Analysis on Business School Admissions Data

STAT 387

Reem Saleh

A report presented for the final to complete Intro to Statistical Learning Class

Portland State University
10 March 2022

Contents

1 Exploratory analysis **2**
1.1 Creating Test and Train Data 2
1.2 Looking at Data 3
1.2.1 Plots 4

2 LDA **5**
2.1 Computing LDA 5
2.2 Decision Boundary 6
2.3 Confusion Matrix and Misclassification Rate 8
2.3.1 For Test Data 8
2.3.2 For Train Data 8

3 QDA **9**
3.1 Computing QDA 9
3.2 Decision Boundary 9
3.3 Confusion Matrix and Misclassification Rate 11
3.3.1 For Test Data 11
3.3.2 For Train Data 11

4 KNN **12**
4.1 Choosing Optimal K-value 12
4.2 Sensitivity and Specificity 13
4.3 Error rates 14
4.4 AUC 14

5 Conclusion **15**

6 R Code **16**

Final Project Report

Reem Saleh

Throughout this report I work towards identifying the "best" method for classification given the admissions data. I begin with creating the test and training set, creating some exploratory analysis on the data; and once this is completed I will use three different classification methods that were taught in class and use them on the data these are : LDA, QDA, KNN. From here I will look at some conclusions from each method to determine which method is best fit for this data and can predict the admission decision accurately. This is the final report for my Statistical Learning Class.

1 Exploratory analysis

The data that was provided is information from a Business Schools Admissions. The data has information on 85 students with information on their GPA, GMAT score, and the admissions decision on that student.

1.1 Creating Test and Train Data

Before any analysis was done, I created the test and training data which will be used throughout my work and when using the various classification methods. The test data contains five from each group of admissions decision(i.e. five students that were admitted, five not admitted, and five on the border). To do this we are required select the first five rows from each group and create their own vector. From this I created a test.x and test.y which are a dataframe and vector that contains the first five vectors to test our models accuracy(these are the rows the we selected earlier). From here our training data was train.x,train.y and this was used to train our model. The code which is easier to understand is shown below.

```
test.index <- c(2:6,33:37,61:65) #the first five observations from
    each group
test.x <- data.frame(cbind(admissions$GPA,admissions$GMAT))[test.index
    ,] #predictors for test
test.y <- admissions$Group[test.index] #response for test
train.x <- data.frame(cbind(admissions$GPA,admissions$GMAT))[-test.
    index,] #predictors for train
train.y <- admissions$Group[-test.index] #response for train
train.y=as.factor(train.y)
colnames(train.x)=colnames(test.x) = c("GPA", "GMAT")
train.data <- data.frame(cbind(train.x,train.y)) #our new data that is
     for training
```

1.2 Looking at Data

Using the str() function we can see the structure of the data overall and from this we see that we have a total of three variables. Two of which are quantitative, GPA and GMAT, and one that is qualitative, train.y(decision). From here I was told to look at the relationships and some comparisons on the training data. It is important to note that there is not a difference between looking at the full data vs just the training, the only thing is that the training data has less observations since we took some out to use as test data. The findings below will be done using the training data. More detailed code is provided in Section 5 the R code section.
Using the summary we see that the highest GPA and GMAT score is 3.8 and 693, and the lowest is 2.13 and 313. There are 26 admit, 21 border, and 23 notadmit so the data is pretty evenly spread with a higher number of admits overall.

Summary of Train

GPA	GMAT	train.y
Min. :2.130	Min. :321.0	1:26
1st Qu.:2.638	1st Qu.:431.0	2:23
Median :3.010	Median :484.0	3:21
Mean :2.983	Mean :491.6	
3rd Qu.:3.345	3rd Qu.:545.2	
Max. :3.800	Max. :692.0	

Figure 1

Using the pairs function we can get a graph of all predictors against each other. We see that there is a very strong relationship between GPA and GMAT, as one increases the other does along with it so there is a positive linear relationship between the two. Along with this from the correlation matrix we can see that the two are **positively** correlated with correlation coefficient of 0.532.

	GPA	GMAT
GPA	1.0000000	0.5320823
GMAT	0.5320823	1.0000000

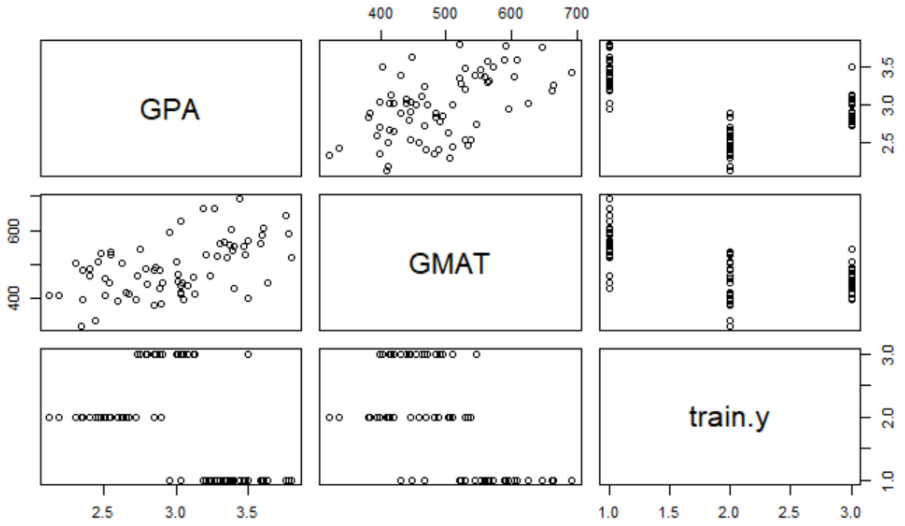
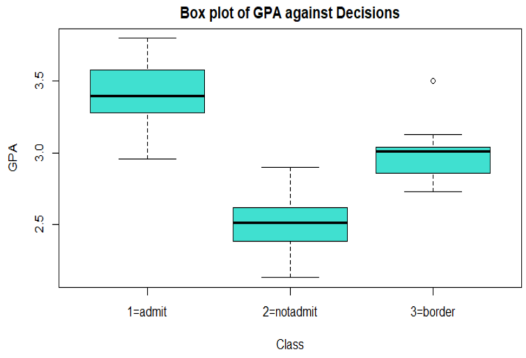
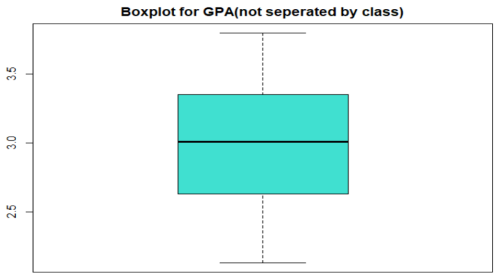


