Linear and Quadratic Discriminant Analysis

Benjamin Draves

Introduction & Background

We implement Quadratic Discriminant Analysis (QDA) and Linear Discriminant Analysis (LDA) for classifying 11 vowels based on $p = 10$ features $\{X_i\}_{i=1}^{p}$. We compare the classifications from QDA based on the $p-$dimensional space $\tilde{\mathcal{X}} = \{X_i : 1 \leq i \leq p\}$ and compare these classifications to LDA on an expanded, $(2p + \binom{p}{2})-$dimensional feature space $\mathcal{X} = \{X_i, X_i^2, X_i X_j : 1 \leq, i, j \leq p\}$. We begin by implementing methods that calculate the quadratic discriminant functions $\tilde{\delta}_k(\cdot)$ and linear discriminant functions $\delta_k(\cdot)$

$$\tilde{\delta}_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k$$
$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k$$

As stated in equation 4.10 (ESL pg. 109), classifying based on LDA is equivalent to the decision rule $G(x) = \arg\max_k \delta_k(x)$. To see why this decision rule also holds for QDA, consider the QDA log-ratio,

$$\begin{aligned}
\log\frac{\mathbb{P}(G=k|X=x)}{\mathbb{P}(G=\ell|X=x)} &= \log\frac{f_k(x)}{f_\ell(x)} + \log\frac{\pi_k}{\pi_\ell} \\
&= \log\frac{(2\pi)^{-p/2}\det(\Sigma_k)^{-1/2}\exp\left\{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right\}}{(2\pi)^{-p/2}\det(\Sigma_\ell)^{-1/2}\exp\left\{-\frac{1}{2}(x-\mu_\ell)^T\Sigma_\ell^{-1}(x-\mu_\ell)\right\}} + \log\frac{\pi_k}{\pi_\ell} \\
&= \log\frac{\det(\Sigma_k)^{-1/2}}{\det(\Sigma_\ell)^{-1/2}} + \log\frac{\exp\left\{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right\}}{\exp\left\{-\frac{1}{2}(x-\mu_\ell)^T\Sigma_\ell^{-1}(x-\mu_\ell)\right\}} + \log\frac{\pi_k}{\pi_\ell} \\
&= -\frac{1}{2}\log\det(\Sigma_k) - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k \\
&+ \frac{1}{2}\log\det(\Sigma_\ell) + \frac{1}{2}(x-\mu_\ell)^T\Sigma_\ell^{-1}(x-\mu_\ell) - \log\pi_k \\
&= \tilde{\delta}_k(x) - \tilde{\delta}_\ell(x)
\end{aligned}$$

Therefore, if we fix ℓ, we would classify based on the rule

$$G(x) = \begin{cases} \arg\max_k \tilde{\delta}_k(x) & \exists k \neq \ell : \tilde{\delta}_k(x) \geq \tilde{\delta}_\ell(x) \\ \ell & \nexists k \neq \ell : \tilde{\delta}_k(x) \geq \tilde{\delta}_\ell(x) \end{cases}$$

Notice, however that $\nexists k \neq \ell : \tilde{\delta}_k(x) \geq \tilde{\delta}_\ell(x)$ is an equivalent condition to $\tilde{\delta}_\ell(x) \geq \tilde{\delta}_k(x)$ for all k. Moreover, the first condition requires we classify to the class corresponding to the largest $\tilde{\delta}_k(x)$ where $\tilde{\delta}_k(x) \geq \tilde{\delta}_\ell(x)$. Therefore, in either case, we see we classify to the class corresponding to the largest discriminant function. Thus we see we can rewrite both classification rules as follows

$$\hat{G}_{LDA} = \arg\max_k \delta_k(x) \qquad \hat{G}_{QDA} = \arg\max_k \tilde{\delta}_k(x)$$

Therefore our implementation will be focused on the efficient calculation of the discriminant functions $(\delta_k, \tilde{\delta}_k)_{k=1}^{K}$ based on the training data. From here, we will evaluate these functions with the test data to produce predictions. We beging with implementing QDA followed by LDA.

