STAT 6340 (Statistical and Machine Learning), Spring 2019

Mini Project 3 (Solution)

April 11, 2019

1. Consider the business school admission data available on eLearning. The admission officer of a business school has used an ”index” of undergraduate grade point average (GPA, X_1) and graduate management aptitude test (GMAT, X_2) scores to help decide which applicants should be admitted to the school’s graduate programs. This index is used categorize each applicant into one of three groups - admit (group 1), do not admit (group 2), and borderline (group 3). We will take the last five observations in each category as test data and the remaining observations as training data.

   (a) *Perform an exploratory analysis of data by examining appropriate plots and comment on how helpful these predictors may be in predicting response.*

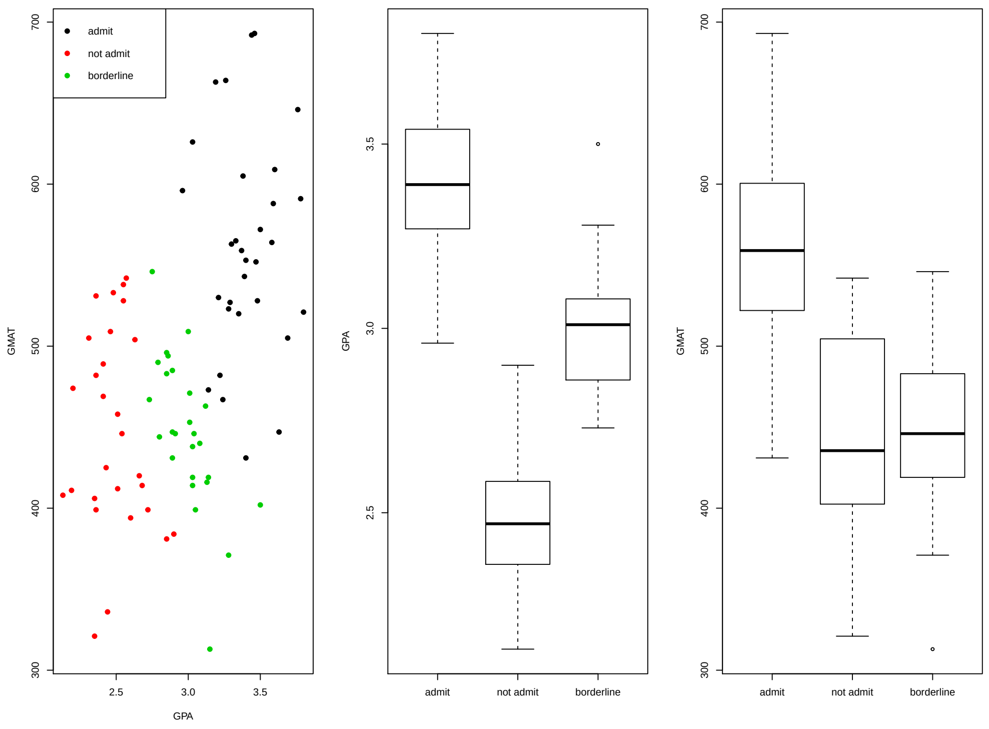

Figure 1: Scatter/box plots

From Figure 1 we see that GMAT and GPA are positively correlated. Moreover, GPA is a useful predictor in predicting the response, but it is difficult to distinguish the class levels "do not admit" and "borderline" based on the GMAT scores.

(b) *Perform an LDA and provide an equation for the decision boundary. Superimpose the decision boundary on an appropriate display of the data. Does the decision boundary seem sensible? In addition, compute the confusion matrix and overall misclassification rate based on both training and test data. What do you observe?*

Decision boundary equations are:

```
- ( 117.1452 ) + ( 32.6288 ) * GPA + ( 0.0445 ) * GMAT = 0 [ 1-2 ]
- ( 64.1129 ) + ( 14.3474 ) * GPA + ( 0.0370 ) * GMAT = 0 [ 1-3 ]
 ( 53.0323 ) + ( -18.2813 ) * GPA + ( -0.0075 ) * GMAT = 0 [ 2-3 ]
```

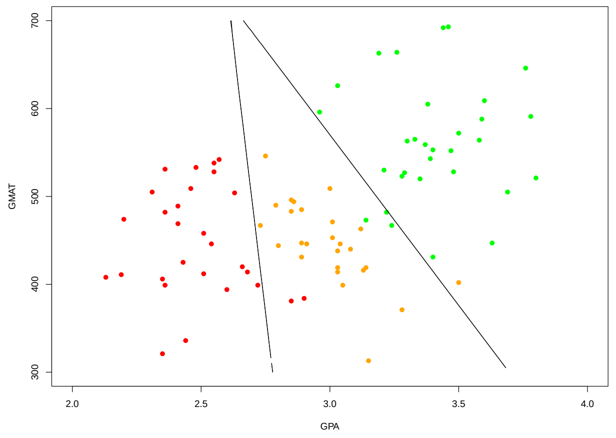

Figure 2: LDA decision boundary

lda.pred.train\Y.train	admit	not admit	borderline
admit	24	0	1
not admit	0	23	0
borderline	2	0	20

Table 1: Confusion matrix for training set using LDA

lda.pred.test\Y.test	admit	not admit	borderline
admit	4	0	0
not admit	0	3	0
borderline	1	2	5

Table 2: Confusion matrix for test set using LDA

Overall misclassification rates on the training and test sets are 0.04286 and 0.2 respectively. The linear decision boundaries look sensible separating most of the data points correctly. However, the misclassification rate on the test set is bit high.