Figure 2

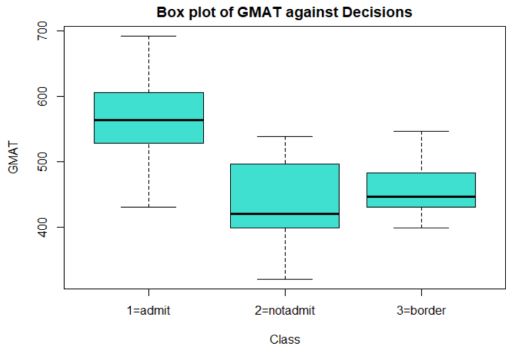
1.2.1 Plots

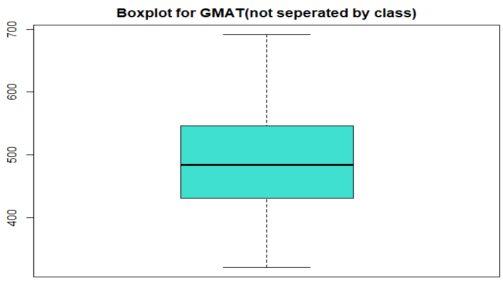
In addition to this I created some box plot to look at each predictor on its own with relation to each class of the decision variable. First I will look at GPA.
From below I added two different box plot both corresponding to GPA, we can see that from the right box plot graph GPA has an average of around 3.0 for the entire training data which is what we got earlier in the summary function, it is also normally distributed with mean of 3. However, lets look at the first figure with 3 box plots each one representing a class. We see that GPA is positively skewed in the admitted class, meaning their are more smaller values than higher valued GPAs for admitted students, more students with GPA between approximately 3.25 and 3.45 than students with GPA between 3.45 and 3.65. The notadmit class has a normal distribution with average GPA of 2.5, and the border class has a left skewed distribution with more values between 3 and 3.2 than between 2.8 and 3. Another very important thing we see is that there is one outlier in the border group, a student with a very high GPA. This can not be seen with the right box plot since it's not seperated by class. We will keep an eye out on this outlier in the future when looking at decision boundaries.





Now let's look at the GMAT box plots. In the right figure we can see GMAT is also normally distributed in the training data with mean around 490. In the left figure we see it is normally distributed in the admitted class with average of 560. Next, both the notadmitted and border class are right skewed with a higher number of low GMATs. In not admitted the majority of values fall between 400 and 420, and in the border most points fall between 420 and 440.





Finally, I created this final plot in ggplot to see how the decisions relate to scores. And from this plot we can see that all of the admitted students have a high GPA and

GMAT score, whereas those border or not admitted have lower scores. We see that there are some borders that have high GPA but low GMAT so this is interesting on the data since it tells us that just because one has a high GPA or GMAT doesn't guarantee acceptance.

