(\text{dataset}[\text{dataset}['y'] = i]) = \frac{1}{N_i - 1} \sum_{g_j = i} (x_j - \hat{\mu}_i)(x_j - \hat{\mu}_i)^T;$$

where N_i is the number of observations in class i.

Therefore, if we want the pooled covariance matrix, $\hat{\Sigma}$, we can compute:

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{N - K} \sum_{k=1}^{K} \sum_{g_i = k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \\ &= \frac{1}{N - K} \sum_{k=1}^{K} (N_k - 1)\hat{\Sigma}_k. \end{aligned}$$

This computation is the basis for the function pooled.cov(). It should be noted that there are some packages that claim to compute the pooled covariance, but since the details of the code could not be verified, we use our own.

Description: This function returns the pooled covariance matrix given by Σ in the above discussion.

Inputs:
data set - a data frame with observations in each row and whose columns consist of observations. Note that the first column is assumed to be the classification of the observation. In practice, the input will be the training data.

Output:
If there are N features, this returns the $N \times N$ pooled covariance matrix. More specifically, this covariance matrix is pooling each of the classes.

```
pooled.cov<-function(dataset){
size=ncol(dataset)-1
by.hand.pooled.cov<-matrix(0,nrow=size, ncol=size)
```

```
for (i in 1:11) {
  class_i<- dataset[dataset['y']==i,]
  cov_i <-cov(class_i[,-1])
  N_i <- nrow(class_i)
  by.hand.pooled.cov<-by.hand.pooled.cov+(N_i - 1)*cov_i
}
  N<- nrow(dataset)
  return(by.hand.pooled.cov/(N-11))
}
```

Quadratic Discriminant Analysis

First, we recall some basic facts about QDA. In this case, we assume that each of the classes have different covariance matrices, Σ_i. If the class mean is denoted as $\hat{\mu}_i$ and the estimated covariance matrix is $\hat{\Sigma}_i$, then the ith quadratic discriminant function is given by

$$\delta_i(x) = -\frac{1}{2}\log|\hat{\Sigma}_i| - \frac{1}{2}(x-\hat{\mu}_k)^T\hat{\Sigma}_i^{-1}(x-\hat{\mu}_k) + \log\hat{\pi}_k.$$

If N_i is the number of points in class i in the training data, N is the total number of points in the training data, the above quantities are given by

$$\begin{aligned}\hat{\mu}_k &= \frac{1}{N_i}\sum_{g_k=i} x_k \\ \hat{\pi}_k &= \frac{N_i}{N} \\ \hat{\Sigma}_i &= \frac{1}{N_i-1}\sum_{g_k=i}(x_k-\hat{\mu}_i)(x_k-\mu_i)^T.\end{aligned}$$

Then, QDA classifies point x as belonging to class k if $k = \text{argmax}_k\delta_k(x)$

The general procedure is as follows:

1. Load training and testing data;
2. for each feature vector x, compute $\text{argmax}_k\delta_k(x)$ where δ_k is the quadratic discriminant function;
3. compare $\text{argmax}_k\delta_k(x)$ to the actual classification of x;
4. repeat the above two steps for every feature vector x to obtain an error rate.

We will perform the last four steps of this procedure on both the training and testing data.

First, we load our training data and our testing data.

```
trainvowels<-read.table("vowel_training_data.txt",
                        header = TRUE, sep=",", dec=".", row.names="row.names")

testvowels<-read.table("vowel_test_data.txt",
                       header=TRUE, sep=",", dec=".", row.names="row.names")

# Note that the file path will be specific to the working directory of the user
```