(c) *Repeat (b) using QDA.*

Decision boundary equations are:

$$-\frac{1}{2}X^T\begin{bmatrix} \text{-22.53} & 0.03 \\ 0.03 & \text{-0.00} \end{bmatrix}X + \begin{bmatrix} \text{-14.09} \\ 0.08 \end{bmatrix}^T X = 54.84728$$

$$-\frac{1}{2}X^T\begin{bmatrix} 0.36 & \text{-0.14} \\ \text{-0.14} & \text{-0.00} \end{bmatrix}X + \begin{bmatrix} \text{-91.16} \\ \text{-0.60} \end{bmatrix}^T X = -301.2936$$

$$-\frac{1}{2}X^T\begin{bmatrix} \text{-22.18} & \text{-0.11} \\ \text{-0.11} & \text{-0.00} \end{bmatrix}X + \begin{bmatrix} \text{-105.25} \\ \text{-0.52} \end{bmatrix}^T X = -246.4463,$$

where $X = [GPA, GMAT]^T$.

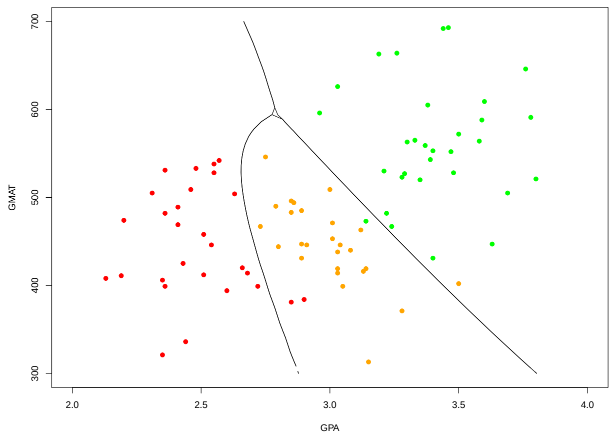

Figure 3: QDA decision boundary

qda.pred.train\Y.train	admit	not admit	borderline
admit	25	0	1
not admit	0	23	0
borderline	1	0	20

Table 3: Confusion matrix for training set using QDA

qda.pred.test\Y.train	admit	not admit	borderline
admit	5	0	0
not admit	0	3	0
borderline	0	2	5

Table 4: Confusion matrix for test set using QDA

Overall misclassification rates on the training and test sets are 0.02857 and 0.13333 respectively. The quadratic decision boundaries look a bit more sensible than those of LDA separating the data points more accurately.

(d) *Repeat (b), with the exception of providing the decision boundary equation, using KNN with K chosen optimally using the test data.*

We know that GPA and GMAT scores are on 0-4 and 200-800 scales respectively. Because GMAT scores are on a larger scale, they will have a stronger impact on the KNN results. Moreover, in our case the GMAT scores are not informative enough to distinguish between the 'do not admit' and 'borderline' decisions. Thus, giving more weight to the GMAT scores may lead to a poor KNN performance. To avoid this situation, we standardize the data to put our predictors on an equal footing.

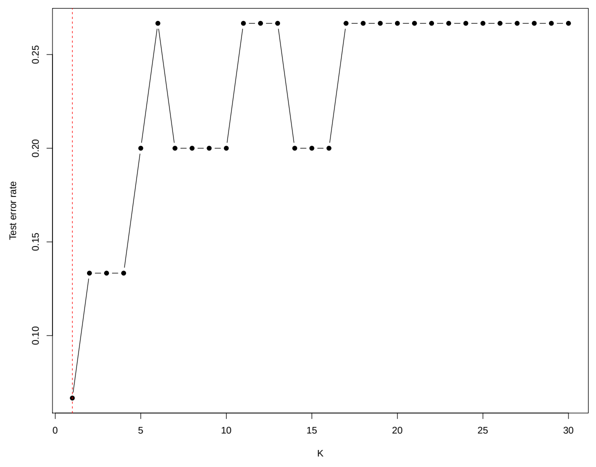

Figure 4: KNN misclassification rate on test set for different values of K

Figure 4 shows that the smallest misclassification error rate on the test set is achieved at $K = 1$.

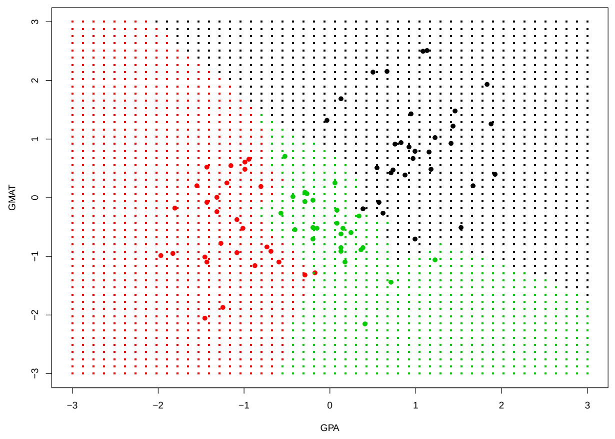

Figure 5: KNN misclassification rate on test set for different values of K

knn.pred.train\Y.train	admit	not admit	borderline
admit	26	0	0
not admit	0	23	0
borderline	0	0	21

Table 5: Confusion matrix for training set using KNN

knn.pred.test\Y.test	admit	not admit	borderline
admit	5	0	0
not admit	0	4	0
borderline	0	1	5

Table 6: Confusion matrix for test set using KNN

Overall misclassification rates on the training and test sets are 0 and 0.06666 respectively. The KNN decision boundaries, being more flexible, separate the data points better than those of LDA and QDA.