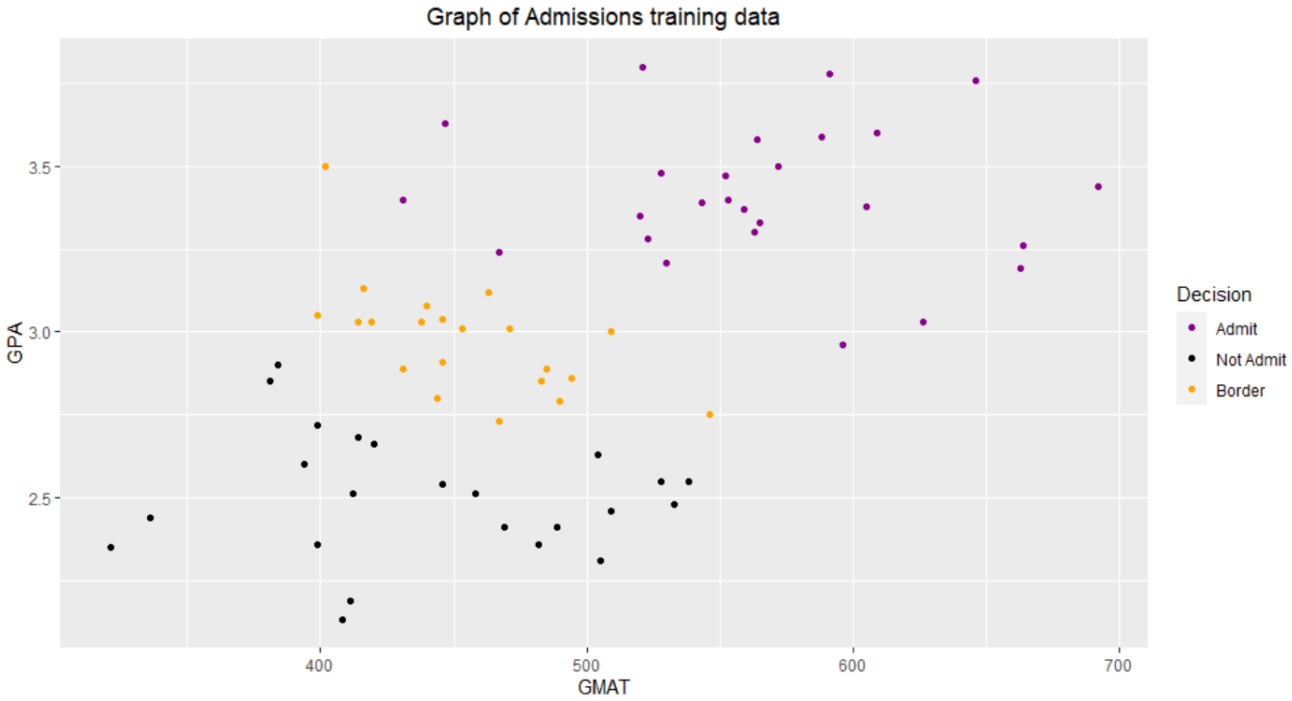


Figure 3

Overall, we can see that both of the predictors, GMAT and GPA will be super helpful in predicting the Decision of the admissions team and although we see some correlation between the two; I am going going to keep both because they play an essential role and if I were to remove one we might miss something and be taking out something crucial. We see a pattern in both the box plots and the scatter plot, that the higher valued GPA and GMATs are in the admitted class and the lowest values in the not admitted class. So both behave similarly with the training data and we saw they were positively correlated, but since we only have two predictors total so there is no loss in keeping both predictors since the data is small, however this might be a different case if we had a large number of predictors and wanted to reduce the amount we had.

2 LDA

2.1 Computing LDA

Below I have added my code that was used to create an LDA on the training data. In addition to this below we can see the summary from the lda.fit. Some things that I thought were interesting to see is the prior probabilities and group means. From the model we can see that it predicts 37% of the students in the training data corresponds to those that were admitted, 33% to students on the border and 30% who are not admitted. The group means tells that the students that are admitted have an average

GPA of 3.41 and GMAT score of 566, the border have a GPA of 3.50 and GMAT of 441 and notadmit have a GPA of 2.98 and GMAT of 455.

Summary of LDA

```
Call:
lda(train.y ~ GPA + GMAT, data = train.data)

Prior probabilities of groups:
        1         2         3
0.3714286 0.3285714 0.3000000

Group means:
       GPA     GMAT
1 3.412308 566.0769
2 2.504348 440.8696
3 2.976190 455.0476

Coefficients of linear discriminants:
              LD1         LD2
GPA  -4.86919500  2.14099181
GMAT -0.01019257 -0.01478839

Proportion of trace:
   LD1    LD2
0.9795 0.0205
```

Figure 4

```
library(MASS) #load library required for LDA and QDA
lda.fit <- lda(train.y~GPA+GMAT, data=train.data)
lda.fit #provide summary on the lda
```

2.2 Decision Boundary

Here I impose a decision boundary on the training data for the LDA model. We can see that these decision boundaries are lines and linear, and the colored areas are called the decision regions. As we learned these decision boundaries and regions are based off of the data points, and each region represents a class(which are also numbered). In this case, 1 = admit, 2 = not admit, 3 = border. We see that there a couple of red numbers and this represents the observations that are in the incorrect region, we see that some of the admits and not admits are in the border region. This is because we choose the higher scores to be admits and lower to be non admits, but there are some cases where an admitted student has a lower GPA. Overall, these decision boundaries make sense in terms of the data and there are only very few variables in the wrong region, so they are sensible and are good regions for now. We will see if the QDA decision boundaries will do a better job.

