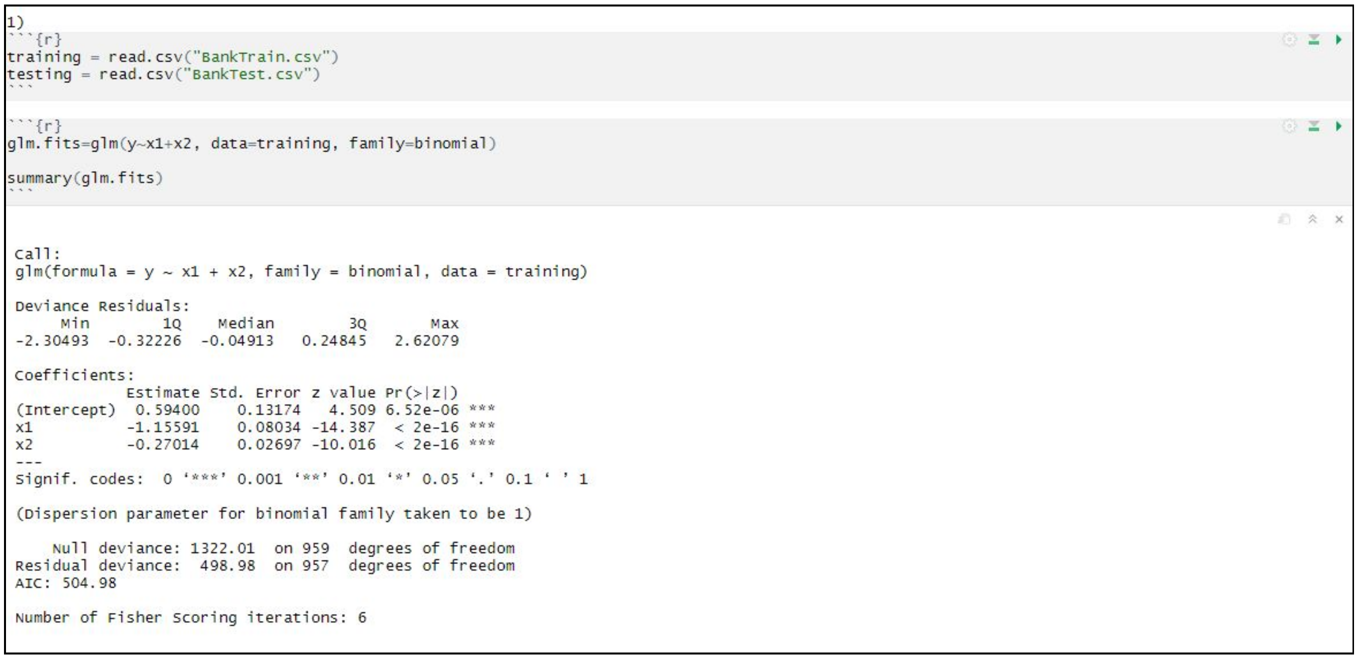
Rhys Jennings - STAT318 - Assignment 2

1)

a)



~~~
1)
```{r}
training = read.csv("BankTrain.csv")
testing = read.csv("BankTest.csv")
~~~

```{r}
glm.fits=glm(y~x1+x2, data=training, family=binomial)