(e) *Compare the results in (b)-(d). Which classifier would you recommend? Justify your conclusions.*

	Test error rate
LDA	0.2000
QDA	0.1333
KNN	0.0666

Table 7: Performance of LDA, QDA and KNN on test set

Based on the performance on the test set, KNN is recommended.

2. Annual financial data are collected for bankrupt firms approximately two years prior to their bankruptcy and financially sound firms at about the same time. The data on four variables, $X1 = CF/TD$ =(cash flow)/(total debt), $X2 = NI/TA$ =(net income)/(total assets), $X3 = CA/CL$=(current assets)/(current liabilities), and $X4 = CA/NS$ =(current assets)/(net sales) are given on eLearning. This is a binary classification problem. Take bankrupt firm as "+" response (indicated as 0 in the data) and nonbankrupt firm as "-" response (indicated as 1 in the data). Use all the data as training data.

(a) *Perform an exploratory analysis of data by examining appropriate plots and comment on how helpful these predictors may be in predicting response.*

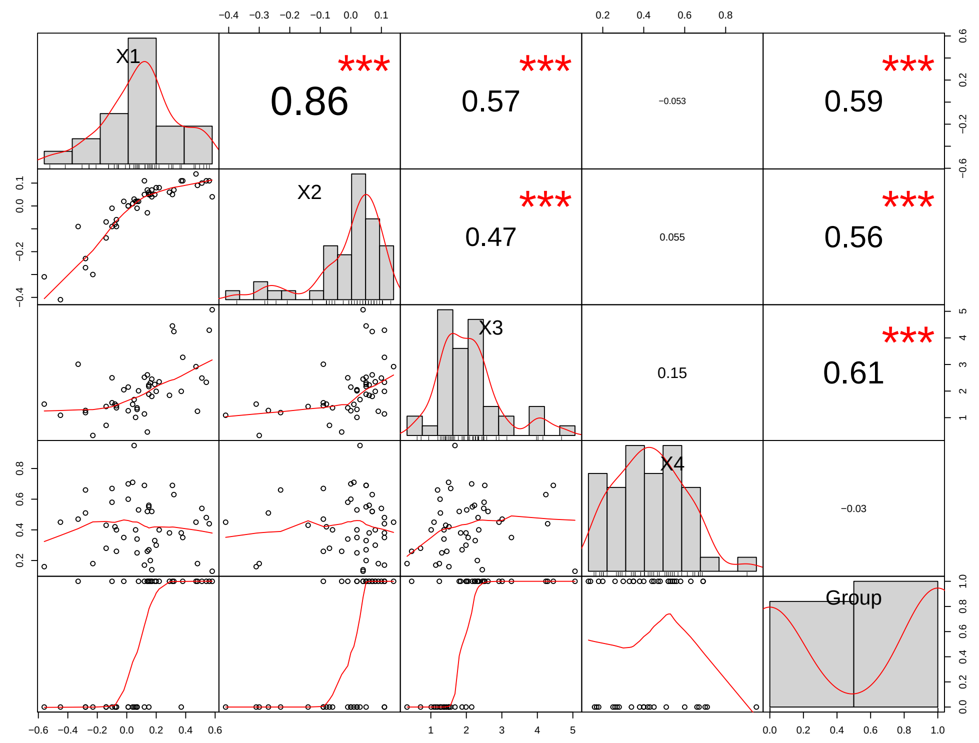

Figure 6: Scatterplots

We see from the scatterplots in Figure 7 that all the predictors except X_4 appear to be helpful in predicting the response (bankruptcy). However, notice that X_1 and X_2 are highly correlated. So it might be sufficient to include only one of them and drop the other during model building.

(b) *Build an appropriate logistic regression model for these data. Interpret the estimated regression coefficients in the final proposed model.*

Based on our observations in part (a), we fit two models - the full model and the reduced model containing only X_1 and X_3 - and perform the anova chi-squared test to check if the predictors X_2 and X_4 can be jointly dropped.

```
Analysis of Deviance Table

Model 1: Group ~ X1 + X2 + X3 + X4
Model 2: Group ~ X1 + X3
Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1         41      27.443
2         43      28.636 -2   -1.1924   0.5509
```

A high p-value of 0.5509 indicates that the predictors X_2 and X_4 can be excluded from our final model.

```
Call:
glm(formula = Group ~ X1 + X3, family = binomial, data = data.bankruptcy)
```

```
Deviance Residuals:
Min        1Q    Median        3Q       Max
-2.26853  -0.47678   0.00942   0.48365   2.70538

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept)   -5.940      1.985  -2.992  0.00277 **
X1             6.556      2.905   2.257  0.02402 *
X3             3.019      1.002   3.013  0.00259 **
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 63.421  on 45  degrees of freedom
Residual deviance: 28.636  on 43  degrees of freedom
AIC: 34.636

Number of Fisher Scoring iterations: 6
```

The above summary shows that both X_1 and X_3 are significant predictors and neither can dropped at the expense of keeping the other. An estimated coefficient of X_1 is 6.556 which means that a one-unit increase in X_1 is associated with an increase in the log odds of nonbankruptcy by 6.556 units assuming the value X_3 is kept fixed. An estimated coefficient of X_3 is 3.019 which means that a one-unit increase in X_3 is associated with an increase in the log odds of nonbankruptcy by 3.019 units assuming the value X_1 is kept fixed.