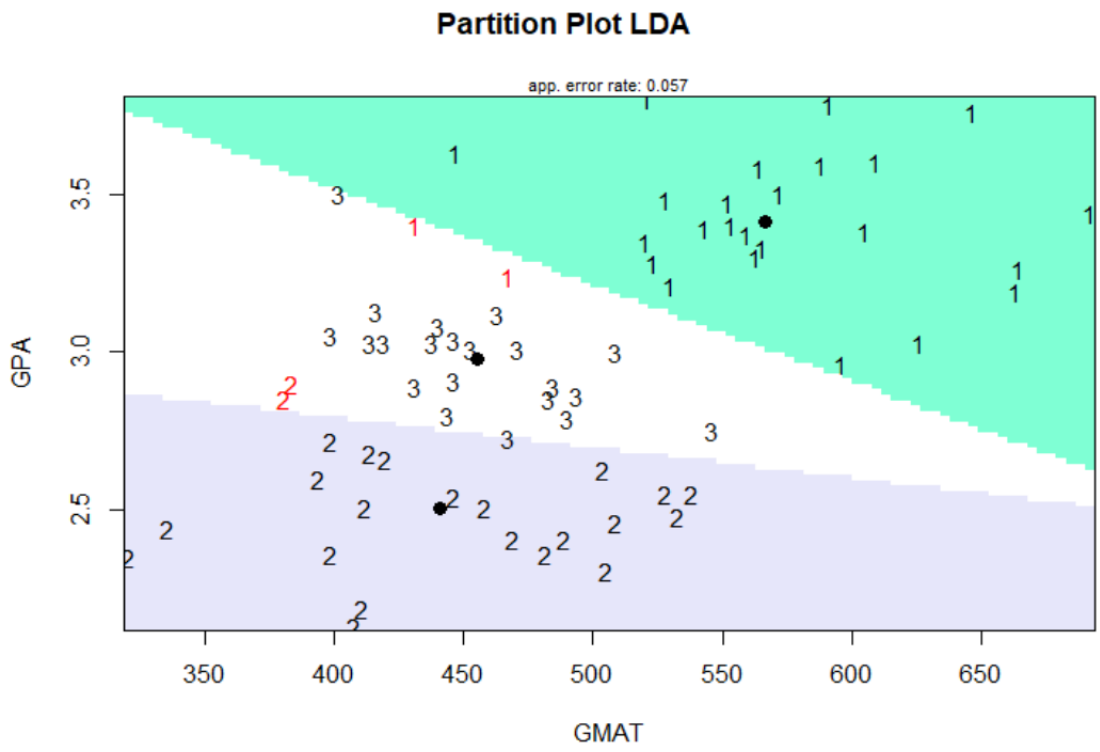


Figure 5

I also created a partition plot corresponding to the test data, just to view, and we can see that we get similar conclusions as above. The decision boundaries are linear and regions are triangular shaped. This time there is only one observation misclassified.

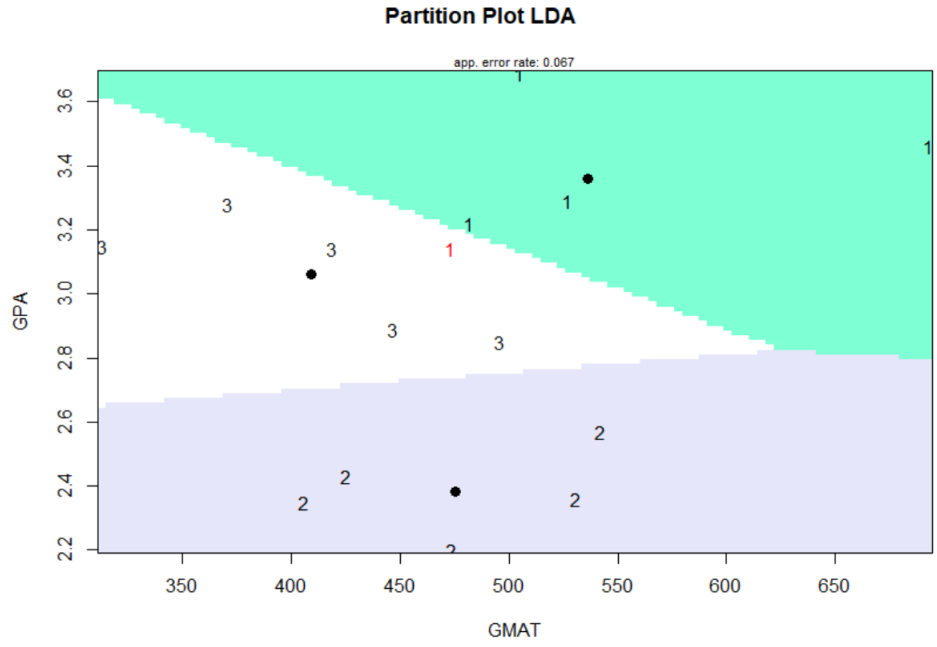


Figure 5b

partimat function was used to create the above graphs

```
library(klaR)
partimat(train.y~., data = train.data, method = "lda", image.colors=
    couleurs, main = "Partition Plot LDA")
```

```
partimat(test.y~., data = test.data, method = "lda", image.colors=
    couleurs, main = "Partition Plot LDA")
```

2.3 Confusion Matrix and Misclassification Rate

The last step is to look at the confusion matrix and misclassification rate of the LDA model. We can see the confusion matrix and misclassification rate code below. Misclassification is the of false negatives and positives/total of observations.

2.3.1 For Test Data

```
lda.fitpred <- predict(lda.fit, test.x)
lda.fitclass <- lda.fitpred$class
table(lda.fitclass, test.y) #produces confusion matrix
mean(lda.fitclass != test.y) #produces misclassification rate
mean(lda.fitclass == test.y) #produces models accuracy on test data
```

From the above code I was able to produce the information below.

```
misclassification error rate is: 0.1333333
                test.y
lda.fitclass 1 2 3
           1 3 0 0
           2 0 5 0
           3 2 0 5
model accuracy: 0.8666667
```

Figure 6

We can see that the LDA model misclassifies around 13% of the time. Which seems very good at first, but lets look at some other things to see if it truly is good. We see that the model is able to accurately predict 13 of the 15 observations correctly=86.67% which is pretty well. And from the confusion matrix we see that it only misclassified two of the test values.

2.3.2 For Train Data

```
lda.fitpred2 <- predict(lda.fit, train.x)
lda.fitclass2 <- lda.fitpred2$class
table(lda.fitclass2, train.y) #produces confusion matrix
mean(lda.fitclass2 != train.y) #produces misclassification rate
mean(lda.fitclass2 == train.y) #produces models accuracy on train data
```

From the above code I was able to produce the information below.

```
misclassification error rate is: 0.05714286
                 train.y
lda.fitclass2  1  2  3
            1 24  0  0
            2  0 21  0
            3  2  2 21
model accuracy: 0.9428571
```

Figure 7

We can see that the LDA model misclassifies around 5.7% of the time. Which is very good and we see that the model is able to accurately predict 66 of the 70 observations correctly=94.29% which is pretty well. And from the confusion matrix we see that it only misclassified four of the training values. Overall, after looking at the confusion matrix and misclassification rate, the LDA model performs very well in predicting the admissions decision.

