SPAMBASE Detection

Bharani

```
pacman::p_load(caret, data.table, MASS, ggplot2,gains,data.table, forecast, leaps, tidyverse)
options(digits = 3)
knitr::opts_chunk$set(echo = TRUE, fig.width=12, fig.height=6, fig.path = 'Figs/')
theme_set(theme_classic())
```

Data Inset

```
spambase.df <- read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.
names.df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names
names.df <- as.matrix(names.df[-1,-2])
names.df <- append(names.df, "Spam_Nonspam")
colnames(spambase.df) <- names.df



# Split the data into training (80%) and validation/test set (20%)
set.seed(42)
training.index <- createDataPartition(spambase.df$Spam_Nonspam, p = 0.8, list = FALSE)
spambase.train <- spambase.df[training.index, ]
spambase.valid <- spambase.df[-training.index, ]
```

Question 1

```
mean_table <- aggregate(. ~ Spam_Nonspam, spambase.train, mean)
#View(mean_table)
mean.diff<- mean_table[1,-1] - mean_table[2,-1]
final.diff.df<- sort(abs(mean.diff),decreasing = TRUE)
var.to.use <- colnames(final.diff.df[1:10])
var.use <- c(var.to.use, "Spam_Nonspam")

spambase.train <- setDF(spambase.train)
spambase.valid <- setDF(spambase.valid)
final.train <- data.frame(spambase.train[,(names(spambase.train)) %in% var.use])
final.valid <- data.frame(spambase.valid[,(names(spambase.valid)) %in% var.use])
```

#Top 10 predictors which have highest difference between Spam and Non-Spam average values are as follows: 1. capital_run_length_total 2. capital_run_length_longest 3. capital_run_length_average 4. word_freq_george 5. word_freq_you 6. word_freq_your 7. word_freq_hp 8. word_freq_free 9. word_freq_hpl 10. char_freq_!

NormaliZation of the data

```
norm.values  <- preProcess(final.train[,1:10], method = c("center", "scale"))
    # Transform the data using the estimated parameters
spambase.train.norm <- predict(norm.values, final.train)
spambase.valid.norm <- predict(norm.values, final.valid)
```

LDA Implementation

```
model <- lda(Spam_Nonspam~., data = spambase.train.norm)
model
```

```
## Call:
## lda(Spam_Nonspam ~ ., data = spambase.train.norm)
##
## Prior probabilities of groups:
##     0     1
## 0.604 0.396
##
## Group means:
##   word_freq_free word_freq_you word_freq_your word_freq_hp word_freq_hpl
## 0         -0.227        -0.218         -0.307        0.205         0.187
## 1          0.346         0.332          0.468       -0.312        -0.285
##   word_freq_george char_freq_. capital_run_length_average
## 0            0.149      -0.181                    -0.0865
## 1           -0.227       0.276                     0.1317
##   capital_run_length_longest capital_run_length_total
## 0                     -0.166                   -0.212
## 1                      0.254                    0.323
##
## Coefficients of linear discriminants:
##                                LD1
## word_freq_free              0.4338
## word_freq_you               0.2222
## word_freq_your              0.5557
## word_freq_hp               -0.2354
## word_freq_hpl              -0.1654
## word_freq_george           -0.2013
## char_freq_.                 0.2963
## capital_run_length_average  0.0651
## capital_run_length_longest  0.0869
## capital_run_length_total    0.4144
```

```
names(model)
```

```
##  [1] "prior"   "counts"  "means"   "scaling" "lev"     "svd"     "N"
##  [8] "call"    "terms"   "xlevels"
```

```
model$xlevels
```

```
## named list()
```

Prior probabilities of groups are as follows:

**Non-Spam -> 0.604 **
Spam -> 0.396

Coefficients of linear descriminants are as follows:

word_freq_free 0.3875 word_freq_you 0.2466 word_freq_your 0.5715 word_freq_hp -0.2354 word_freq_hpl -0.1506 word_freq_george -0.2104 char_freq_. 0.3268 capital_run_length_average 0.0527 capital_run_length_longest 0.1297 capital_run_length_total 0.3725

#*These coefficients help us in generating weighted averages for all the values in a table. It means that for the first row of observations if we mulitiply all the observations with the coefficients of Linear descriminants we can generate a new column (LDA Score) with the weighted average for a row thus helping us in putting them on a similar scale.*