summary(glm.fits)
```

Call:
glm(formula = y ~ x1 + x2, family = binomial, data = training)

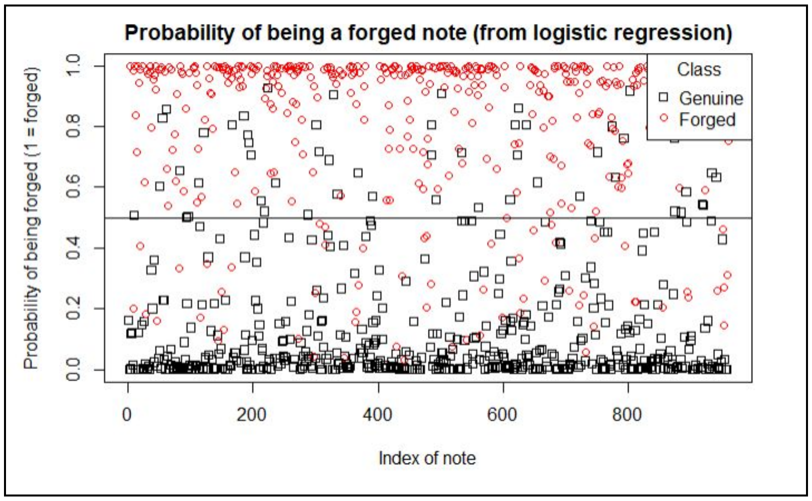
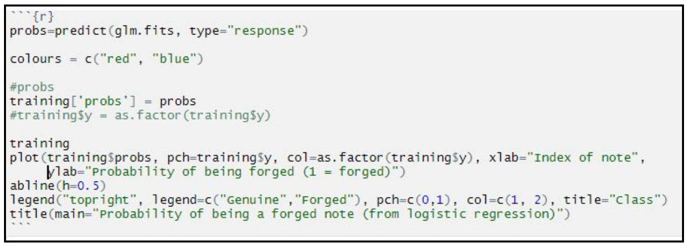
Deviance Residuals:
     Min   1Q   Median   3Q   Max
-2.30493 -0.32226 -0.04913   0.24845   2.62079

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.59400   0.13174   4.509 6.52e-06 ***
x1   -1.15591   0.08034 -14.387 < 2e-16 ***
x2   -0.27014   0.02697 -10.016 < 2e-16 ***

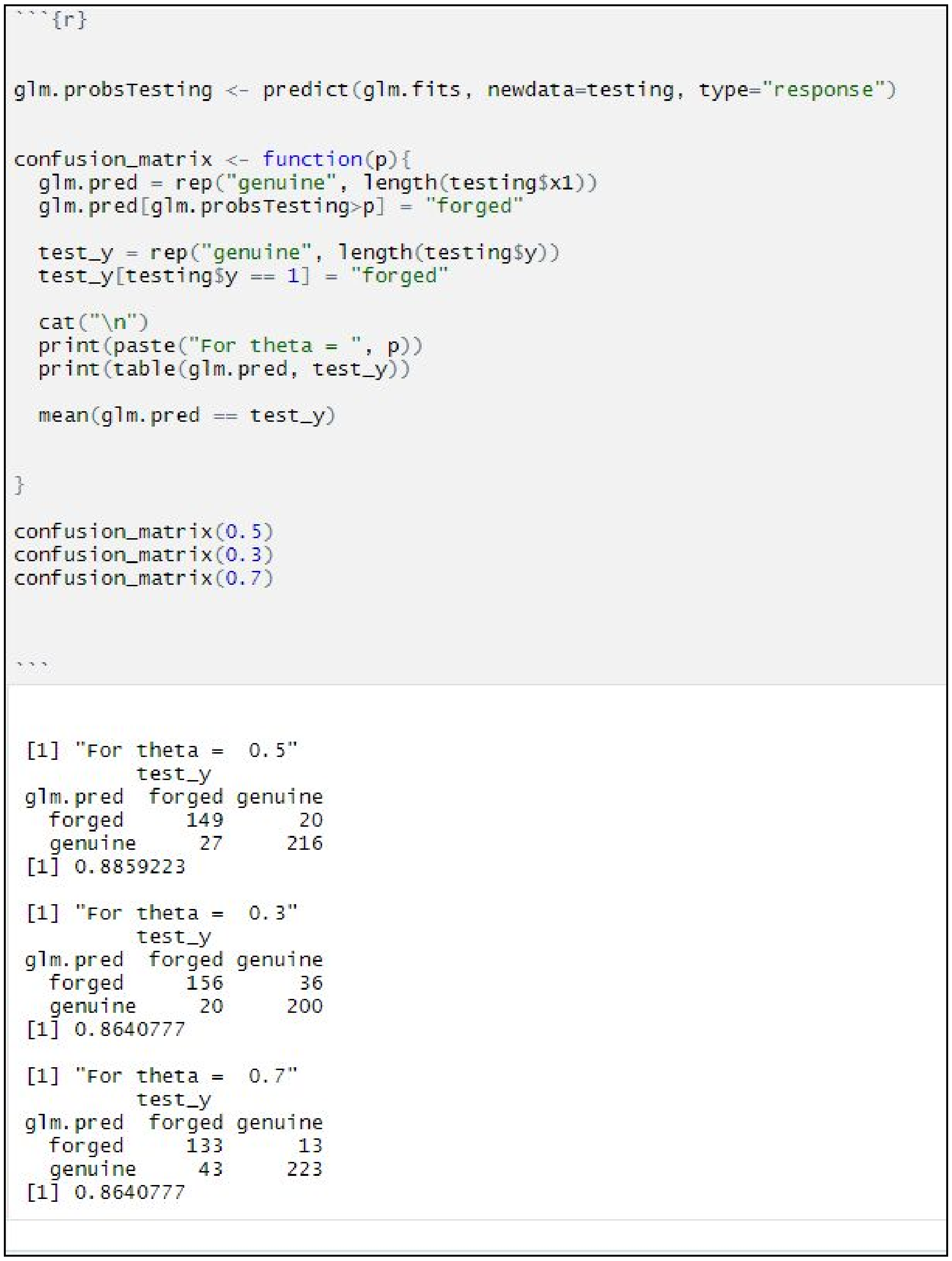
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1322.01 on 959 degrees of freedom
Residual deviance: 498.98 on 957 degrees of freedom
AIC: 504.98

Number of Fisher Scoring iterations: 6
```