3 QDA

3.1 Computing QDA

Below I have added my code that was used to create an QDA on the training data. In addition to this below we can see the summary from the qda.fit. From the model we can see that the posterior probabilities and group means are very similar to those in the LDA model, so I will not write the summary again as the values are very similar.

Summary of QDA

```
Call:
qda(train.y ~ GPA + GMAT, data = train.data)

Prior probabilities of groups:
        1         2         3
0.3714286 0.3285714 0.3000000

Group means:
        GPA     GMAT
1 3.412308 566.0769
2 2.504348 440.8696
3 2.976190 455.0476
```

Figure 8

```
#library previousely loaded already
qda.fit<-qda(train.y~GPA+GMAT, data=train.data)
qda.fit
```

3.2 Decision Boundary

Here I impose a decision boundary on the training data for the QDA model as I had done for the LDA one. As from the LDA one the plot has, 1 = admit, 2 = not admit, 3 = border. It is important to realize that the decision boundaries in LDA were linear, but the ones in QDA are non-linear hence the quadratic name, LDA has only linear boundaries. We see in this graph that there is only one missclassified observation compared to four in LDA, so a 75% decrease. This seems way better but it is important to notice that some of these points are right on the border of their boundary. So if were to have another observation that is on the border there is a chance of it being missclassified. There is a region on the bottom right that is considered not admit, but what if someone had a really high GMAT but kind of low GPA, they would not be in the border only not admit or admit, which is unfair and maybe not as accurate. Of course, more thought goes into the admission process(hopefully) but in this case that would be not good. Overall, these decision boundaries make sense in terms of the data and only one in the wrong region, so they are sensible and are good regions for now. We can see that the QDA may be a little but better than LDA.

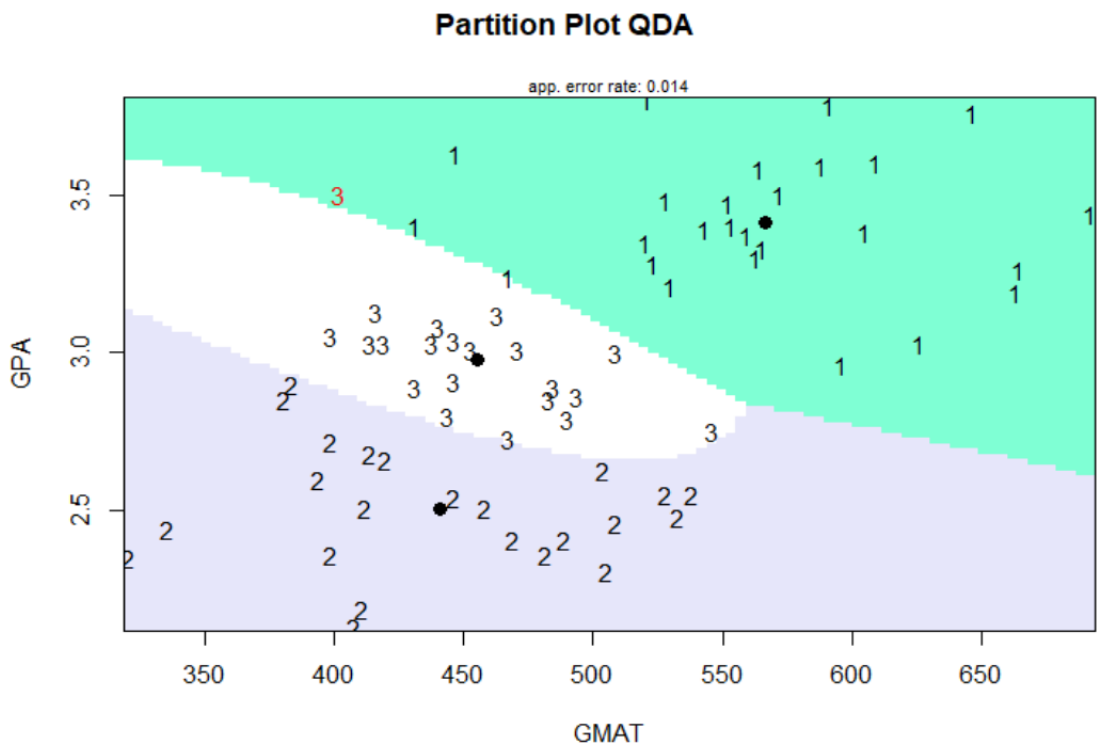


Figure 9

I also created a partition plot corresponding to the test data just for fun and we can see that we get similar conclusions as above. This time there is zero that is misclassified. The shape of the decision regions are a little different but we see that they are still non-linear. The boundaries from the training data make much more sense.

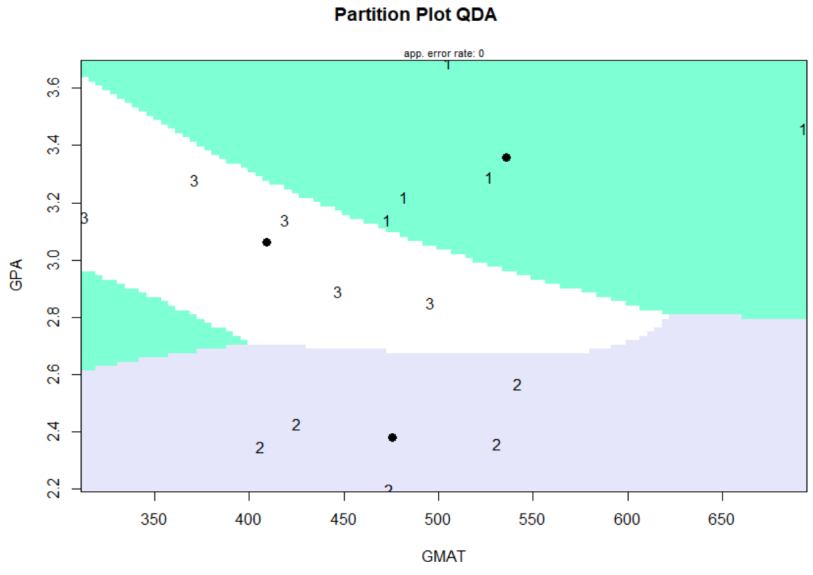


Figure 9b

partimat function was used to create the above graph

```
library(klaR)
partimat(train.y~., data = train.data, method = "qda", image.colors=
    couleurs, main = "Partition Plot QDA")
partimat(test.y~., data = test.data, method = "qda", image.colors=
    couleurs, main = "Partition Plot QDA")
```