Implementation of Quadratic Discriminant Analysis (QDA)

```
#read in data
train <- read.csv("~/Documents/Work/github/Courses/MA 751/DA1/Data/vowel.train.txt")
test <- read.csv("~/Documents/Work/github/Courses/MA 751/DA1/Data/vowel.test.txt")

#clean data
train <- as.matrix(train[,-1])
test <- as.matrix(test[,-1])

#-----------------------------------
#
#         QDA Implemenation
#
#-----------------------------------

#function: est - return mean/covariance estimates
#arguments:
#       X - data matrix corresponding to class k
est <- function(X){
  mu <- colMeans(X)
  Sigma <- cov(scale(X, scale = FALSE))
  return(list(mu = mu, Sigma = Sigma))
}

#function: deltaQDA - return quadratic discriminant function
#arguments:
#       x - test point
#       mu - estimated group mean
#       Sigma - estimated group covariance
#       lp - estimated log - prior prob
deltaQDA <- function(x, mu, Sigma, lp){

    #numerically stable matrix inversion
    C <- chol(Sigma)
    y <- backsolve(C, matrix(x - mu, ncol = 1), trans = TRUE)
    b <- backsolve(C,y)

    #return discriminant function
    -.5 * log(det(Sigma)) - .5 * crossprod(x - mu, b) + lp
}

#function: QDA - return classification
#arguments:
#       x - test point
QDA <-local({

  #local variable declarations to avoid redundant calculations
  #calculate log prior class probability
  lpi <- log(as.numeric(prop.table(table(train[,1]))))

  #calculate class means & covariances
  K <- length(unique(train[,1]))
```

```
  mus <- list(K)
  Sigmas <- list(K)

  for(k in 1:K){
    #subset dataframe to get class k & get estimates
    df <- train[,-1][which(train[,1] == k), ]
    tmp <- est(df)

    #store estimates
    mus[[k]] <- tmp$mu
    Sigmas[[k]] <- tmp$Sigma

  }
  #return maximum discriminant function
  function(x) which.max(sapply(1:K, function(y) deltaQDA(x, mus[[y]], Sigmas[[y]], lpi[y])))

})
```

Implementation of Linear Discriminant Analysis (LDA)

```
#-----------------------------------
#
#         LDA Implemenation
#
#-----------------------------------
#Set up new training/testing data in X~
train.new <- matrix(NA, nrow = nrow(train) , ncol  = 1 + 10 + 10 + choose(10, 2))
train.new[,1:11] <- as.matrix(train)

test.new <- matrix(NA, nrow = nrow(test) , ncol  = 10 + 10 + choose(10, 2))
test.new[,1:10] <- as.matrix(test[,-1])

#Expand covariate space to incldue (Xi*Xj, Xi^2)
count <- 1
for(i in 1:10){
  for(j in i:10){
    train.new[,11 + count] <- train[,i + 1] * train[,j + 1]
    test.new[,10 + count] <- test[,i + 1] * test[,j + 1]
    count <- count + 1
  }
}

#function: deltaLDA - return linear discriminant function
#arguments:
#       x - test point
#       mu - estimated group mean
#       lp - estimate log prior probs

deltaLDA <- local({
    #calculate common Sigma with bias correction
    N <- nrow(train.new); K <- length(unique(train[,1]))
    Sigmas <- list(K)
```

```
    #calculate group covariances
    for(k in 1:K){
      df <- train.new[,-1][which(train.new[,1] == k), ]
      Sigmas[[k]] <- (N-1)*cov(scale(df, scale = FALSE))
    }
    #calculate full covariance
    Sigma <- 1/(N - K) * Reduce("+", Sigmas)


    function(x, mu, lp){
      #numerically stable matrix inversion
      C <- chol(Sigma)
      y <- backsolve(C, matrix(mu, ncol = 1), trans = TRUE)
      b <- backsolve(C,y)

      #return discriminant function
      crossprod(x, b) - .5 * crossprod(mu, b) + lp
    }
})

#function: LDA - return classification
#arguments:
#       x - test point
LDA <-local({
  #define local varibles to avoide redundants caculations
  #calculate prior class probabilities
  lpi <- log(as.numeric(prop.table(table(train.new[,1]))))

  #calculate class means
  K <- length(unique(train.new[,1]))
  mus <- list(K)
  for(k in 1:K){
    #subset dataframe & get estimates
    df <- train.new[,-1][which(train.new[,1] == k), ]

    #store estimates
    mus[[k]] <- colMeans(df)
  }
  #return classifications
  function(x) which.max(sapply(1:K, function(y) deltaLDA(x, mus[[y]], lpi[y])))

})
```

Application to Test Set

As we discussed above, these methodologies are based on maximizing the log-odds that $\mathbf{x}$ belongs to class k or class ℓ. From this, the decision rule can be reduced to the following.

$$\hat{G}_{QDA}(x) = \arg\max_k \tilde{\delta}(x) \qquad \hat{G}_{LDA}(x) = \arg\max_k \delta(x)$$