This data frame contains each of the observations in a row. The first entry is the classification of the observation and the following 10 entries are the observations for the 10 features. Visually:

```
head(trainvowels)
```

```
##   y    x.1   x.2    x.3   x.4    x.5   x.6    x.7    x.8    x.9   x.10
## 1 1 -3.639 0.418 -0.670 1.779 -0.168 1.627 -0.388  0.529 -0.874 -0.814
## 2 2 -3.327 0.496 -0.694 1.365 -0.265 1.933 -0.363  0.510 -0.621 -0.488
## 3 3 -2.120 0.894 -1.576 0.147 -0.707 1.559 -0.579  0.676 -0.809 -0.049
## 4 4 -2.287 1.809 -1.498 1.012 -1.053 1.060 -0.567  0.235 -0.091 -0.795
## 5 5 -2.598 1.938 -0.846 1.062 -1.633 0.764  0.394 -0.150  0.277 -0.396
## 6 6 -2.852 1.914 -0.755 0.825 -1.588 0.855  0.217 -0.246  0.238 -0.365
```

We obtain the estimated class means, priors, and covariance matrices from the training data. This is pre-computed in the interest of computation time.

```
data<-get.class.data(trainvowels)
```

Now, we can compute the predicted class using QDA for each feature vector in the training set. We compare the predicted classification to the actual classification to obtain an error rate.

```
error.list<-get.error(trainvowels,'Q', data)
```

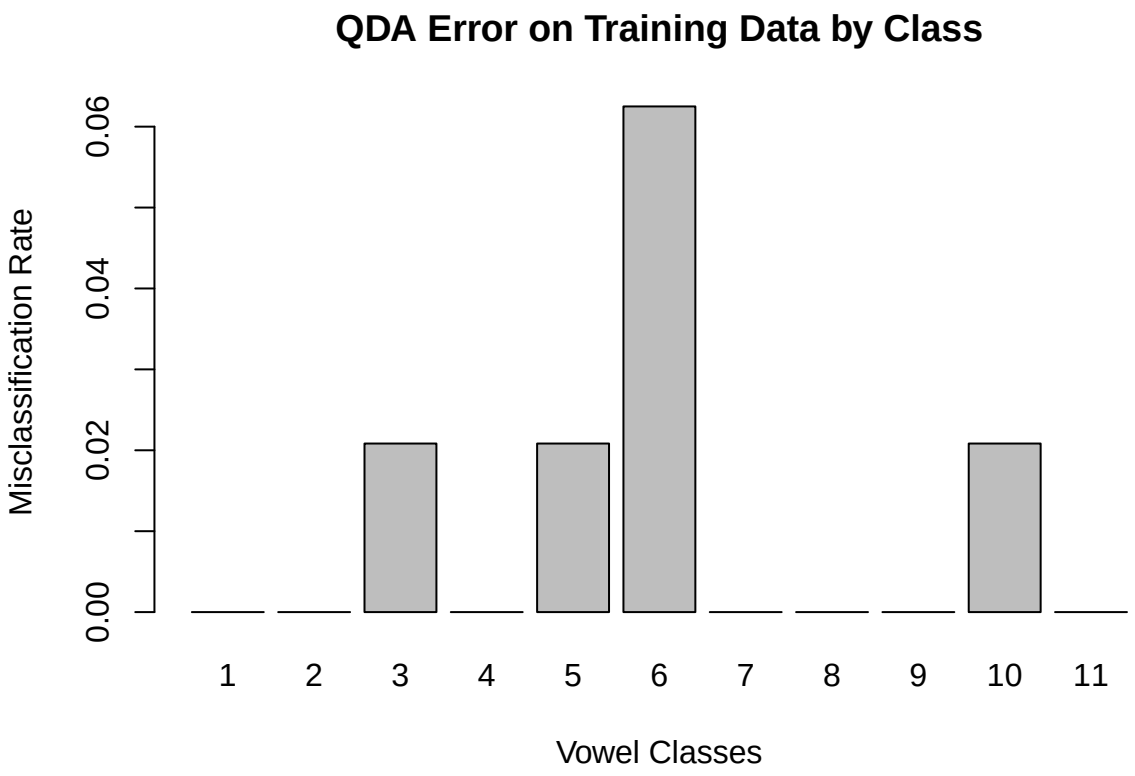
The total misclassification rate and misclassification rate within each class are as follow:

```
  print(paste("The total proportion of misclassified training data is:", error.list[2]))
```

```
## [1] "The total proportion of misclassified training data is: 0.0113636363636364"
```

```
  train.error<-data.frame(
    name=c(1:11),
    value=unlist(error.list[1]))

  barplot(height=train.error$value, names=train.error$name,
          xlab='Vowel Classes',
          ylab='Misclassification Rate',
          main='QDA Error on Training Data by Class')
```



Intuitively, this tells us that vowel classes $1, 2, 7, 8, 9$ could be the 'easiest' to correctly classify, while vowel classes $3, 5, 6, 10$ could be the 'hardest' to correctly classify. In this instance, the terms 'easiest' and 'hardest' are used non rigorously.