3.3 Confusion Matrix and Misclassification Rate

The last step is to look at the confusion matrix and misclassification rate of the QDA model. We can see the confusion matrix and misclassification rate code below. Misclassification is the of false negatives and positives/total of observations.

3.3.1 For Test Data

```
qda.fitpred <- predict(qda.fit, train.x)
qda.fitclass <- qda.fitpred$class
table(qda.fitclass, test.y) #produces confusion matrix
mean(qda.fitclass != test.y) #produces misclassification rate
mean(qda.fitclass == test.y) #produces models accuracy on test data
```

From the above code I was able to produce the information below.

```
misclassification error rate is: 0.1333333
                test.y
qda.fitclass 1 2 3
           1 4 0 0
           2 0 5 1
           3 1 0 4

model accuracy: 0.8666667
```

Figure 10

From this we see that actually the final conclusions that we get from the confusion matrix and misclassification rate is **the exact same as the LDA model**. The QDA model misclassifies around 13% of the time and model is able to accurately predict 13 of the 15 observations correctly=86.67% which is pretty well. And from the confusion matrix we see that it only misclassified two of the test values. Although the misclassification happens in different areas, the number of observations predicted incorrectly are the same. Overall, after looking at the confusion matrix and misclassification rate, we can see that it performs exactly the same as an LDA.

3.3.2 For Train Data

```
qda.fitpred2 <- predict(qda.fit, train.x)
qda.fitclass2 <- qda.fitpred2$class
table(qda.fitclass2, train.y) #produces confusion matrix
mean(qda.fitclass2 != train.y) #produces misclassification rate
mean(qda.fitclass2 == train.y) #produces models accuracy on train data
```

From the above code I was able to produce the information below.

```
misclassification error rate is: 0.01428571
                 train.y
qda.fitclass2  1  2  3
            1 26  0  1
            2  0 23  0
            3  0  0 20

model accuracy: 0.9857143
```

Figure 11

We can see that the QDA model misclassifies training data around 1.4% of the time. Which is very good since its so low and we see that the model is able to accurately predict 69 of the 70 observations correctly=98.57% which is very good! And from the confusion matrix we see that it only misclassified only one of the training values. I notice that this misclassification is in done in group 3, which leads me to remember the outlier that we had found in the EDA section. The GPA value was so high, and the model accidently identified it as admit instead of border. So, if this outlier had been maybe taken out the model would be amazing. Overall, after looking at the confusion matrix and misclassification rate, the QDA model performs very very well, a little bit better than LDA in fact in predicting the admissions decision.

4 KNN

4.1 Choosing Optimal K-value

Before I am able to do anything with KNN model I need to figure out what k-value is the most optimal to use. This will be done by finding the minimal test error rate, which will help us avoid overfitting and underfitting as much as we can. I have created a function that can easily do this for me, which is awesome. So instead of having to create multiple knn models and finding each ones test error rate separately this function goes through dim(test.x) k-values and finds their corresponding test error rates. Once this is done, I can print the minimum error rate, and the corresponding k-value that gives me this minimal number. This obviosely will vary if there is no seed set, so my below conclusions correspond to seed=12
Below I have attached my code for this.

```
library(class)
knn_pred_y = NULL
error_rate = NULL
for(i in 1:dim(test.x)[1]){
set.seed(12)
knn_pred_y = knn(train.x,test.x,train.y,k=i)
error_rate[i] = mean(test.y != knn_pred_y) #misclassification rate
}
min_error_rate = min(error_rate)
print(min_error_rate)
K = which(error_rate == min_error_rate) #model accuracy
print(K)
```

When I print minimum error rate I get a value of 0.5333 and this corresponds to multiple k-values, 3 and 4. The graph below shows this easier, we can see the "elbow" is at the k=3 so I will be using this from here on. Overall, the most optimal KNN model is the one that has **k=3** since it has the minimum test error rate(although k=4 also produces the same rate).