As do not know the functions $\tilde{\delta}$ and δ, we estimate them using our training data. Recall that we expand the feature space to estimate $\delta(\cdot)$, we estimate $\tilde{\delta}_k(\cdot)$ from $\tilde{\tau} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and $\delta_k(\cdot)$ from $\tau = \{(\mathbf{x}_i, \mathbf{x}_i^2, \mathbf{x}_i\mathbf{x}_j, y_i)\}_{i\neq j}^n$. Once we have estimated these discriminant functions $\hat{\tilde{\delta}}(\cdot)$, $\hat{\delta}(\cdot)$, we then based our predictions on the rule

$$\hat{G}_{QDA}(x) = \arg\max_k \hat{\tilde{\delta}}(x) \qquad \hat{G}_{LDA}(x) = \arg\max_k \hat{\delta}(x)$$

Therefore, we simply need to specify how these functions are estimated. Following ESL, we calculate $\hat{\tilde{\delta}}_k(\cdot)$ and $\hat{\delta}_k(\cdot)$ as follows

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

where $\tilde{\mathbf{x}} \in \tilde{X}$ and $x \in X$ and we define

$$\hat{\mu}_k = \frac{1}{n_k}\sum_{\{i:y_i=k\}} \mathbf{x}_i$$
$$\hat{\tilde{\mu}}_k = \frac{1}{n_k}\sum_{\{i:y_i=k\}} \tilde{\mathbf{x}}_i$$
$$\hat{\tilde{\Sigma}}_k = \frac{1}{n_k - 1}\sum_{\{i:y_i=k\}} (\tilde{\mathbf{x}}_i - \hat{\tilde{\mu}}_k)(\tilde{\mathbf{x}}_i - \hat{\tilde{\mu}}_k)^T$$
$$\hat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^K \sum_{\{i:y_i=k\}} (\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T$$
$$\hat{\pi}_k = \frac{n_k}{n}$$

In essence, training this model is done by simply estimating these values. From here, we use them as pluggin estimators to obtain the functions $\hat{\tilde{\delta}}_k(\cdot)$ and $\hat{\delta}_k(\cdot)$. This allows for QDA and LDA to be run efficiently even when the number of datapoints grows significantly.

Having now trained our models (QDA/LDA) on seperate features spaces $\tilde{\mathcal{X}} \subseteq \mathcal{X}$ we look to apply these models to our test data. The models were not trained on these test points which will allow us to compare each methods performance on this dataset. We simply evaluate each function $\hat{\tilde{\delta}}(\tilde{x})$, $\hat{\delta}(x)$ for each $\tilde{x}$ and constructed x in the test set. With these, we can derive an estimate of the class and compare these estimates to the true class labels to compare the methods. We do exactly this in the following blocks of code.

```
#format test data
test.X <- as.matrix(test[,-1]); colnames(test.X) = rep("", ncol(test.X))

#get predictions
yhat <- apply(test.X, 1, QDA)

#get get misclassification rate
MR <- sum(yhat != test[,1])/nrow(test)

#get predictions
```

```r
yhat.new <- apply(test.new, 1, LDA)

#get get misclassification rateth4
MR.new <- sum(yhat.new != test[,1])/nrow(test)

df0 <- data.frame(Method = c("LDA", "QDA"),
                  Correct = c(1 - round(MR.new,2),1 - round(MR,2)),
                  Incorrect = c(round(MR.new,2), round(MR, 2)))
knitr::kable(df0)
```