Now, we apply our QDA model to the testing data. We predict the class of each feature vector and compare it to the actual classification to obtain an error rate.

```
test.error.list<-get.error(testvowels,'Q', data)
```

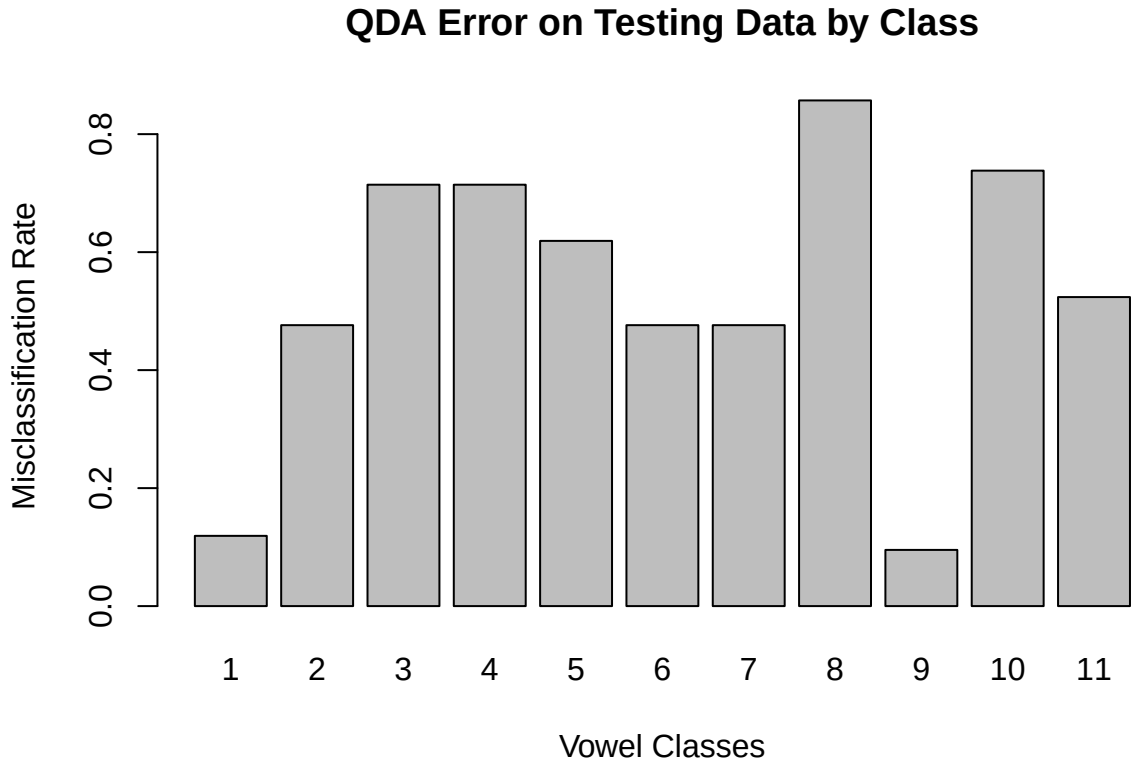
The total misclassification rate and misclassification rate within each class are as follow:

```
print(paste("The total proportion of misclassified test data is:",test.error.list[2]))
```

```
## [1] "The total proportion of misclassified test data is: 0.528138528138528"
```

```
test.error<-data.frame(
  name=c(1:11),
  value=unlist(test.error.list[1]))

barplot(height=test.error$value, names=test.error$name,
        xlab='Vowel Classes',
        ylab='Misclassification Rate',
        main='QDA Error on Testing Data by Class')
```



In summary, the overall misclassification rate for the training data was .01, while the overall misclassification rate for the testing data was .53.