3. Consider the bankruptcy data of the previous problem.

   (a) *Use the logistic regression model built in the previous problem to provide an equation for the decision boundary for the classification problem. In addition, compute the confusion matrix, sensitivity, specificity, and overall misclassification rate, and plot the ROC curve. What do you observe?*

   Following the same setup as in problem 2, we take the entire dataset as the training set. An equation for the decision boundary is:

   ```
   -5.94 + ( 6.56 ) * X1 + ( 3.02 ) * X3 = 0
   ```

lr.pred\Y	0	1
0	18	1
1	3	24

Table 8: Confusion matrix using logistic regression model built in problem 2

Sensitivity	0.960
Specificity	0.857
Misclassification rate	0.087

Table 9: Performance summary of logistic regression model built in problem 2

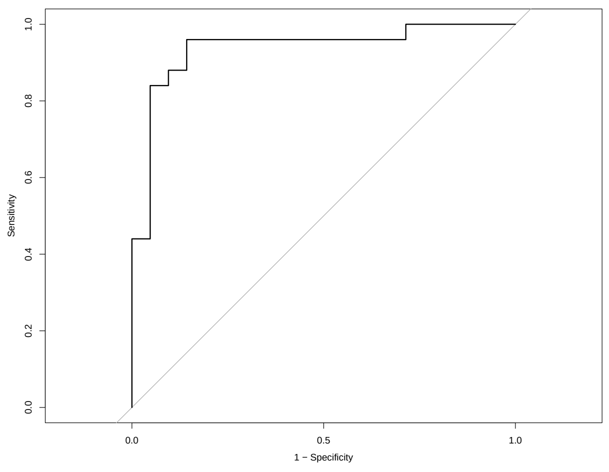

Figure 7: ROC curve (AUC=0.9371)

The logistic model built in problem 2 performs reasonably well on the training data in terms of sensitivity, specificity, misclassification rate and AUC.

(b) *Repeat (a) with all predictors (not just those appearing the final proposed model from the previous problem). Comment on whether there is any benefit in performing variable selection over using all predictors.*

An equation for the decision boundary is:

```
-5.32 + ( 7.14 ) * X1 + ( -3.70 ) * X2 + ( 3.41 ) * X3 + ( -2.97 ) * X4 = 0
```

lr.pred\Y	0	1
0	18	1
1	3	24

Table 10: Confusion matrix using logistic regression model with all predictors

Sensitivity	0.960
Specificity	0.857
Misclassification rate	0.087

Table 11: Performance summary of logistic regression model with all predictors

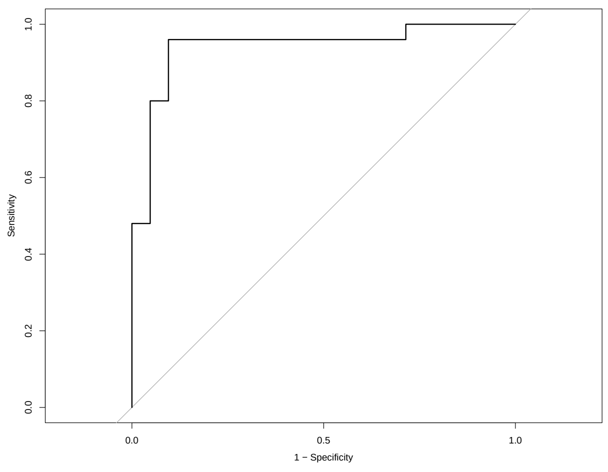

Figure 8: ROC curve (AUC=0.941)

Comparing the results we see that the performances of logistic regression models built in problem 2 and the one containing all the predictors are the same (except for a small difference in the ROC curves). Thus, there is no benefit in performing variable selection over using all the predictors.

(c) *Repeat (a) with all predictors using LDA.*

Equation for the decision boundary is:

```
- ( -2.20 ) + ( -1.26 ) * X1 + ( -8.37 ) * X2 + ( -1.69 ) * X3 + ( 2.24 ) * X4 = 0
```

lda.pred\Y	0	1
0	18	1
1	3	24

Table 12: Confusion matrix using LDA

Sensitivity	0.960
Specificity	0.857
Misclassification rate	0.087

Table 13: Performance summary of LDA

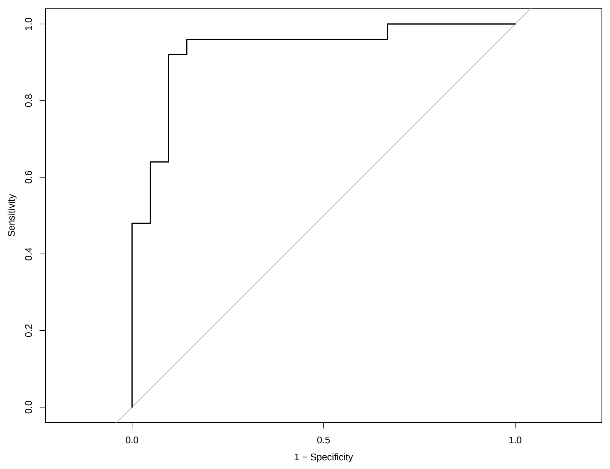

Figure 9: ROC curve (AUC=0.9333)

For LDA we observe the same performance as with logistic regression (except for an insignificant difference in AUC).