The p-values for x1 and x2 are very low so we can conclude that there is an association between both x1 x2 and whether the note has been forged or not. We can see that a one-unit increase in x1 results in decreasing the log odds of being a forged banknote by -1.01(2dp). One unit increase of x2 results in decreasing the log odds of being a forged bank note by -0.28(2dp). This tells us that increasing both these variables reduces the probability that the bank note has been forged.

b) i) *Note: Horizontal line is our decision boundary at 0.5.*


Probability of being a forged note (from logistic regression)
Probability of being forged (1 = forged)
Index of note
Class
Genuine
Forged
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800



```
```{r}
probs=predict(glm.fits, type="response")

colours = c("red", "blue")

#probs
training['probs'] = probs
#training$y = as.factor(training$y)

training
plot(training$probs, pch=training$y, col=as.factor(training$y), xlab="Index of note",
      ylab="Probability of being forged (1 = forged)")
abline(h=0.5)
legend("topright", legend=c("Genuine","Forged"), pch=c(0,1), col=c(1, 2), title="Class")
title(main="Probability of being a forged note (from logistic regression)")
```
```
```

ii) and iii)



~~~
```{r}

glm.probsTesting <- predict(glm.fits, newdata=testing, type="response")


confusion_matrix <- function(p){
  glm.pred = rep("genuine", length(testing$x1))
  glm.pred[glm.probsTesting>p] = "forged"

  test_y = rep("genuine", length(testing$y))
  test_y[testing$y == 1] = "forged"

  cat("\n")
  print(paste("For theta = ", p))
  print(table(glm.pred, test_y))

  mean(glm.pred == test_y)


}

confusion_matrix(0.5)
confusion_matrix(0.3)
confusion_matrix(0.7)



```
~~~

```
 [1] "For theta =  0.5"
         test_y
glm.pred  forged genuine
  forged     149      20
  genuine     27     216
[1] 0.8859223

[1] "For theta =  0.3"
         test_y
glm.pred  forged genuine
  forged     156      36
  genuine     20     200
[1] 0.8640777

