Spambase

Poon HY

10/30/2021

1 INTRODUCTION

The aim of this project is to develop a model that predicts whether emails are spam, i.e. unsolicited commercial email. The model is developed from a dataset comprising of thousands of emails with a set of attributes, namely the frequency of occurrence of certain words, characters and capital letters.

The dataset is downloaded from the UC Irvine Machine Learning repository at https://archive.ics.uci.edu/ml. The code below downloads the dataset, separates it into two sets ("temp" and "val_set"), and further separates the temp set into a training dataset ("train_set") and testing dataset ("test_set"). The datasets **train_set** and **test_set** will be used to develop the spam detection model which will then be applied to **val_set** as a final test of the model.

```
# Install packages as required
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")

# Load required libraries
library(tidyverse)
library(caret)
library(e1071)

# Set global option for significant digits
options(digits = 4)

# Load data from UCI site
url_uci <-
  "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data"
spambase <- read.csv(url_uci, header=FALSE)

# Insert column names
colnames(spambase) <- c("f_make", "f_address", "f_all", "f_3d", "f_our", "f_over",
                        "f_remove", "f_internet", "f_order", "f_mail", "f_receive",
                        "f_will", "f_people", "f_report", "f_addresses", "f_free",
                        "f_business", "f_email", "f_you", "f_credit", "f_your",
                        "f_font", "f_000", "f_money", "f_hp", "f_hpl", "f_george",
                        "f_650", "f_lab", "f_labs", "f_telnet", "f_857", "f_data",
                        "f_415", "f_85", "f_technology", "f_1999", "f_parts", "f_pm",
                        "f_direct", "f_cs", "f_meeting", "f_original", "f_project",
                        "f_re", "f_edu", "f_table", "f_conference",
```

```
                        "f_;", "f_(", "f_[", "f_!", "f_$", "f_3",
                        "c_average", "c_longest", "c_total", "spam")

# Classify spam outcome as factor of 2 levels "0" (false) and "1" (true)
spambase$spam <- factor(spambase$spam, levels = c(0,1))

# Set aside 10% of dataset as validation set
set.seed(2021, sample.kind="Rounding")
val_index <- createDataPartition(spambase$spam, times = 1, p = 0.1, list = FALSE)
temp <- spambase[-val_index,]
val_set <- spambase[val_index,]

# Split remaining data into train_set (80%) and test_set (20%)
set.seed(2021, sample.kind = "Rounding")
test_index <- createDataPartition(temp$spam, times = 1, p = 0.2, list = FALSE)
train_set <- temp[-test_index,]
test_set <- temp[test_index,]
```

So now we have 3 relevant datasets: **train_set**, **test_set** and **val_set**, which have the following number of rows:

```
##                Number_of_rows
## train_set                3311
## test_set                  829
## Validation set            461
```

2 ANALYSIS

2.1 Description of data

The first 3 rows of train_set are displayed below:

```
##   f_make f_address f_all f_3d f_our f_over f_remove f_internet f_order f_mail
## 1   0.00      0.64  0.64    0  0.32   0.00     0.00       0.00    0.00   0.00
## 2   0.21      0.28  0.50    0  0.14   0.28     0.21       0.07    0.00   0.94
## 3   0.06      0.00  0.71    0  1.23   0.19     0.19       0.12    0.64   0.25
##   f_receive f_will f_people f_report f_addresses f_free f_business f_email
## 1      0.00   0.64     0.00     0.00        0.00   0.32       0.00    1.29
## 2      0.21   0.79     0.65     0.21        0.14   0.14       0.07    0.28
## 3      0.38   0.45     0.12     0.00        1.75   0.06       0.06    1.03
##   f_you f_credit f_your f_font f_000 f_money f_hp f_hpl f_george f_650 f_lab
## 1  1.93     0.00   0.96      0  0.00    0.00    0     0        0     0     0
## 2  3.47     0.00   1.59      0  0.43    0.43    0     0        0     0     0
## 3  1.36     0.32   0.51      0  1.16    0.06    0     0        0     0     0
##   f_labs f_telnet f_857 f_data f_415 f_85 f_technology f_1999 f_parts f_pm
## 1      0        0     0      0     0    0            0   0.00       0    0
## 2      0        0     0      0     0    0            0   0.07       0    0
## 3      0        0     0      0     0    0            0   0.00       0    0
##   f_direct f_cs f_meeting f_original f_project f_re f_edu f_table f_conference
## 1     0.00    0         0       0.00         0 0.00  0.00       0            0
## 2     0.00    0         0       0.00         0 0.00  0.00       0            0
```

```
## 3     0.06    0         0       0.12         0 0.06  0.06       0            0
##    f_;   f_( f_[   f_!   f_$   f_3 c_average c_longest c_total spam
## 1 0.00 0.000   0 0.778 0.000 0.000     3.756        61     278    1
## 2 0.00 0.132   0 0.372 0.180 0.048     5.114       101    1028    1
## 3 0.01 0.143   0 0.276 0.184 0.010     9.821       485    2259    1
```