(d) *Repeat (a) with all predictors using QDA.*

An equation for the decision boundary is:

$$-\frac{1}{2}X^T \begin{bmatrix} 95.52 & 45.93 & 5.39 & -28.00 \\ 45.93 & -1037.46 & -23.29 & 100.26 \\ 5.39 & -23.29 & 7.53 & -2.84 \\ -28.00 & 100.26 & -2.84 & -23.25 \end{bmatrix} X + \begin{bmatrix} 4.85 \\ -30.14 \\ 8.70 \\ -11.60 \end{bmatrix}^T X = 2.8978,$$

where $X = [X_1, X_2, X_3, X_4]^T$.

qda.pred\Y	0	1
0	19	1
1	2	24

Table 14: Confusion matrix using QDA

Sensitivity	0.960
Specificity	0.905
Misclassification rate	0.065

Table 15: Performance summary of QDA

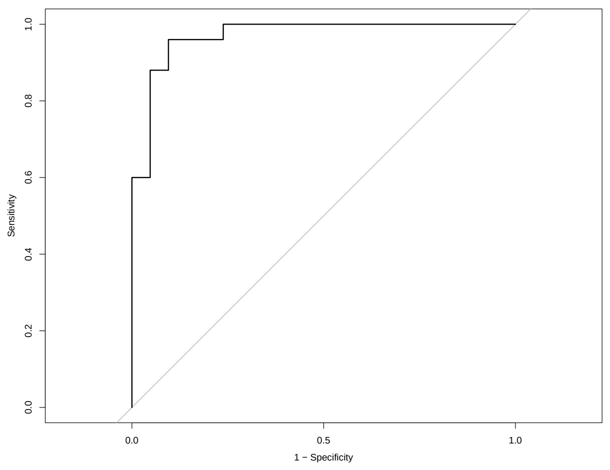

Figure 10: ROC curve (AUC=0.9695)

(e) *Compare the results in (a)-(d). Which classifier would you recommend? Justify your conclusions.*