Method	Correct	Incorrect
LDA	0.56	0.44
QDA	0.47	0.53

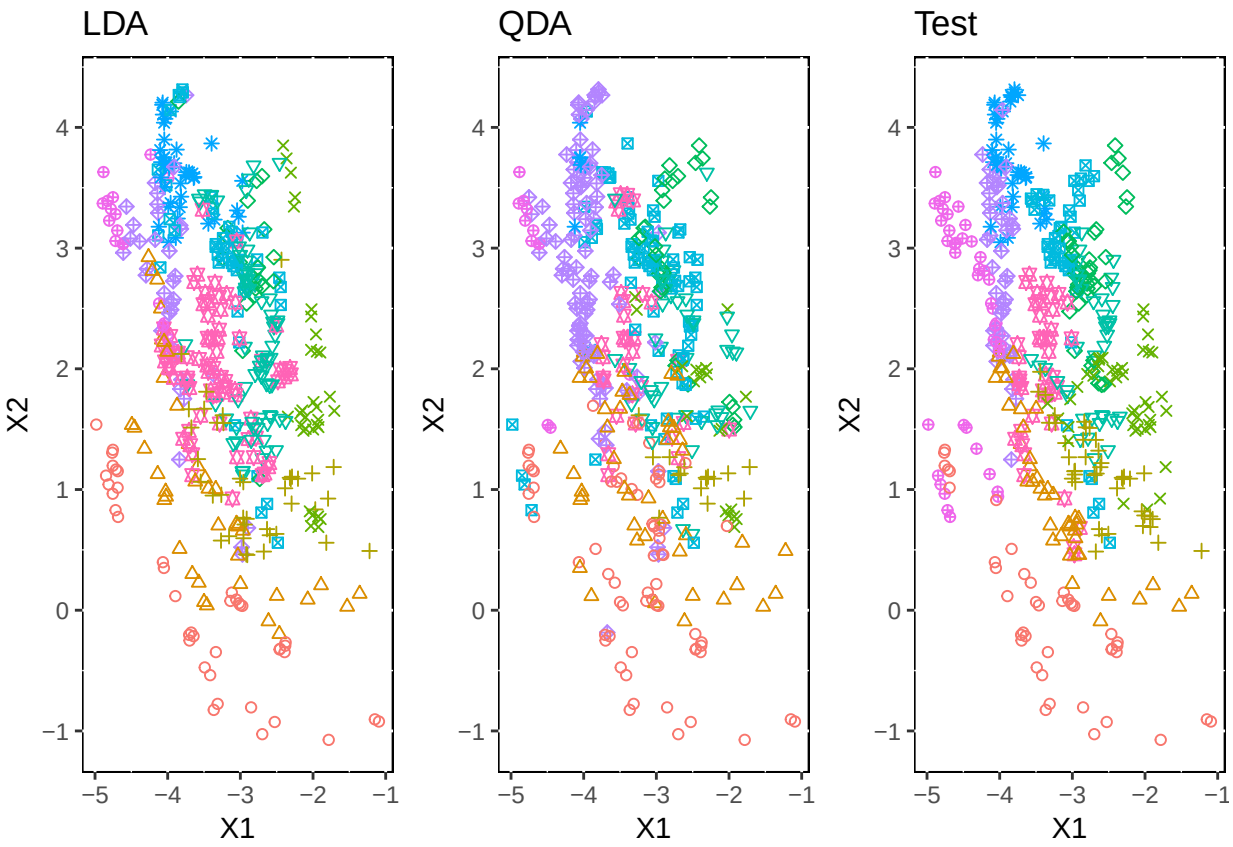
Here we clearly see that LDA out preforms QDA. By expanding the feature space, we are able to reduce the missclassification rate by 7%. In general, however, it appears that these methods do not fit the data well. While a marked improvement from the uniform classifier in which we would expect a 10/11 misslcassification rate, there is still quite a bit of room for improvement.

Comparison of Methods

In this section, we look to visualize the predictions made by both LDA and QDA. We begin by coloring the predicted values and plotting them on the $X_1 - X_2$ plane. By projecting into this two-dimensional space, we see that both QDA and LDA generally are picking up on the seperation of the classes in the test data in this two-dimensional feature space. One can imagine in higher dimensions that this seperation is much more clear.