Linear Discriminant Analysis on Augmented Feature Space

Next, we perform linear discriminant analysis on an augmented data set. More specifically, we will add extra features in the form of $x_i x_j$, $i, j \in [1, 11]$, where x_i and x_j are features already included in the data set and then apply LDA. The hope is this methodology will result in a lower overall misclassification rate on the testing data than QDA.

Recall that linear discriminant function for class k is formulated as

$$\delta_k(x) = x^T \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k.$$

The terms $\hat{\mu}_k$ and $\hat{\pi}_k$ are computed in the same way as in QDA, but now we used a pooled covariance estimate over all the classes:

$$\hat{\Sigma} = \frac{1}{N - K} \sum_{k=1}^{K} \sum_{g_i = k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T.$$

The general procedure is as follows:

1. Create a data frame in the augmented feature space consisting of linear and quadratic terms;
2. for each feature vector x, compute $\text{argmax}_k \delta_k(x)$ where δ_k is the linear discriminant function;
3. compare $\text{argmax}_k \delta_k(x)$ to the actual classification of x;
4. repeat the above two steps for every feature vector x to obtain an error rate.

We will perform this procedure on both the training and testing data.

First, we will create an augmented data set with quadratic terms. Our strategy will be to specify the indices of the original feature vectors that will generate the quadratic terms. For example, if we wanted to create

quadratic terms from the first feature x_1 and the second feature x_2, we add columns that contain the features x_1x_1, x_1x_2 and x_2x_2 to the right of the original data set.

```
  # Here we specify the vector of indices from which we create our quadratic terms.
  # This can be experimented with.
  ############################
  new.terms=c(1,3)
  ############################

  # Now we form the augmented training and testing data sets.
  aug.train.data=aug.data.set(new.terms,trainvowels)
  aug.test.data=aug.data.set(new.terms, testvowels)
```

Our augmented data sets have a form similar to that of the ordinary data sets with the quadratic features appended as columns on the right. In this augmented data set, each observation forms a row, the first entry is the classification of the observation, the first 10 entries are the original observed features and the remaining terms correspond to the created quadratic features. If we add k terms in the above code block, we will have $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ new columns. The first several observations in the new feature space are as follow:

```
head(aug.train.data)
```

```
##   y    x.1   x.2    x.3   x.4    x.5   x.6    x.7    x.8    x.9   x.10
## 1 1 -3.639 0.418 -0.670 1.779 -0.168 1.627 -0.388  0.529 -0.874 -0.814
## 2 2 -3.327 0.496 -0.694 1.365 -0.265 1.933 -0.363  0.510 -0.621 -0.488
## 3 3 -2.120 0.894 -1.576 0.147 -0.707 1.559 -0.579  0.676 -0.809 -0.049
## 4 4 -2.287 1.809 -1.498 1.012 -1.053 1.060 -0.567  0.235 -0.091 -0.795
## 5 5 -2.598 1.938 -0.846 1.062 -1.633 0.764  0.394 -0.150  0.277 -0.396
## 6 6 -2.852 1.914 -0.755 0.825 -1.588 0.855  0.217 -0.246  0.238 -0.365
##      x.1x.1   x.1x.3   x.3x.3
## 1 13.242321 2.438130 0.448900
## 2 11.068929 2.308938 0.481636
## 3  4.494400 3.341120 2.483776
## 4  5.230369 3.425926 2.244004
## 5  6.749604 2.197908 0.715716
## 6  8.133904 2.153260 0.570025
```

Again, we pre-compute the class means, priors and pooled covariance matrix for the augmented training data so that our computations are more efficient.

```
aug.data<-get.class.data(aug.train.data)
```

Now we perform analysis with LDA. Note that we are using the pooled covariance and the linear discriminant function. See 'Functions and Documentation' for more information. We compute the proportion of training data points that are misclassified as well as the the proportion of misclassified training data withing each class.

```
aug.train.error.list<-get.error(aug.train.data,'L', aug.data)
```