[1] "For theta =  0.7"
         test_y
glm.pred  forged genuine
  forged     133      13
  genuine     43     223
[1] 0.8640777
```

Note: Forged notes are our main class of interest, and thus the positive examples. Genuine notes are the negative examples.

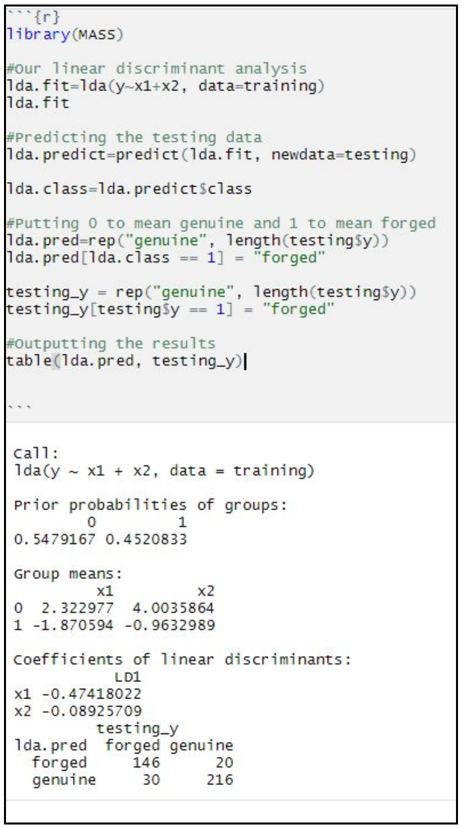
- **Theta=0.5:** The error rate = 47/412 or 0.114 (3dp). The null classifier has an error of 176/412, or 0.427, which is relatively high, meaning it does not appear we have much of a class imbalance. The precision (TP/P*) is 149/169 or 0.847which is also relatively high. Our model appears to be doing better than random guessing.

- **Theta=0.7:** The error rate using theta = 0.3 is 56/412, or 0.136 (3dp), which is worse than when using 0.5 We can see that we are are classifying more genuine notes correctly, however this comes at the cost of decreasing our the amount of forged notes we are correctly classifying. This is because we are moving our boundary further towards 1, meaning we will include more genuine notes but we are more likely to include forged notes

- **Theta=0.3:** Similarly to using a theta of 0.7, we have an error rate of 65/412, or 0.136(3dp). However instead of moving our boundary towards 1, we are bringing it towards 0, meaning we will classify more forged notes correctly, at the cost of classifying more genuine notes as forged.

  Using this boundary would be useful if it is very important to classify a high amount of forged notes, while it is not being very important that he have a higher error rate classifying genuine notes. For example, it is important to dispose of forged notes, whereas disposing genuine notes is not that big of a deal. However, if we had a higher boundary, we would misclassify more forged notes, releasing them back into the public which is more of a danger than simply disposing of a bit more genuine notes.

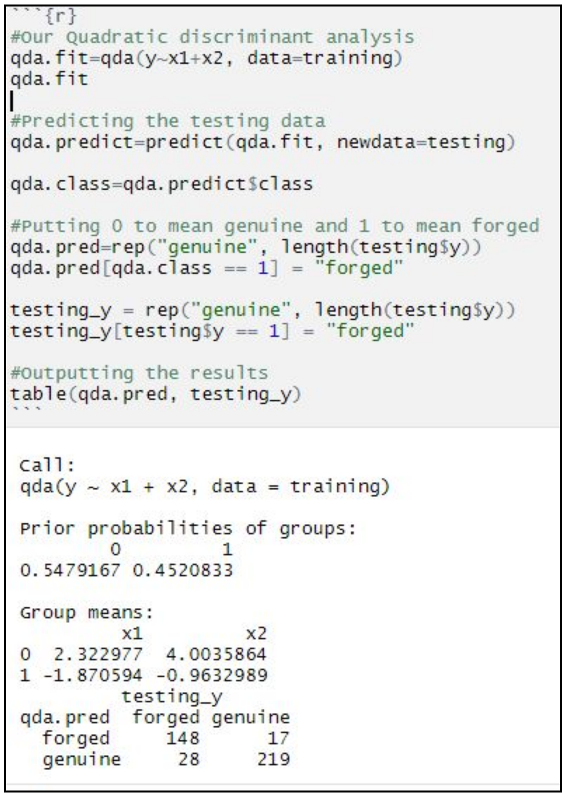
2) a) Linear Discriminant Analysis