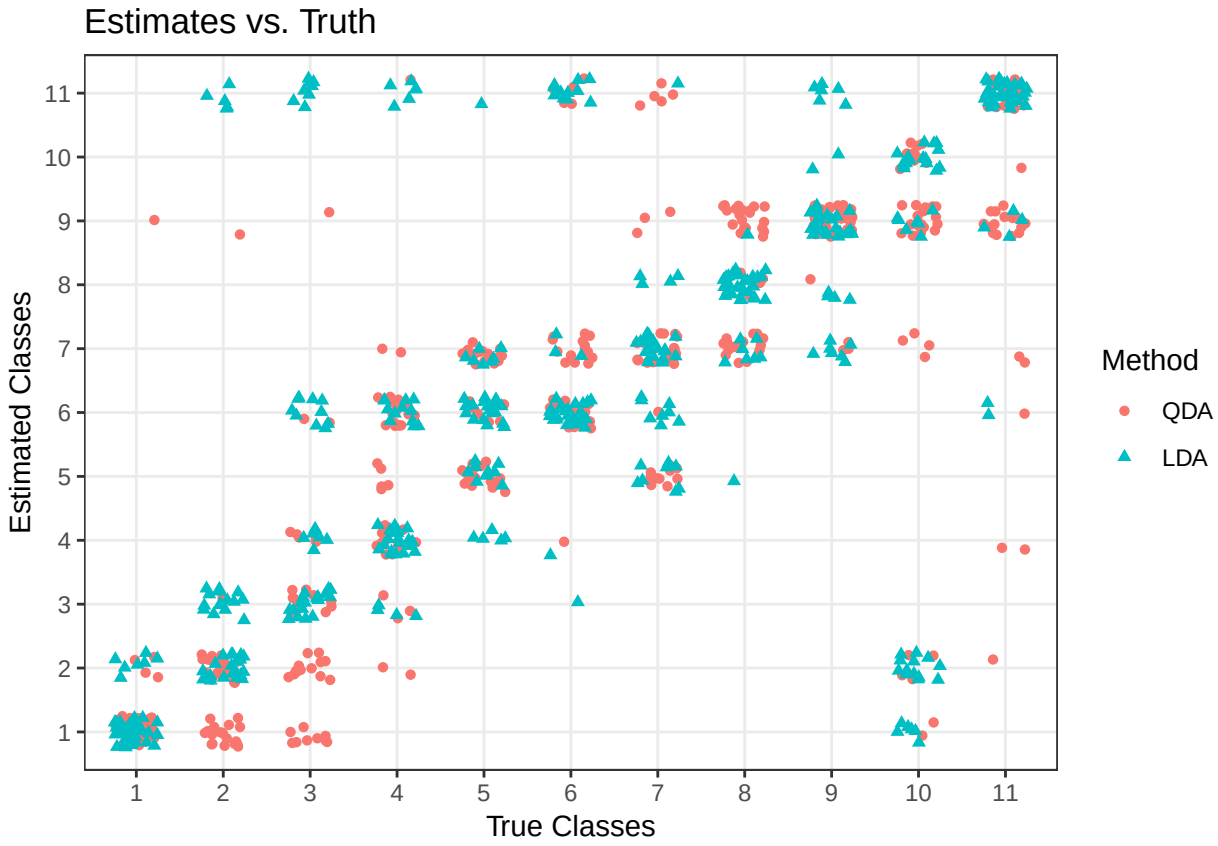
```r
#make figures
library(ggplot2); library(gridExtra)

df1 <- data.frame(X1 = test.X[,1], X2 = test.X[,2], class = as.factor(yhat))
df2 <- data.frame(X1 = test.X[,1], X2 = test.X[,2], class = as.factor(test[,1]))

p1 <- ggplot(df1, aes(X1, X2, shape = class, col = class))+
   scale_shape_manual(values=1:nlevels(df1$class)) +
  geom_point(alpha = 1)+
  labs(title = "QDA")+
  theme(legend.position="none",
        panel.background = element_rect(fill = 'white', colour = 'black'))

p2 <- ggplot(df2, aes(X1, X2, col = class, shape = class))+
  labs(title = "Test")+
  scale_shape_manual(values=1:nlevels(df2$class)) +
  geom_point(alpha = 1)+
  theme(legend.position="none",
        panel.background = element_rect(fill = 'white', colour = 'black'))

df3 <- data.frame(X1 = test.new[,1], X2 = test.new[,2], class = as.factor(yhat.new))

p3 <- ggplot(df3, aes(X1, X2, col = class, shape = class))+
  geom_point(alpha = 1)+
  scale_shape_manual(values=1:nlevels(df3$class)) +
  labs(title = "LDA")+
```

```
  theme(legend.position="none",
        panel.background = element_rect(fill = 'white', colour = 'black'))

grid.arrange(p3, p1, p2, ncol=3)
```



Lastly, we look to compare the predictions of each method against the test points. We plot the true classes on the x-axis and the estimated classes on the y-axis. We color the points by which method they correspond to. In addition we jitter the points to see the density of each method. A perfect estimator in this plot would only have points along the line $y = x$. Here we see in general that these estimators follow that trend. It appears that LDA out preforms QDA in this setting. QDA appears to not classify points in class 2, 3, 5, and 8 well and LDA appears to not classify points well in classes 2, 4, and 5.

```
library(reshape2)
df4 <- data.frame(QDA = as.factor(yhat),
                  LDA = as.factor(yhat.new),
                  test = as.factor(test[,1]))
df4 <- melt(df4, id.vars = c("test"))
df4$value <- factor(as.factor(df4$value), levels = as.character(1:11))
colnames(df4)[2] <- c("Method")

ggplot(df4, aes(x = test, y = value, col = Method, shape = Method))+
  geom_point(position = position_jitter(w = .25, h = .25),
             alpha = 1)+
  labs(title = "Estimates vs. Truth", x = "True Classes", y = "Estimated Classes")+
  theme_bw()
```


Estimates vs. Truth
Estimated Classes
True Classes
1
2
3
4
5
6
7
8
9
10
11
Method
QDA
LDA