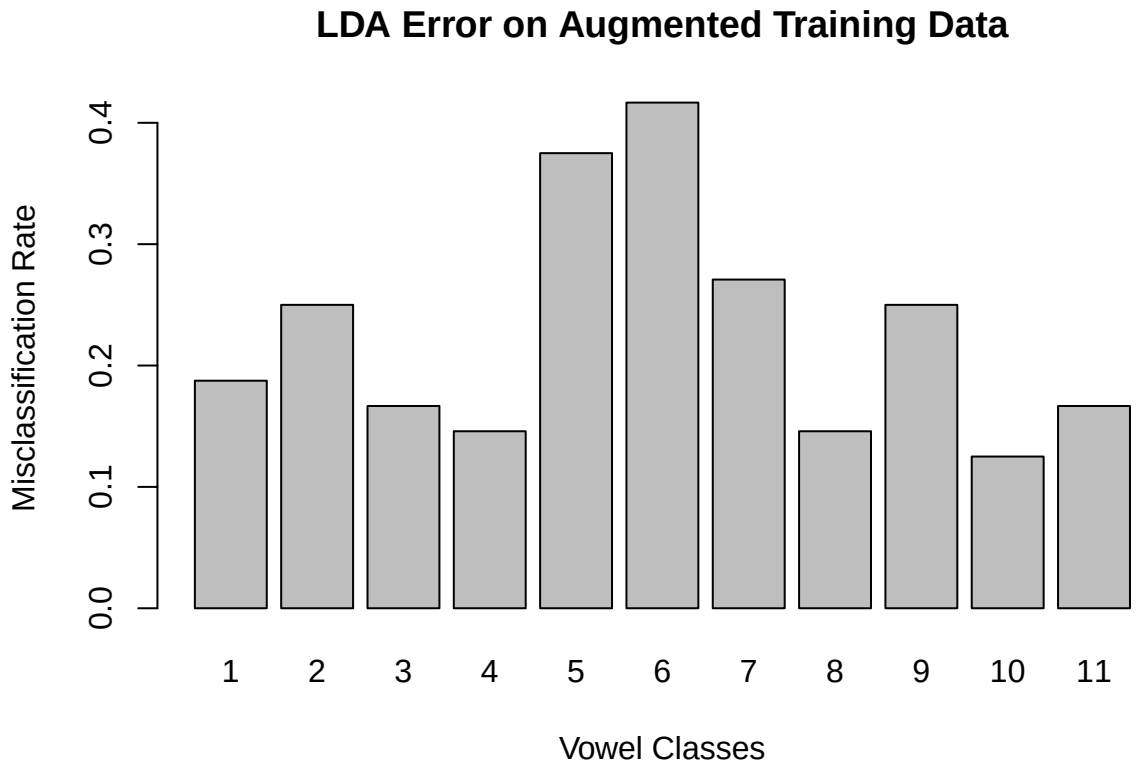
The error rates for the augmented training data are as follows:

```
  print(paste('The total proportion of misclassified training data is:',aug.train.error.list[2]))
```

```
## [1] "The total proportion of misclassified training data is: 0.227272727272727"
```

```
  train.error<-data.frame(
    name=c(1:11),
    value=unlist(aug.train.error.list[1]))
```

```
  barplot(height=train.error$value, names=train.error$name,
          xlab='Vowel Classes',
          ylab='Misclassification Rate',
          main='LDA Error on Augmented Training Data')
```



Now, we apply our LDA function to the augmented testing data. We first compute the total misclassification rate, then we break up the misclassification rate by class.

```
aug.test.error.list<-get.error(aug.test.data, 'L', aug.data)
```