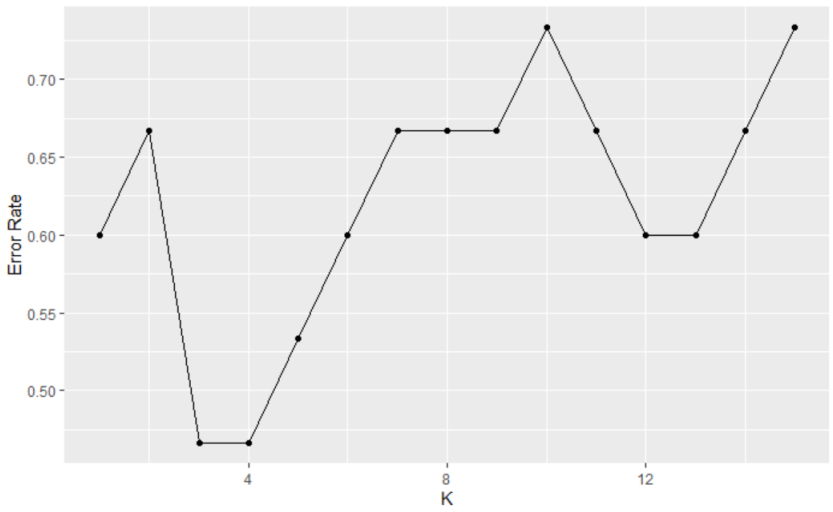


Figure 12

4.2 Sensitivity and Specificity

Finding the sensitivity and specificity is a little different than one would first expect because we do not have a normal 2x2 confusion matrix as we have seen throughout the term. However it is the same concept except for the fact that we have to find the sensitivity and specificity for each class separately. I have found a function in R that can find this for each class on it's own. I have attached the code below. Along with this, recall that sensitivity is equal to the proportion of positive results out of the total number that were actually positive, this is repeated for each class. Specificity is the same but for the true negative values.
code here

```
library(caret)
set.seed(12)
knnpred3<-knn(train.x,test.x,train.y,k=3) #this was the most optimal k
table(knnpred3,test.y)
#below this produces the sensitivity and specificity of each class
test<-as.factor(test.y)
knn<-as.factor(knnpred3)
confusionMatrix(knn,test,positive="1")
```

Below we see a part of what the confusionmatrix() function generates. I have only pasted a small subsection that is relevant, but it output more information such as detection rate, balance accuracy, etc. many of which I haven't learned about yet and don't need.
Specificity of Class 1 : the ability of the model to correctly identify people who will not be in the admitted class, is 1.00 which is perfect
Specificity of Class 2 : the ability of the model to correctly identify people not in the notadmit class is 0.6
Specificity of Class 3 : the ability of the model to correctly identify people not in the border class is 0.7
Sensitivity of Class 1 : the ability of the model to correctly identify people who will be in the admitted class is 0.4
Sensitivity of Class 2 : the ability of the model to correctly identify people who will be

in the notadmitted class is 0.6
Sensitivity of Class 3 : the ability of the model to correctly identify people who will be in the border class is 0.6

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.4000   0.6000   0.6000
Specificity            1.0000   0.6000   0.7000
```

4.3 Error rates

Below are the training and test error rates for K=3.

Figure 13

4.4 AUC

Using the auc() function from the pROC library in R I was able to produce the AUC value. We get a value of 0.8571 which is considerably high. We recall that the AUC value is an overall performance of a classifier summarized over all possible thresholds, AUC can range from between 0-1 with 0 being the worst and the 1 being the best. Since we have a value close to one we see that this classifier is actually much better than expected which is not what we had been getting from the errors we were seeing.

```
library(pROC)
auc(knnpred3, test.y)
```

5 Conclusion

First here are some final conclusions:
Model Accuracy based off of test data:

```
LDA
model accuracy: 0.8666667

misclassification error rate is: 0.1333333
QDA
model accuracy: 0.8666667

misclassification error rate is: 0.1333333
KNN
model accuracy: 0.5333333

misclassification error rate is: 0.4666667
```

We see that KNN has the highest misclassification rate, which we don't want. But now LDA and QDA have the exact same value. So, I take out KNN and compare LDA and QDA with its training data rather than test.
Model Accuracy based off of train data:

```
LDA
model accuracy: 0.9428571

misclassification error rate is: 0.05714286
QDA
model accuracy: 0.9857143
misclassification error rate is: 0.01428571
```

Here we see that the QDA has a higher accuracy than LDA in regards to the training data. So, I will use this to help back up my conclusion.

In the end, after multiple analysis and factors were looked at I have found that either LDA or QDA methods would be best to use. I lean more towards recommending QDA, but given more data it would be good to test both QDA and LDA and choose one leaving out KNN. We saw above that LDA and QDA have significantly lower misclassification rates than KNN, a comparison of 13.33% versus 46.67% for KNN. This is a huge difference so we see that KNN has a tendency to misclassify the observations more than QDA and LDA. Along with this, when comparing the decision boundaries for LDA and QDA there were less observations that were in the wrong region. In addition to this we saw that the KNN was actually a good classifier from the AUC, but I chose not to go with this model because it's a completely non-parametric method and we also don't have a way of finding what predictors are significant or not in the data which would be helpful if given more data to predict decision. So, from these conclusions and others shown in the report I recommend using the QDA method, but that the LDA method also works the same for our sampled data but it is safe to start with QDA since it is more flexible. It is essentially the middle ground and compromise between LDA and KNN since it is more flexible than LDA, but not as much as KNN but worked really well with the limited number of training observations we had with this data. **In the end, the classifier I recommend is QDA.**

6 R Code

This section has all of my R code that was used throughout the entire project. Even if some things were not shown above I used them in my thought process and when working to complete this project.