~~~
```{r}
library(MASS)

#Our linear discriminant analysis
lda.fit=lda(y~x1+x2, data=training)
lda.fit

#Predicting the testing data
lda.predict=predict(lda.fit, newdata=testing)

lda.class=lda.predict$class

#Putting 0 to mean genuine and 1 to mean forged
lda.pred=rep("genuine", length(testing$y))
lda.pred[lda.class == 1] = "forged"

testing_y = rep("genuine", length(testing$y))
testing_y[testing$y == 1] = "forged"

#Outputting the results
table(lda.pred, testing_y)

```
~~~

```
Call:
lda(y ~ x1 + x2, data = training)

Prior probabilities of groups:
        0         1 
0.5479167 0.4520833 

Group means:
         x1         x2
0  2.322977  4.0035864
1 -1.870594 -0.9632989

Coefficients of linear discriminants:
           LD1
x1 -0.47418022
x2 -0.08925709
         testing_y
lda.pred  forged genuine
  forged     146      20
  genuine     30     216
```

b) Quadratic Discriminant Analysis



~~~
```{r}
#Our Quadratic discriminant analysis
qda.fit=qda(y~x1+x2, data=training)
qda.fit

#Predicting the testing data
qda.predict=predict(qda.fit, newdata=testing)

qda.class=qda.predict$class

#Putting 0 to mean genuine and 1 to mean forged
qda.pred=rep("genuine", length(testing$y))
qda.pred[qda.class == 1] = "forged"

testing_y = rep("genuine", length(testing$y))
testing_y[testing$y == 1] = "forged"

#Outputting the results
table(qda.pred, testing_y)
```
~~~

```
Call:
qda(y ~ x1 + x2, data = training)

Prior probabilities of groups:
        0         1 
0.5479167 0.4520833 

Group means:
         x1         x2
0  2.322977  4.0035864
1 -1.870594 -0.9632989
         testing_y
qda.pred  forged genuine
  forged     148      17
  genuine     28     219
```

Note: Forged notes are our main class of interest, and thus the positive examples. Genuine notes are the negative examples.

LDA: The error rate for this is 50/412, or 0.121(3dp). The TPP is 146/176 or 0.830(3dp). The precision is 146/166 or 0.880(3dp)

QDA: The error rate for this is 45/412, or 0.109(3dp). The TPP is 148/176 or 0.841(3dp). The precision is 148/165, or 0.897(3dp)

Preferably, we would like to find a model that has both good sensitivity, and good specificity. All models perform similarly well. However for this data, it is likely to be more important to catch forged notes than it is to ensure genuine notes are not classed as forged. If this is the case, then the logistic function performs the best on this testing data, classifying the most correct forged notes.

	Sensitivity (3dp)	Specificity (3dp)
Logistic (0.5)	0.847	0.915
LDA	0.830	0.915
QDA	0.841	0.928

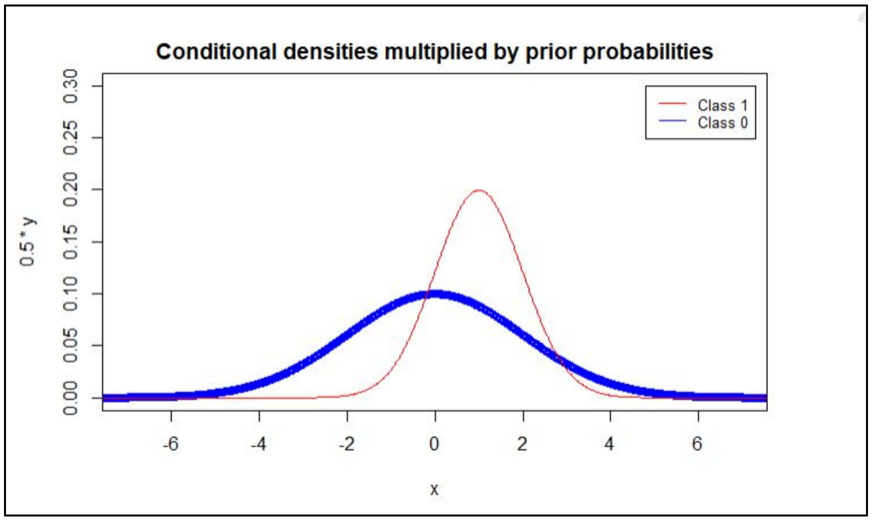
3

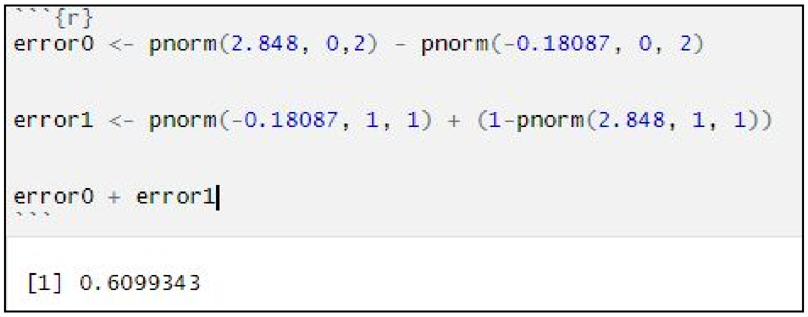
a)

We can find the boundaries are x = -0.18087, 2.848 (Shown mathematically at the back)

The error rate is the percentage of the area for each class, that will be misclassified. We can see there is a big area between -0.18087 and 2.848 that class 0 values will be misclassified, and outside of these values, class 1 will always be misclassified.

We can find the error rate by looking at the area of class that will be misclassified. This results in a error rate of 0.61 (2dp)