The last column, "spam", indicates whether the email is spam ("1") or not ("0"). The first 48 columns show the frequencies of occurrence of certain words, measured in percentage of occurrence against the total number of words in the email. For example, an "f_make" value of 0.15 means that the word "make" made up 0.15% of the number of words in the email. The next 6 columns show the same, but for certain characters instead of words. Then the next 3 columns are all about capital letters - the average length of uninterrupted sequences, length of longest sequence, and total number.

The proportion of spam emails in the dataset is 39.4%, as given by the simple code below.

```
sum(spambase$spam==1)/nrow(spambase)*100
```

```
## [1] 39.4
```

2.2 Summary statistics

I now look at the summary statistics of the spambase dataset. For brevity, I will just show the statistics of the first 8 columns:

```
# Show summary statistics of first 6 columns of spambase dataset
summary(spambase[1:8])
```

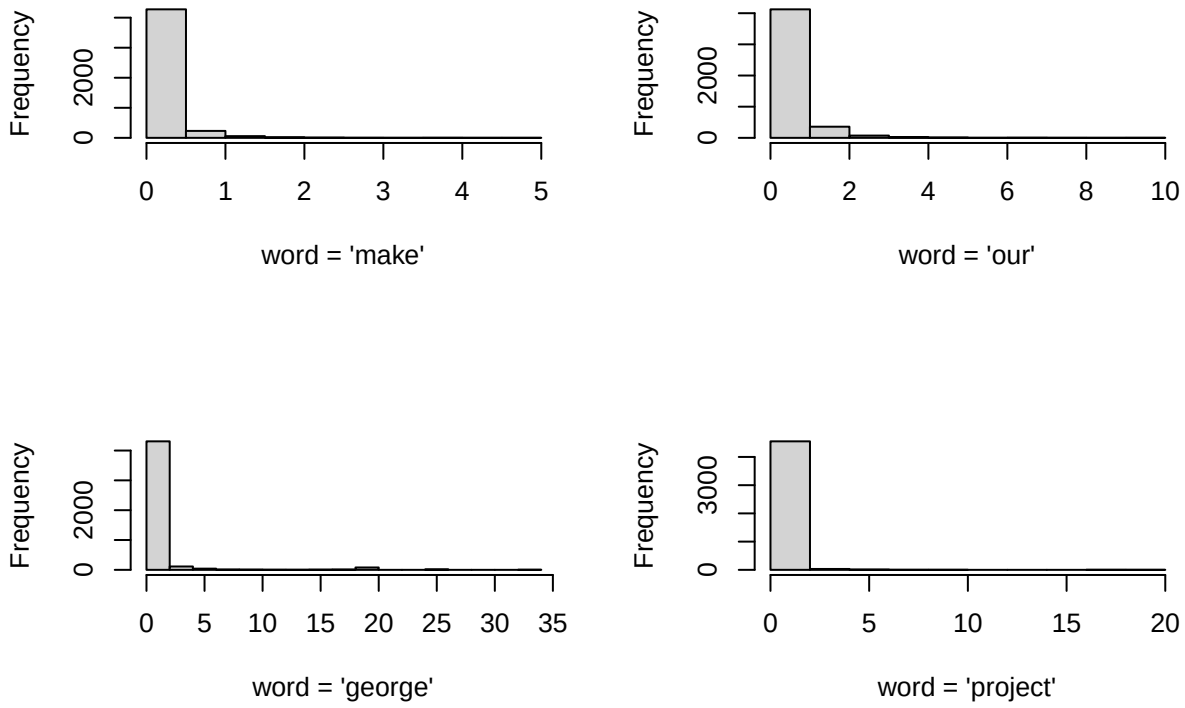
```
##      f_make        f_address          f_all            f_3d
##  Min.   :0.000   Min.   : 0.000   Min.   :0.000   Min.   : 0.00
##  1st Qu.:0.000   1st Qu.: 0.000   1st Qu.:0.000   1st Qu.: 0.00
##  Median :0.000   Median : 0.000   Median :0.000   Median : 0.00
##  Mean   :0.105   Mean   : 0.213   Mean   :0.281   Mean   : 0.07
##  3rd Qu.:0.000   3rd Qu.: 0.000   3rd Qu.:0.420   3rd Qu.: 0.00
##  Max.   :4.540   Max.   :14.280   Max.   :5.100   Max.   :42.81
##      f_our            f_over         f_remove       f_internet
##  Min.   : 0.000   Min.   :0.000   Min.   :0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.:0.000   1st Qu.:0.000   1st Qu.: 0.000
##  Median : 0.000   Median :0.000   Median :0.000   Median : 0.000
##  Mean   : 0.312   Mean   :0.096   Mean   :0.114   Mean   : 0.105
##  3rd Qu.: 0.380   3rd Qu.:0.000   3rd Qu.:0.000   3rd Qu.: 0.000
##  Max.   :10.000   Max.   :5.880   Max.   :7.270   Max.   :11.110
```