#Prediciton

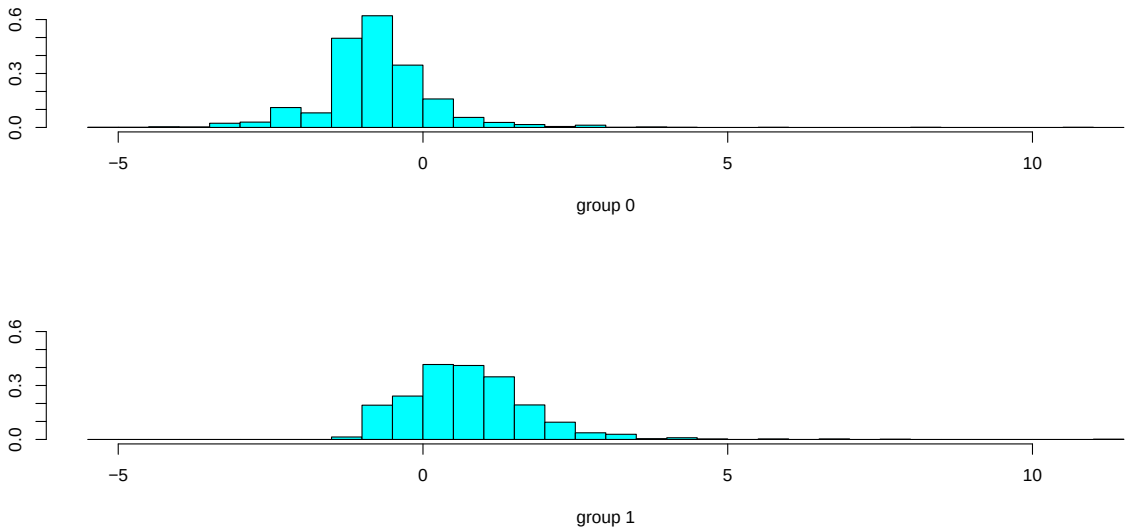
```
pred.train <- predict(model, spambase.train.norm)
```

Creating a column of linear discriminants (x) help us in predicting if a given set of values will be spam/non-spam. Class and Posterior probabilities help us in identifying if a set of value will be classified as Spam or Non-Spam.

#*Observation*: There is 1 Linear discriminant in the model as classification variable has only 2 classes (Spam/Non-spam), as number of linear discrimants are always one less than number of classes (n-1).

#LDA Plot

```
# Generate LDA plot
plot(model)
```




```
# Predict - using Validation data
pred.valid <- predict(model, spambase.valid.norm)
table(pred.valid$class)
```

```
##
##   0   1
## 638 282
```

#Inference The plot help us in differenting between a Spam(Group 1)/Non-Spam(Group 0) email. From the plot we can see that LDA scores for Non Spam emails tend towards values lower than zero (negative) whereas for Spam email LDA score tend towards values more than zero (positive). From this we can interpret that if LDA score has a postive value it will be more likely a Spam email whereas if it has a negative value it has more chances of being identified as Non-Spam.

Confusion matrix to understand the prediction

```
# Confusion matrix

cm <- table(pred.valid$class, spambase.valid.norm$Spam_Nonspam)
confusionMatrix(cm)
```

```
## Confusion Matrix and Statistics
##
##
##       0   1
##   0 520 118
##   1  46 236
##
##                Accuracy : 0.822
##                  95% CI : (0.795, 0.846)
```

```
##     No Information Rate : 0.615
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.609
##
##  Mcnemar's Test P-Value : 2.95e-08
##
##             Sensitivity : 0.919
##             Specificity : 0.667
##          Pos Pred Value : 0.815
##          Neg Pred Value : 0.837
##              Prevalence : 0.615
##          Detection Rate : 0.565
##    Detection Prevalence : 0.693
##       Balanced Accuracy : 0.793
##
##        'Positive' Class : 0
##
```