~~~
```{r}
error0 <- pnorm(2.848, 0,2) - pnorm(-0.18087, 0, 2)

error1 <- pnorm(-0.18087, 1, 1) + (1-pnorm(2.848, 1, 1))

error0 + error1
```

[1] 0.6099343
~~~

b) To find the boundary, we find where the discriminant score is the same for both classes. Which we find to be:

$$\frac{\mu_0}{\sigma^2}x - \frac{\mu_0^2}{2\sigma^2} = \frac{\mu_1}{\sigma^2}x - \frac{\mu_1^2}{2\sigma^2}$$

*(Note, we did not include log(pi*k) as the probability for both is the same (0.5)*

Because the variance is being assumed as equal for LDA, we rearrange to find the boundary as:

$$x = \frac{1}{2}(\mu_0 + \mu_1)$$

The means for our functions are 0, and 1, so our boundary is at (0+1)/2 = 0.5

To find the error rate for LDA, we need to find:

$$0.5P(X > 0.5|Y = 0) + 0.5P(X < 0.5|Y = 1)$$

QDA: We can see the error rate is quite large because our two normal density functions overlap by quite a lot, meaning a lot of values will be misclassified. The QDA performs worse than random guessing. If the two classes are well separated, with no overlap or not much, this function will perform much better than when they overlap a great deal.

LDA:

Question 3 - finding boundaries

- To find the boundary we set both equations to equal each other.

$$0.5 * \frac{1}{\sqrt{2\pi} * 2} e^{\frac{-1}{2*4}*(x^2)} = 0.5 * \frac{1}{\sqrt{2\pi}} e^{\frac{-1}{2}*(x-1)^2}$$

- We can then divide both sides by 0.5, and divide the left side by 1/sqrt(2pi)

$$0.5 * e^{\frac{-1}{8}*(x^2)} = e^{\frac{-1}{2}*(x-1)^2}$$

- Applying log rules

$$log(0.5) - \frac{x^2}{8} = -\frac{1}{2}(x-1)^2$$

- Multiplying both sides by 8 and expanding the right side.

$$8log(0.5) - x^2 = -4x^2 + 8x - 4$$

- Grouping to one side

$$0 = -3x^2 + 8x - 4 - 8log(0.5)$$

- Applying quadratic formula

$$x = \frac{-8 \pm \sqrt{8^2 - 4*(-3)*8log(0.5)}}{2*(-3)}$$

This results in the boundaries x = -0.18087, x= 2.84754