A few words like "you" and "your", and the characters on the list appear in most of the emails, as indicated by the non-zero median occurrence. Most of the words on the list, however, do not appear in most of the emails, since the median number of occurrence is zero.

When I plotted histograms of the frequency of 4 words in the list, you can see that the typical skew towards the zero end of the count.

```
# Plot 4 histograms in 2 rows of 2
par(mfrow = c(2,2))
hist(spambase$f_make, main = NULL, xlab = "word = 'make'")
```

```
hist(spambase$f_our, main = NULL, xlab = "word = 'our'")
hist(spambase$f_george, main = NULL, xlab = "word = 'george'")
hist(spambase$f_project, main = NULL, xlab = "word = 'project'")
```

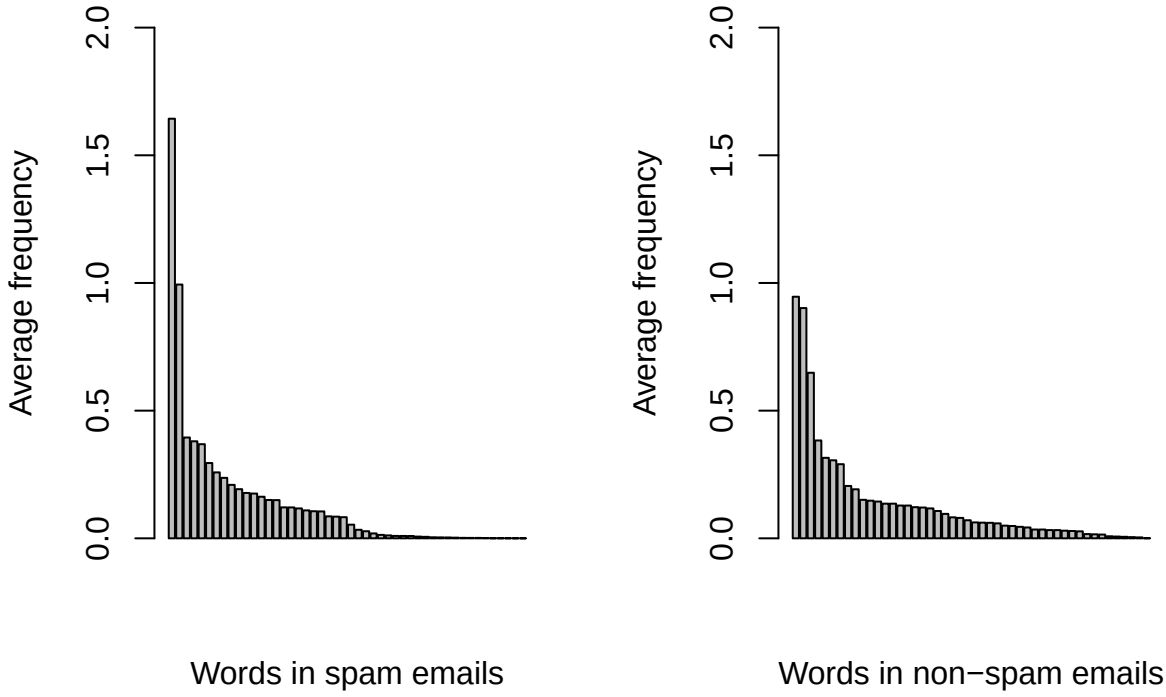


2.3 Frequency of occurrence of words

How do the distributions of key words in columns 1 to 48 look for spam and non-spam emails? Using train_set, I looked at this with the code below.

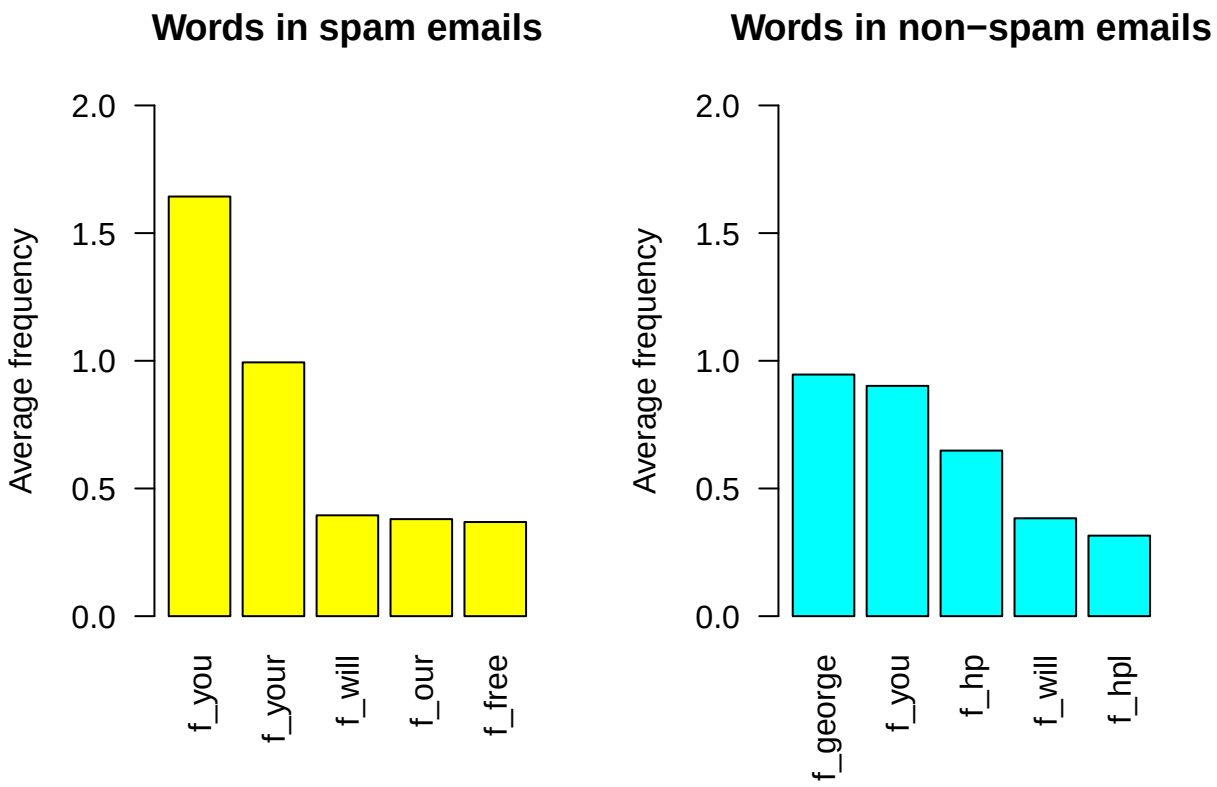
```
# Plot variables with high
num_spam_1 <- sum(spambase$spam==1)
# Number of non-spam emails
num_spam_0 <- sum(spambase$spam==0)
# Frequency of words per spam email
freq_w_1 <- colSums(train_set[which(train_set$spam==1), 1:48])/num_spam_1
# Frequency of words per non=spam email
freq_w_0 <- colSums(train_set[which(train_set$spam==0), 1:48])/num_spam_0
# Put bar charts of average frequencies of words for spam and non-spam emails
# side-by-side
# Average frequency of words for spam email in descending order
order_w_1 <- freq_w_1[order(-freq_w_1)]
# Average frequency of words for non-spam email in descending order
order_w_0 <- freq_w_0[order(-freq_w_0)]
# 1 row of 2 plots
```

```
par(mfrow = c(1,2))
# Set same y-axis scales with ylim for comparability
barplot(order_w_1, ylim = c(0, 2), xlab = "Words in spam emails",
        ylab = "Average frequency", xaxt = 'n')
barplot(order_w_0, ylim = c(0, 2), xlab = "Words in non-spam emails",
        ylab = "Average frequency", xaxt = 'n')
```