```
1
2  *********
3  #loading the data
4  admissions <- read.csv("C:\\Users\\User\\Documents\\PSU\\WINTER 2022\\
       STAT 387\\FINAL PROJECT\\admission.csv")
5  str(admissions) #confirming it loaded the entire dataset
6  *********
7  #train and test data
8  test.index <- c(2:6,33:37,61:65) #first 5 rows of each group
9  test.x <- data.frame(cbind(admissions$GPA,admissions$GMAT))[test.index
       ,] #predictors for test
10 test.y <- admissions$Group[test.index] #response for test
11 train.x <- data.frame(cbind(admissions$GPA,admissions$GMAT))[-test.
       index,] #predictors for train
12 train.y <- admissions$Group[-test.index] #response for train
13 train.y=as.factor(train.y)
14 test.y=as.factor(test.y)
15 colnames(train.x)=colnames(test.x) = c("GPA", "GMAT")
16 train.data <- data.frame(cbind(train.x,train.y))
17 xtabs(~ test.y, data = train.data) #double check we pulled 5 from each
        group
18 *********
19 #EDA Code that I used
20 summary(train.data)
21 pairs(train.data)
22 cor(train.data[,-3])
23 library(ggplot2)
24 ggplot(data = train.data) + geom_point(aes(GMAT, GPA, color = train.y)
       ) + scale_color_manual(labels = c("Admit", "Not Admit","Border"),
       values = c("darkmagenta","black","orange")) +guides(color=guide_
       legend("Decision")) + ggtitle("Graph of Admissions training data")
       + theme(plot.title = element_text(hjust = 0.5))
25 boxplot(train.data$GPA) #general boxplot
26 boxplot(train.data$GMAT)
27 boxplot(GPA~train.y, data=train.data, main="Box plot of GPA against
       Decisions", xlab="Class", col="turquoise", border="black", names=c(
       "1=admit","2=notadmit","3=border")) #class specified box plots
28 boxplot(GMAT~train.y, data=train.data, main="Box plot of GMAT against
       Decisions", xlab="Class", col="turquoise", border="black", names=c(
       "1=admit","2=notadmit","3=border"))
29 boxplot(train.data$GMAT, main="Boxplot for GMAT(not seperated by class
       )", col="turquoise", border="black")
30 boxplot(train.data$GPA, data=train.data, main="Boxplot for GPA(not
       seperated by class)", col="turquoise", border="black")
31 *********
32 #LDA model
33 library(MASS)
34 lda.fit<-lda(train.y~GPA+GMAT, data=train.data)
35 lda.fit
36 ##Test data Confusion matrix & Errors
```

```
lda.fitpred <- predict(lda.fit, test.x)
lda.fitclass <- lda.fitpred$class
table(lda.fitclass, test.y)
mean(lda.fitclass != test.y) #equal to the misclassification rate
mean(lda.fitclass == test.y) #produces models accuracy on test data
##Train data Confusion matrix & Errors
lda.fitpred2 <- predict(lda.fit, train.x)
lda.fitclass2 <- lda.fitpred2$class
table(lda.fitclass2, train.y)
mean(lda.fitclass2 != train.y)
mean(lda.fitclass2 == train.y)
##Decision Boundary
library(klaR)
couleurs=c("aquamarine","lavender","white")
partimat(train.y~., data = train.data, method = "lda", image.colors=
    couleurs, main = "Partition Plot LDA") #1=admit, 2=not admit, 3=
    border
*********
#QDA model
library(MASS)
qda.fit<-qda(train.y~GPA+GMAT, data=train.data)
qda.fit
##Test data Confusion matrix & Errors
qda.fitpred <- predict(qda.fit, test.x)
qda.fitclass <- qda.fitpred$class
table(qda.fitclass, test.y)
mean(qda.fitclass != test.y)
mean(qda.fitclass == test.y)
##Train data Confusion matrix & Errors
qda.fitpred2 <- predict(qda.fit, train.x)
qda.fitclass2 <- qda.fitpred2$class
table(qda.fitclass2, train.y)
mean(qda.fitclass2 != train.y)
mean(qda.fitclass2 == train.y)
##Decision Boundary
library(klaR)
couleurs=c("aquamarine","lavender","white")
partimat(train.y~., data = train.data, method = "qda", image.colors=
    couleurs, main = "Partition Plot QDA") #1=admit, 2=not admit, 3=
    border
*********
#KNN model
library(class)
knn_pred_y = NULL
error_rate = NULL
for(i in 1:dim(test.x)[1]){
set.seed(12)
knn_pred_y = knn(train.x,test.x,train.y,k=i)
error_rate[i] = mean(test.y != knn_pred_y)
} #function to go through k-values from 1-15
min_error_rate = min(error_rate)
print(min_error_rate)#prints minimum error rate
K = which(error_rate == min_error_rate)
print(K) #kvalue that has minimum error rate
library(ggplot2)
```

```
qplot(1:dim(test.x)[1], error_rate, xlab = "K", ylab = "Error Rate",
   geom=c("point", "line")) #plot of test error rate and k-values
##Sensitivity and Specificity
library(caret)
set.seed(12)
knnpred3<-knn(train.x,test.x,train.y,k=3) #this was the most optimal k
#this produces the sensitivity and specificity of each class
test<-as.factor(test.y)
knn<-as.factor(knnpred3)
confusionMatrix(knn,test,positive="1")
##Error Rates
table(knnpred3,test.y) #confusion matrix
mean(knnpred3!=test.y)
cat("misclassification error rate is:", mean(knnpred3!=test.y) )
cat("model accuracy:", mean(knnpred3==test.y) )
#used all code below to create test&training error graph
library(class)
M<-dim(test.x)[1]
train.err<-rep(NA,M)
test.err<-rep(NA,M)
for (i in 1:M){
  set.seed(12)
  knn.pred=knn(train.x,test.x,train.y,k=i)
  test.err[i]<-mean(test.y != knn.pred)

  knn.pred=knn(train.x,test.x,train.y,k=i)
  train.err[i]<-mean(test.y != knn.pred)
}
df <-data.frame(c(rep("Training",M),rep("Test",M)),rep(seq(1:M),2),c(
   train.err,test.err))
colnames(df)<-c("Data","K","ErrorRate")
ggplot(data=df, aes(x=K, y=ErrorRate, group=Data, colour=Data)) +
  geom_line() +
  geom_point() +
  ggtitle("Training and Test Errors") + theme(plot.title =
element_text(color="black", face="bold", size=16))
##AUC
library(pROC)
auc(knnpred3, as.numeric(test.y))
*********
```