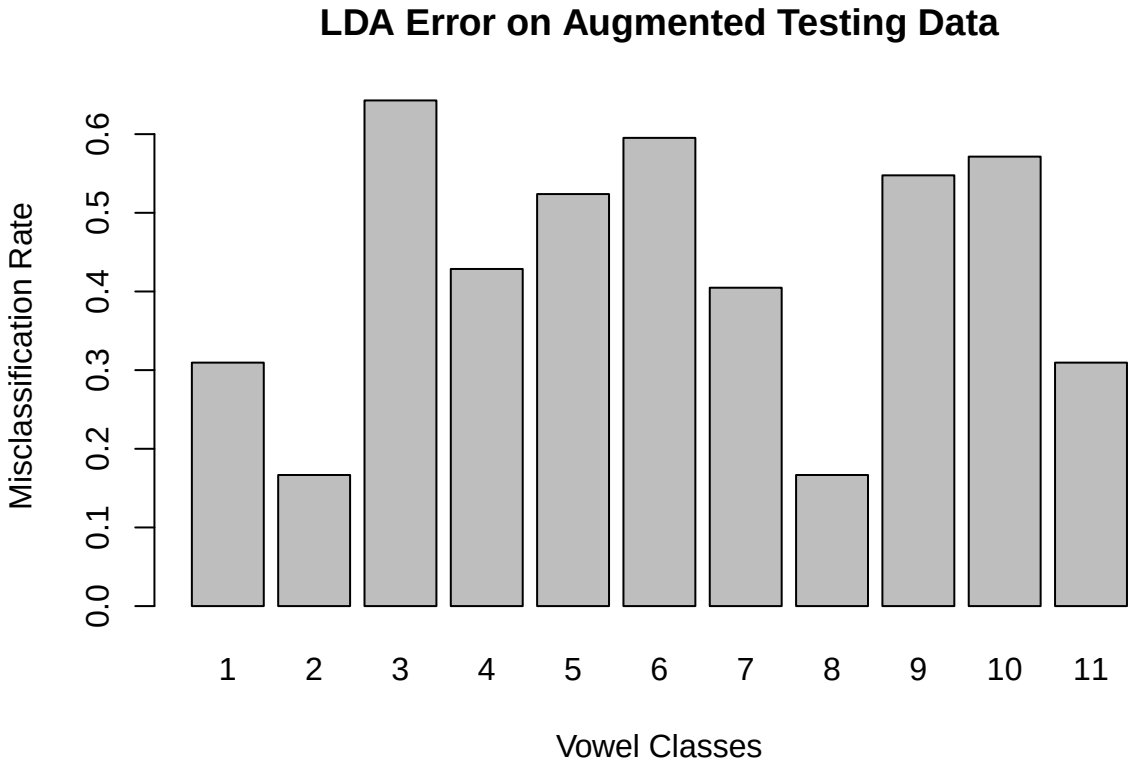
The error rates are as follows:

```
  print(paste("The total proportion of misclassified test data is:",aug.test.error.list[2]))
```

```
## [1] "The total proportion of misclassified test data is: 0.424242424242424"
```

```
  test.error<-data.frame(
    name=c(1:11),
    value=unlist(aug.test.error.list[1]))

  barplot(height=test.error$value, names=test.error$name,
          xlab='Vowel Classes',
          ylab='Misclassification Rate',
          main='LDA Error on Augmented Testing Data')
```



The following table contains the results for several choices of added quadratic terms.

Features Added as Quad. Terms	LDA Training Error	LDA Testing Error
–	.32	.56
9	.30	.55
1	.25	.45
3	.28	.49
1, 9	.24	.44
1, 3	.23	.42
1, 4	.22	.46
8, 10	.31	.54
1, 3, 9	.20	.43
5, 7, 8	.24	.50
1, 2, 6, 9, 10	.08	.46

-	QDA Training Error	QDA Testing Error
–	.01	.53

The first row displays the results of LDA on the vowel data in the original feature space. Both the training and testing error is higher than that of those of QDA computed in the previous section. Furthermore, performing LDA on a feature space that contains quadratic terms consistently gives a better testing error than the testing error corresponding to LDA on the original feature space.

It seems that adding quadratic variables in the first and third feature improve our correct classification rate on the testing data the most. As we add more terms the training error goes down significantly but the testing error increases slightly; indicating that we are overfitting the model to the training data. Additionally, while the training data error rate is higher for LDA than it is for QDA, there are some choices of quadratic features that result in a lower LDA testing error.