Based on the sensitivity, specificity, overall misclassification rate and AUC results in (a)-(d) computed for the training data, QDA delivers the best performance and therefore would be recommended.

```
#################################### R CODE ########################################

library(MASS) # for lda and qda
library(class) # for knn
library(pROC) # for ROC curve
library(PerformanceAnalytics) # for scatterplots

# Problem 1
# Read data and define X and Y
data.adm=read.csv('admission.csv')[,-(4:7)]
X=data.adm[,1:2]
Y=as.factor(data.adm[,3]); levels(Y)=c('admit','not admit','borderline')

# (a) - Exploratory analysis
par(mfrow=c(1,3))
plot(X[,1],X[,2],xlab='GPA',ylab='GMAT',col=Y,pch=19)
legend('topleft',legend = c('admit','not admit','borderline'),col = 1:3,pch=19)
boxplot(X[,1]~Y,ylab='GPA'); boxplot(X[,2]~Y,ylab='GMAT')
par(mfrow=c(1,1))

# (b) - Perform LDA
# Define indeices of training set
n=cumsum(table(Y))
train=1:nrow(X) %in% c(1:(n[1]-5),(n[1]+1):(n[2]-5),(n[2]+1):(n[3]-5))
```

```
# fit LDA
lda.fit=lda(Y~.,data=X,subset = train)

# Define function to find equation of lda decision boundary
s6340.lda <- function(y, X) {
# y = training data response vector (a factor) X = training
# data predictor matrix
N <- length(y) # no of observations
K <- nlevels(y) # no of classes
p <- ncol(X) # no of predictors
n <- as.numeric(table(y)) # class frequencies
names(n) <- levels(y)
pi <- n/N # class proportions
# mean vector
mu <- matrix(unlist(by(X, y, colMeans)), byrow = T, ncol = p)
rownames(mu) <- levels(y)
colnames(mu) <- colnames(X)
# pooled covariance matrix
S <- by(X, y, cov)
Sigma <- Reduce("+", lapply(1:K, FUN = function(k) {
(n[k] - 1) * S[[k]]
}))/(N - K)
# its inverse
Sigma.inv <- solve(Sigma)
# delta functions
delta <- t(sapply(1:K, FUN = function(k) {
c(-(1/2) * drop(t(mu[k, ]) %*% Sigma.inv %*% mu[k, ]) +
log(pi[k]), t(mu[k, ]) %*% Sigma.inv)
}))
rownames(delta) <- levels(y)
colnames(delta) <- c("(Intercept)", colnames(X))
# pairwise difference of delta functions
idx.pair <- combn(K, 2)
delta.diff <- t(apply(idx.pair, MAR = 2, FUN = function(pair) {
delta[pair[1], ] - delta[pair[2], ]
}))
rownames(delta.diff) <- apply(idx.pair, MAR = 2, FUN = function(pair) {
paste0(levels(y)[pair[1]], "-", levels(y)[pair[2]])
})
# multiply intecept difference by 1 to get the cutoff c
delta.diff[, 1] <- -delta.diff[, 1]
colnames(delta.diff)[1] <- "Cutoff"
# result
result <- format(delta.diff,digits = 3)
return(result)
}

# decision booundary equations
```

```
lda.db.coef <- s6340.lda(as.factor(data.adm[train,3]),data.adm[train,1:2])
for (i in 1:nrow(lda.db.coef))
{
db=paste('- (',lda.db.coef[i,1],')')
for (j in 2:ncol(lda.db.coef))
{
db=paste(db,'+ (',lda.db.coef[i,j],') *',colnames(lda.db.coef)[j])
}
db=paste(db,'=',0,'[',rownames(lda.db.coef)[i],']')
print(db)
}

# draw the decision boundary (using the contour approach)
# set up the grid
n.grid <- 50
x1.grid <- seq(f = 2.0, t = 4.0, l = n.grid)
x2.grid <- seq(f = 300, t = 700, l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- c("GPA", "GMAT")

# get the posterior probabilities on the grid
lda.pred.grid <- predict(lda.fit, grid)

# get the probability functions whose zero contours provide
# the decision boundaries (need only two functions for three classes)

p1 <- lda.pred.grid$posterior[, 1] -
pmax(lda.pred.grid$posterior[, 2], lda.pred.grid$posterior[, 3])
p1 <- matrix(p1, nrow = n.grid, ncol = n.grid, byrow = F)

p2 <- lda.pred.grid$posterior[, 2] -
pmax(lda.pred.grid$posterior[, 1], lda.pred.grid$posterior[, 3])
p2 <- matrix(p2, nrow = n.grid, ncol = n.grid, byrow = F)

p3 <- lda.pred.grid$posterior[, 3] -
pmax(lda.pred.grid$posterior[, 1], lda.pred.grid$posterior[, 2])
p3 <- matrix(p3, nrow = n.grid, ncol = n.grid, byrow = F)

# plot data and superimpose decision boundary
plot(NULL, xlim = range(x1.grid), ylim = range(x2.grid),
xlab = "GPA", ylab = "GMAT")
points(subset(data.adm, Group == "1")[, 1:2], col = "green",pch=19)
points(subset(data.adm, Group == "2")[, 1:2], col = "red",pch=19)
points(subset(data.adm, Group == "3")[, 1:2], col = "orange",pch=19)

contour(x1.grid, x2.grid, p1, levels = 0, labels = "", xlab = "", ylab = "",
main = "", add = T)
contour(x1.grid, x2.grid, p2, levels = 0, labels = "", xlab = "", ylab = "",
main = "", add = T)
contour(x1.grid, x2.grid, p3, levels = 0, labels = "", xlab = "", ylab = "",
main = "", add = T)

# Compute predicted classes for training and test sets
lda.pred.train=predict(lda.fit)$class
lda.pred.test=predict(lda.fit,X[!train,])$class

# Compute confusion matrices for training and test sets
lda.confmat.train=table(lda.pred.train,Y[train]); print(lda.confmat.train)
lda.confmat.test=table(lda.pred.test,Y[!train]); print(lda.confmat.test)

# Compute misclassification error rates for training and test sets
lda.error.train=mean(lda.pred.train!=Y[train]); print(lda.error.train)
lda.error.test=mean(lda.pred.test!=Y[!train]); print(lda.error.test)

# (c)
# Fit QDA
qda.fit=qda(Y~.,data=X,subset = train)