The bars are plotted in descending order of the average frequencies of the words' occurrences. Other than the first bar ion the spam group, the distribution looks similar, though of course the order of the bars do not represent the same words in both sets. To see which bars represent which words, we zoom in to the first 5 bars of each set.

```
# Plot top 5 frequency words of spam and non-spam emails side-by-side
# 1 row of 2 plots
par(mfrow = c(1,2))
barplot(order_w_1[1:5], main = "Words in spam emails", ylim = c(0, 2), las=2,
        ylab = "Average frequency", col = "yellow")
barplot(order_w_0[1:5], main = "Words in non-spam emails", ylim = c(0, 2), las=2,
        ylab = "Average frequency", col = "cyan")
```

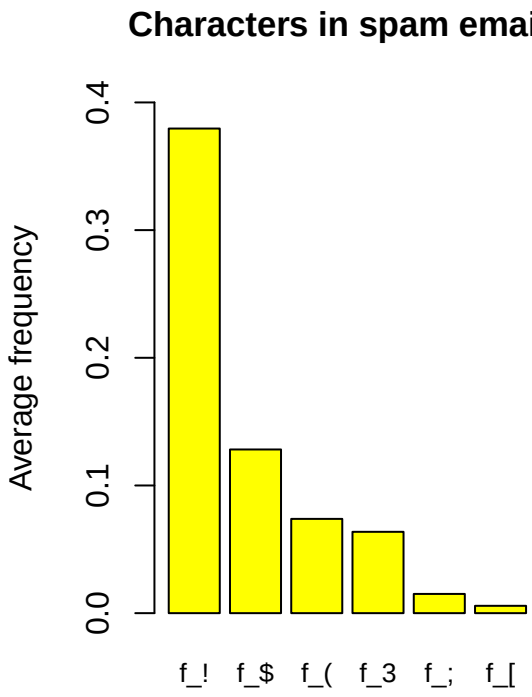


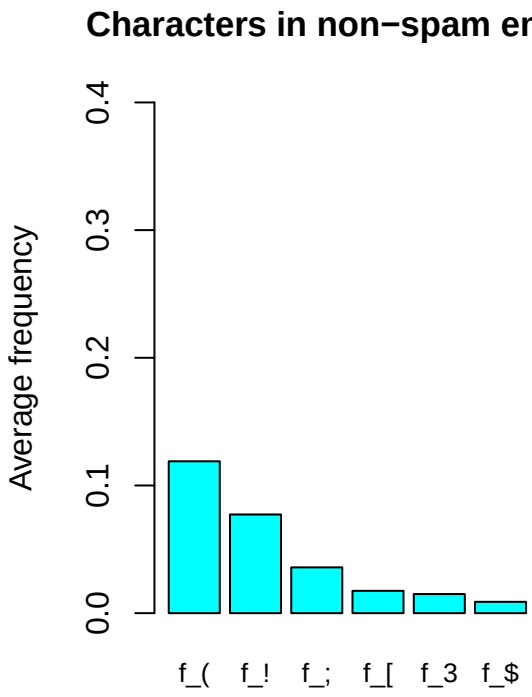
As expected, words with top occurrences in both sets are not the same, except "will" which appears in the top 5 in both.