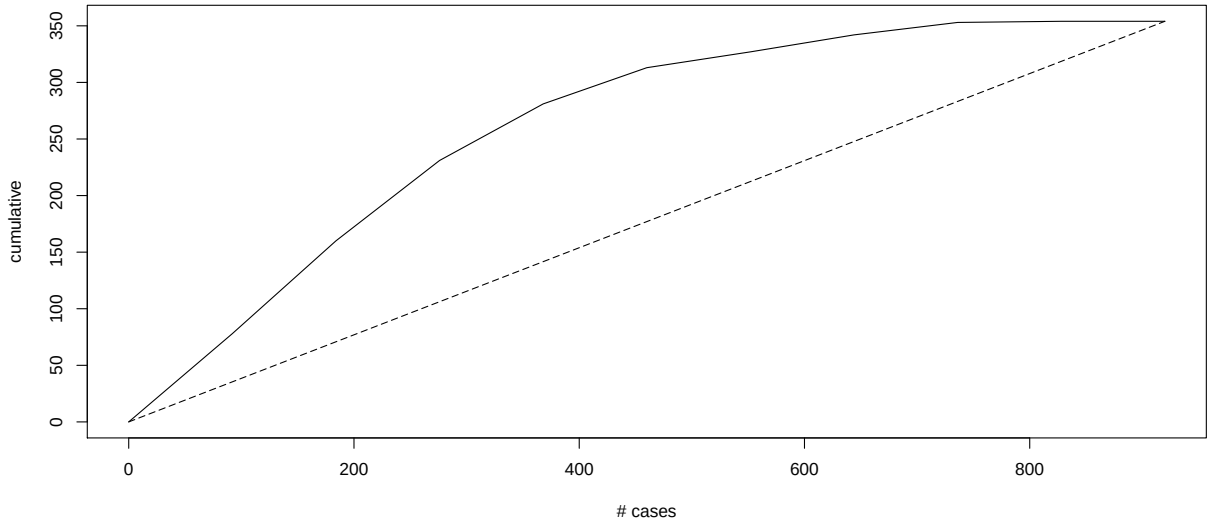
Sensitivity : 0.919

Specificity : 0.667

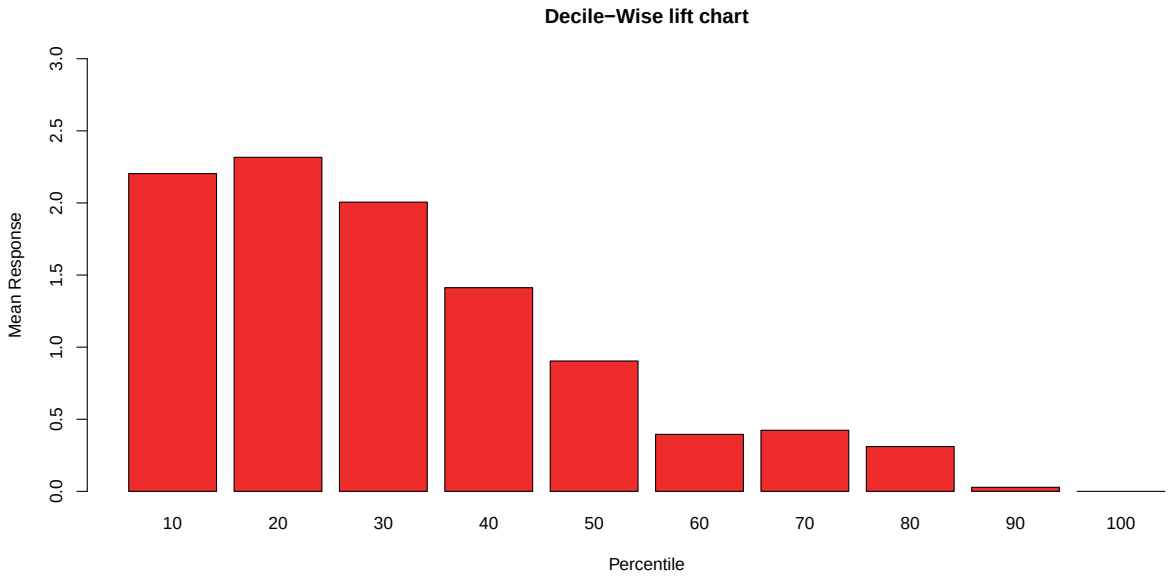
Understanding the gains with Lift and Decile Chart

```
gain <- gains((spambase.valid.norm$Spam_Nonspam), pred.valid$posterior[,2],groups=10)

#lift chart
spam_num <- spambase.valid.norm$Spam_Nonspam
plot(c(0,gain$cume.pct.of.total*sum(spambase.valid.norm$Spam_Nonspam))~c(0,gain$cume.obs),
     xlab = "# cases", ylab = "cumulative", main = "", type = "l")
        lines(c(0,sum(spam_num))~c(0, dim(final.valid)[1]), lty =5)
```




```
#Decile-chart
heights <-gain$mean.resp/mean(spam_num)
midpoints <- barplot(heights,names.arg = gain$depth , ylim = c(0,3), col= "firebrick2",
                     xlab = "Percentile", ylab = "Mean Response", main= "Decile-Wise lift chart")
```



#From the **lift chart** we can see that our model is doing relatively good compared to no model line. There is a visible lift and we are able to identify almost 100% of Spam/Non-Spam till ~ 750 observations while our model gives the same classification of Spam/Non-spam in 920 observations (validations dataset). # From the **decile chart** we can see that our second decile is doing the best in identifying Spam/Non-Spam emails whereas 4 deciles in our model help us in classifying most of our observations. This spike shows that our model is doing relatively better that no model.

What happens if we change the prediciton threshold ?

```
spam <- pred.valid$posterior[,2] >=0.2
table(spam)
```

```
## spam
## FALSE  TRUE
##   383   537
```

```
mean(pred.valid$class == spambase.valid.norm$Spam_Nonspam)
```

```
## [1] 0.822
```

```
mean(spam == spambase.valid.norm$Spam_Nonspam)
```

```
## [1] 0.742
```

We can see that the accuracy of the model reduces when we set the threshold probability of 0.2. This happens as our model is now identifying an email to be spam when the probability is just over 0.2, so it is incorrectly classifying some emails as spam therefore reducing our model accuracy from 82.2% to 74.2%.