# Define function to find equation of qda decision boundary
s6340.qda <- function(y, X) {
# y = training data response vector (a factor) X = training
# data predictor matrix
N <- length(y) # no of observations
K <- nlevels(y) # no of classes
p <- ncol(X) # no of predictors
n <- as.numeric(table(y)) # class frequencies
names(n) <- levels(y)
pi <- n/N # class proportions
# mean vector
mu <- matrix(unlist(by(X, y, colMeans)), byrow = T, ncol = p)
rownames(mu) <- levels(y)
colnames(mu) <- colnames(X)
# pooled covariance matrix
Sigma <- by(X, y, cov)
#Sigma <- Reduce("+", lapply(1:K, FUN = function(k) {
#  (n[k] - 1) * S[[k]]}))/(N - K)
#its inverse
Sigma.inv <- by(X, y, function(x) solve(cov(x)))
# delta functions
if (K >= 2) {
term1 <- 0.5 * (log(det(Sigma[[1]])) - log(det(Sigma[[2]])))
term2 <- 0.5 * (t(mu[1, ]) %*% Sigma.inv[[1]] %*% mu[1,
] - t(mu[2, ]) %*% Sigma.inv[[2]] %*% mu[2, ])
term3 <- log(pi[2]/pi[1])
cutoff <- term1 + term2 + term3
rownames(cutoff) <- paste(levels(y)[1], "-", levels(y)[2],
sep = "")
Sigma.inv.diff <- Sigma.inv[[1]] - Sigma.inv[[2]]
mu.Sigma.inv.diff <- t(mu[1, ]) %*% Sigma.inv[[1]] -
t(mu[2, ]) %*% Sigma.inv[[2]]
# result
result <- list(N = N, n = n, pi = pi, mu = mu, Sigma = Sigma,
Sigma.inv = Sigma.inv, quad.coef.matrix = Sigma.inv.diff,
linear.coef.vector = mu.Sigma.inv.diff, cutoff = cutoff)
} else {
result <- list(N = N, n = n, pi = pi, mu = mu, Sigma = Sigma,
Sigma.inv = Sigma.inv)
}
return(result)
}

# decision boundaries
y=as.factor(data.adm[train,3])
qda.db.coef=s6340.qda(y.train,data.adm[train,1:2])
qda.db.coef$quad.coef.matrix; qda.db.coef$linear.coef.vector; qda.db.coef$cutoff
# relabel classes
y[data.adm[train,3]==1]=3
y[data.adm[train,3]==2]=1
y[data.adm[train,3]==3]=2
qda.db.coef=s6340.qda(y,data.adm[train,1:2])
qda.db.coef$quad.coef.matrix; qda.db.coef$linear.coef.vector; qda.db.coef$cutoff
# relabel classes
y[data.adm[train,3]==1]=1
y[data.adm[train,3]==2]=3
y[data.adm[train,3]==3]=2
qda.db.coef=s6340.qda(y,data.adm[train,1:2])
qda.db.coef$quad.coef.matrix; qda.db.coef$linear.coef.vector; qda.db.coef$cutoff

# get the posterior probabilities on the grid
qda.pred.grid <- predict(qda.fit, grid)

# get the probability functions whose zero contours provide
# the decision boundaries (need only two functions for three classes)
p1 <- qda.pred.grid$posterior[, 1] -
pmax(qda.pred.grid$posterior[, 2], qda.pred.grid$posterior[, 3])
p1 <- matrix(p1, nrow = n.grid, ncol = n.grid, byrow = F)

p2 <- qda.pred.grid$posterior[, 2] -
pmax(qda.pred.grid$posterior[, 1], qda.pred.grid$posterior[, 3])
p2 <- matrix(p2, nrow = n.grid, ncol = n.grid, byrow = F)

p3 <- qda.pred.grid$posterior[, 3] -
pmax(qda.pred.grid$posterior[, 1], qda.pred.grid$posterior[, 2])
p3 <- matrix(p3, nrow = n.grid, ncol = n.grid, byrow = F)
```

```
# plot data and superimpose decision boundary
plot(NULL, xlim = range(x1.grid), ylim = range(x2.grid),
xlab = "GPA", ylab = "GMAT")
points(subset(data.adm, Group == "1")[, 1:2], col = "green",pch=19)
points(subset(data.adm, Group == "2")[, 1:2], col = "red",pch=19)
points(subset(data.adm, Group == "3")[, 1:2], col = "orange",pch=19)

contour(x1.grid, x2.grid, p1, levels = 0, labels = "", xlab = "", ylab = "",
main = "", add = T)
contour(x1.grid, x2.grid, p2, levels = 0, labels = "", xlab = "", ylab = "",
main = "", add = T)
contour(x1.grid, x2.grid, p3, levels = 0, labels = "", xlab = "", ylab = "",
main = "", add = T)

# Compute predicted classes for training and test sets
qda.pred.train=predict(qda.fit)$class
qda.pred.test=predict(qda.fit,X[!train,])$class

# Compute confusion matrices for training and test sets
qda.confmat.train=table(qda.pred.train,Y[train]); print(qda.confmat.train)
qda.confmat.test=table(qda.pred.test,Y[!train]); print(qda.confmat.test)

# Compute misclassification error rates for training and test sets
qda.error.train=mean(qda.pred.train!=Y[train]); print(qda.error.train)
qda.error.test=mean(qda.pred.test!=Y[!train]); print(qda.error.test)

# (d)
# Standardize data
X=as.data.frame(scale(X))

# Find optimal value of K
ks=1:30
knn.error=c()

for (i in ks)
{
set.seed(1)
knn.fit=knn(X[train,],X[!train,],Y[train],k=i,prob = T)
knn.error[i]=mean(knn.fit!=Y[!train])
}
plot(knn.error,type = 'b',xlab='K',ylab = 'Test error rate',pch=19)
k.opt=which.min(knn.error)
abline(v=k.opt,lty=2,col='red')

# draw the decision boundary (using the contour approach)
# set up the grid
n.grid <- 50
```

```
x1.grid <- seq(f = -3, t = 3, l = n.grid)
x2.grid <- x1.grid
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- c("GPA", "GMAT")

# get predicted classes on the grid
knn.pred.grid <- knn(X[train,],grid,Y[train],k=k.opt)


# plot data and superimpose decision boundary
plot(NULL, xlim = range(x1.grid), ylim = range(x2.grid),
xlab = "GPA", ylab = "GMAT")
points(X[Y=='admit',], col = 1,pch=19)
points(X[Y=='not admit',], col = 2,pch=19)
points(X[Y=='borderline',], col = 3,pch=19)
points(grid, col = knn.pred.grid,pch='.',cex=3)