2.4 Frequency of occurrence of characters

I did the same for the occurrence of characters in columns 49 to 54.

```
# Frequency of characters per spam email
freq_c_1 <- colSums(train_set[which(train_set$spam==1), 49:54])/num_spam_1
# Frequency of characters per non=spam email
freq_c_0 <- colSums(train_set[which(train_set$spam==0), 49:54])/num_spam_0
# Put bar charts of average frequencies of characters for spam and non-spam emails
# side-by-side
# 1 row of 2 plots
par(mfrow = c(1,2))
# Set same y-axis scales with ylim for comparability
barplot(freq_c_1[order(-freq_c_1)], main = "Characters in spam emails",
        ylim = c(0, 0.4), ylab = "Average frequency", col = "yellow",
        cex.main = 1.1, cex.names = 0.8)
barplot(freq_c_0[order(-freq_c_0)], main = "Characters in non-spam emails",
        ylim = c(0, 0.4), ylab = "Average frequency", col = "cyan",
        cex.main = 1.1, cex.names = 0.8)
```



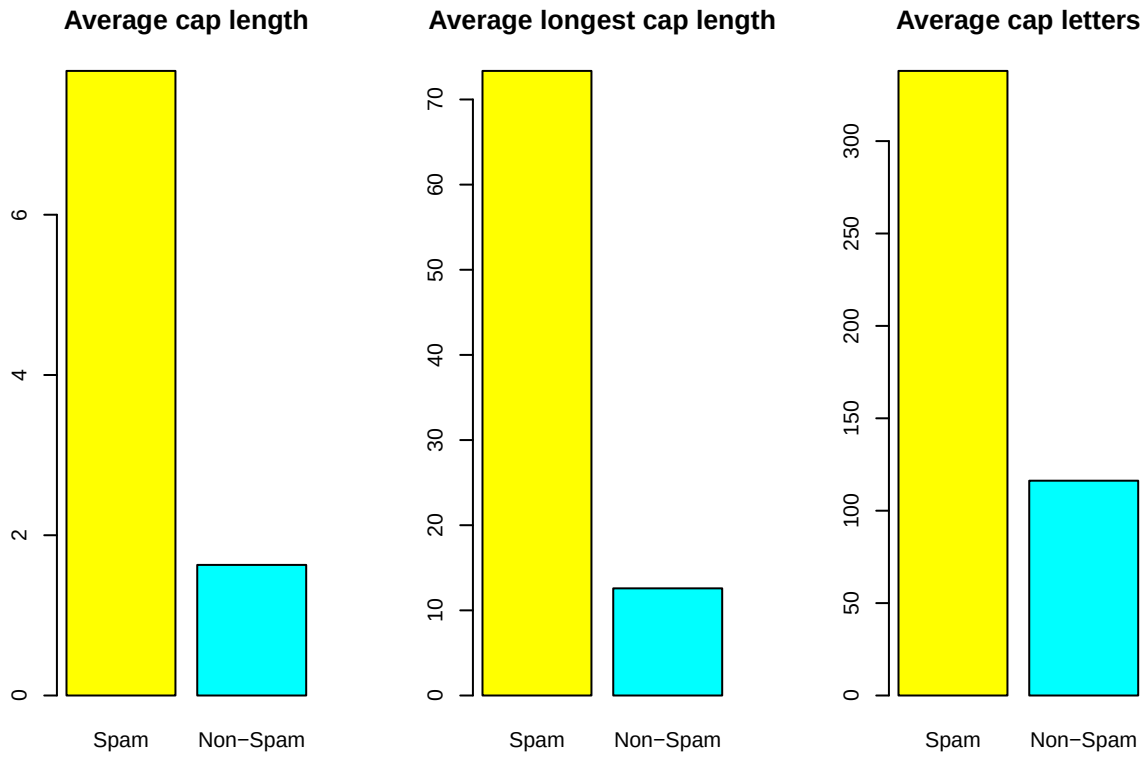


Interestingly, the characters "!" and "$" appear much more often in the spam emails, which makes sense since the purpose of spam is to grab people's attention and reach into their wallets.

2.5 Capital letters

Next, I looked at the 3 columns with the capital letter attributes.

```
# Capital letter occurrences per spam email
cap_1 <- colSums(train_set[which(train_set$spam==1), 55:57])/num_spam_1
# Capital letter occurrences per non=spam email
cap_0 <- colSums(train_set[which(train_set$spam==0), 55:57])/num_spam_0
# Insert a row of 3 bar charts with 3 capital letter attributes
par(mfrow = c(1,3))
# Plot of average length of uninterrupted sequences of capital letters
barplot(c(cap_1[1], cap_0[1]), main = "Average cap length",
        names.arg = c("Spam", "Non-Spam"), col = c("yellow", "cyan"))
# Plot of avearge longest sequence of capital letters
barplot(c(cap_1[2], cap_0[2]), main = "Average longest cap length",
        names.arg = c("Spam", "Non-Spam"), col = c("yellow", "cyan"))
# Plot of average number of capital letters
barplot(c(cap_1[3], cap_0[3]), main = "Average cap letters",
        names.arg = c("Spam", "Non-Spam"), col = c("yellow", "cyan"))
```



This is where we see quite a stark difference between spam and non-spam emails. It seems capital letters feature quite prominently in spam emails.

3 METHODS FOR CONSTRUCTING PREDICTION MODELS

3.1 Accuracy function

To measure the accuracy of the predictions, I defined the function below:

```
# Accuracy function
accuracy <- function(predicted_spam, true_spam){
  confusionMatrix(predicted_spam, true_spam)$overall["Accuracy"]
}
```

3.2 Just guessing

In the most basic model, I simply guessed the outcome with equal probability of predicting whether an email is spam or not.

```
# Just guessing
set.seed(2021, sample.kind = "Rounding")
# Generate guesses for all observations in test_set
pred_guess <- as.factor(sample(c(0,1), nrow(test_set), replace = TRUE))
# Check accuracy
```

```r
accuracy_guess <- accuracy(pred_guess, test_set$spam)
accuracy_guess
```

```
## Accuracy
##    0.503
```

The accuracy of just guessing was about 50%, as expected.

3.3 Naive Bayes

The first machine learning algorithm I used for prediction was the Naive Bayes classifier, which is based on Bayes Theorem, and known to be fast. I split train_set into the outcome ("y") and all other columns which are the predictors ("x").

```r
# Select "spam" column (no. 58) and place into y and place rest of columns into x
y <- train_set[, 58]
x <- train_set[, -58]

# Naive Bayes
# Obtain fit by training using Naive Bayes method
train_nb <- train(x, y, method = "naive_bayes")
# Calculate predictions using Naive Bayes fit
pred_nb <- predict(train_nb, test_set)
# Check accuracy
accuracy_nb <- accuracy(pred_nb, test_set$spam)
accuracy_nb
```

```
## Accuracy
##   0.7189
```

The accuracy for Naive Bayes was obviously a significant improvement over just guessing, but not very high.

3.4 Logistic regression

Next, I looked at logistic regression. The resulting accuracy was a significant improvement over Naive Bayes.

```r
# Logistic regression
# Obtain fit by training using GLM method
train_glm <- train(x, y, method = "glm")
# Calculate predictions using GLMs fit
pred_glm <- predict(train_glm, test_set, type = "raw")
# Check accuracy
accuracy_glm <- accuracy(pred_glm, test_set$spam)
accuracy_glm
```

```
## Accuracy
##    0.924
```

3.5 Support Vector Machine (SVM)

I then used a algorithm called Support Vector Machine (SVM), which tries to find a hyperplane that distinctly classifies data points.

```
# Support Vector Machine (SVM)
# Obtain fit by training using SVM method
train_svm <- svm(spam ~ ., kernel = "linear", data = train_set)
# Calculate predictions using SVM fit
pred_svm <- predict(train_svm, test_set, type = "raw")
# Check accuracy
accuracy_svm <- accuracy(pred_svm, test_set$spam)
accuracy_svm
```

```
## Accuracy
##     0.93
```

There was a slight improvement, and I noticed it was fast.