# Fit KNN using optimal K
# Compute predicted classes for training and test sets
knn.pred.train=knn(X[train,],X[train,],Y[train],k=k.opt,prob = T)
knn.pred.test=knn(X[train,],X[!train,],Y[train],k=k.opt,prob = T)

# Compute confusion matrices for training and test sets
knn.confmat.train=table(knn.pred.train,Y[train]); print(knn.confmat.train)
knn.confmat.test=table(knn.pred.test,Y[!train]); print(knn.confmat.test)

# Compute misclassification error rates for training and test sets
knn.error.train=mean(knn.pred.train!=Y[train]); print(knn.error.train)
knn.error.test=mean(knn.pred.test!=Y[!train]); print(knn.error.test)

# (e)
# Summary of results
result.1=as.data.frame(c(lda.error.test,qda.error.test,knn.error.test))
rownames(result.1)=c('lda','qda','knn')
colnames(result.1)='Test error rate'
print(result.1)

# Project 2
data.bankruptcy=read.csv('bankruptcy.csv')[,-(6:7)]
attach(data.bankruptcy)

# (a)
# matrix of scatterplots/correlations
chart.Correlation(data.bankruptcy)

# (b)
# Fit logistic regression using all predictors
fit1=glm(Group~.,data=data.bankruptcy,family = binomial)
```

```
# Fit logistic regression using X1 and X3
fit2=glm(Group~X1+X3,data=data.bankruptcy,family = binomial)

# Perform anova chi-squared test to check if X2 and X4 can be jointly dropped.
anova(fit1,fit2,test = 'Chisq')

# Summary of reduced model
summary(fit2)

# Problem 3
# (a)
# Estimated probabilities for train data
lr.prob <- predict(fit2, type = "response")

# Predicted classes (using 0.5 cutoff)
lr.pred <- ifelse(lr.prob >= 0.5, 1, 0)

# Equation of decision booundary
lr.coef=format(coef(fit2),digits = 3)
db=lr.coef[1]
for (i in 2:length(lr.coef)) db=paste(db,'+ (',lr.coef[i],') *',names(lr.coef)[i])
db=paste(db,'=',0)
print(db)

# Compute confusion matrix
lr.confmat=table(lr.pred,Group); print(lr.confmat)

# Compute secsitivity, specificity and error rate
lr.sen=lr.confmat[2,2]/sum(Group==1)
lr.spec=lr.confmat[1,1]/sum(Group==0)
lr.error=1-sum(diag(lr.confmat))/sum(lr.confmat)

# Summary of results
result.2a=as.data.frame(c(lr.sen,lr.spec,lr.error))
rownames(result.2a)=c('sensitivity','specificity','train error')
colnames(result.2a)='Logistic regression (best model)'
print(result.2a)

# Plot ROC curve
plot(roc(Group,lr.prob),legacy.axes = T)

# (b)
# Estimated probabilities for train data
lr.prob <- predict(fit1, type = "response")

# Predicted classes (using 0.5 cutoff)
lr.pred <- ifelse(lr.prob >= 0.5, 1, 0)
```

```
# Equation of decision booundary
lr.coef=format(coef(fit1),digits = 3)
db=lr.coef[1]
for (i in 2:length(lr.coef)) db=paste(db,'+ (',lr.coef[i],') *',names(lr.coef)[i])
db=paste(db,'=',0)
print(db)

# Compute confusion matrix
lr.confmat=table(lr.pred,Group); print(lr.confmat)

# Compute secsitivity, specificity and error rate
lr.sen=lr.confmat[2,2]/sum(Group==1);
lr.spec=lr.confmat[1,1]/sum(Group==0);
lr.error=1-sum(diag(lr.confmat))/sum(lr.confmat)

# Summary of results
result.2b=as.data.frame(c(lr.sen,lr.spec,lr.error))
rownames(result.2b)=c('sensitivity','specificity','train error')
colnames(result.2b)='Logistic regression (full model)'
print(result.2b)

# ROC curve
plot(roc(Group,lr.prob),legacy.axes = T)

# (c)
# Fit LDA
lda.fit=lda(Group~.,data=data.bankruptcy)

# decision booundary equation
lda.db.coef <- s6340.lda(as.factor(data.bankruptcy[,5]),data.bankruptcy[,1:4])
for (i in 1:nrow(lda.db.coef))
{
db=paste('- (',lda.db.coef[i,1],')')
for (j in 2:ncol(lda.db.coef))
{
db=paste(db,'+ (',lda.db.coef[i,j],') *',colnames(lda.db.coef)[j])
}
db=paste(db,'=',0,'[',rownames(lda.db.coef)[i],']')
print(db)
}

# Compute predicted classes
lda.pred=predict(lda.fit)$class

# Compute confusion matrix
lda.confmat=table(lda.pred,Group); print(lda.confmat)
```

```
# Compute secsitivity, specificity and error rate
lda.sen=lda.confmat[2,2]/sum(Group==1);
lda.spec=lda.confmat[1,1]/sum(Group==0);
lda.error=1-sum(diag(lda.confmat))/sum(lda.confmat)

# Summary of results
result.2c=as.data.frame(c(lda.sen,lda.spec,lda.error))
rownames(result.2c)=c('sensitivity','specificity','train error')
colnames(result.2c)='lda (full model)'
print(result.2c)

# ROC curve
plot(roc(Group,predict(lda.fit)$posterior[,2]),legacy.axes = T)

# (d)
# Fit QDA
qda.fit=qda(Group~.,data=data.bankruptcy)

# decision booundary equation
qda.db.coef <- s6340.qda(as.factor(data.bankruptcy[,5]),data.bankruptcy[,1:4])
print(qda.db.coef$quad.coef.matrix)
print(qda.db.coef$linear.coef.vector)
print(qda.db.coef$cutoff)

# Compute predicted classes
qda.pred=predict(qda.fit)$class

# Compute confusion matrix
qda.confmat=table(qda.pred,Group); print(qda.confmat)

# Compute secsitivity, specificity and error rate
qda.sen=qda.confmat[2,2]/sum(Group==1);
qda.spec=qda.confmat[1,1]/sum(Group==0);
qda.error=1-sum(diag(qda.confmat))/sum(qda.confmat)

# Summary of results
result.2d=as.data.frame(c(qda.sen,qda.spec,qda.error))
rownames(result.2d)=c('sensitivity','specificity','train error')
colnames(result.2d)='qda (full model)'
print(result.2d)

# ROC curve
plot(roc(Group,predict(qda.fit)$posterior[,2]),legacy.axes = T)
```