3.6 K-Nearest Neighbours (KNN)

Next up was the use of the K-Nearest Neighbours (KNN) algorithm.

```
# K-Nearest Neighbours or KNN
set.seed(2021, sample.kind = "Rounding")
# Obtain fit by training using KNN method
train_knn <- train(x, y, method = "knn")
# Calculate predictions using KNN fit
pred_knn <- predict(train_knn, test_set, type = "raw")
# Check accuracy
accuracy_knn <- accuracy(pred_knn, test_set$spam)
accuracy_knn
```

```
## Accuracy
##   0.7732
```

The resulting accuracy was actually quite low. Computation also took some time.

3.7 Random Forest

Finally, I applied the Random Forest algorithm to train_set.

```
# Random forest
set.seed(2021, sample.kind = "Rounding")
# Obtain fit by training using Random Forest (RF) method
train_rf <- train(spam ~ ., method = "rf", data = train_set)
# Calculate predictions using RF fit
pred_rf <- predict(train_rf, test_set, type = "raw")
# Check accuracy
accuracy_rf <- accuracy(pred_rf, test_set$spam)
accuracy_rf
```

```
## Accuracy
##   0.9469
```

Accuracy was good, although computation time was the longest among the algorithms that I used.

4 RESULTS

The various results are tabulated below:

```
##         Method Accuracy
## 1   Just guess   0.5030
## 2  Naive Bayes   0.7189
## 3          GLM   0.9240
## 4          SVM   0.9300
## 5          KNN   0.7732
## 6 RandomForest   0.9469
```

Accuracy is not the only relevant measure for a spam email predictor, however. A model that throws up a high false positive rate is also bothersome as it means a high number of legitimate emails would be flagged as spam and possibly dumped into a spam folder. Hence, I calculated the false positive rates for each method and tabulated the results together with accuracy.

```
# # Calculate false positive rates of the various methods
preds <- list(pred_guess, pred_nb, pred_glm, pred_svm, pred_knn, pred_rf)
false_pos <- sapply(preds, function(pred){
  sum(pred==1 & test_set$spam==0)/nrow(test_set)
})

#Tabulate results with false positive rates
results_fp <- data.frame(Method=c("Just guess", "Naive Bayes", "GLM", "SVM",
                                  "KNN", "RandomForest"),
                         Accuracy=c(accuracy_guess, accuracy_nb, accuracy_glm,
                                    accuracy_svm, accuracy_knn, accuracy_rf),
                         False_positive=c(false_pos[1:6]))
results_fp
```

```
##         Method Accuracy False_positive
## 1   Just guess   0.5030        0.28709
## 2  Naive Bayes   0.7189        0.25935
## 3          GLM   0.9240        0.03257
## 4          SVM   0.9300        0.02774
## 5          KNN   0.7732        0.11460
## 6 RandomForest   0.9469        0.02654
```

Although computation time was longest for the **Random Forest** method, it delivered the highest accuracy and lowest false positive rate. I therefore chose this method to work on the validation set.

```
# Use Random Forest on validation set
pred_val_rf <- predict(train_rf, val_set, type = "raw")
# Check accuracy
accuracy_val_rf <- accuracy(pred_val_rf, val_set$spam)
accuracy_val_rf
```

```
## Accuracy
##   0.9501
```

The resulting accuracy of applying Random Forest to the validation set was 0.95, close to that for test_set.

I also checked that the false positive rate was acceptable.

```
# Check false positive rate
falsepos_val_rf <- sum(pred_val_rf==1 & val_set$spam==0)/nrow(val_set)
falsepos_val_rf
```

```
## [1] 0.01735
```

5 CONCLUSION

The quality of the predictor model obviously depends on the quality of the data and the choice of data attributes. In this case, since the choice of words and characters to include in the dataset is discretionary, there is a risk that the data collectors have chosen the wrong words and characters, or not picked the right ones. Fortunately, the dataset has delivered quite an accurate model, so it appears the attributes have been well chosen.

Given that spam emails are created by people keen to evade spam detectors, such emails might be tweaked over time to render the detector less effective. Thus, data would need to be regularly collected and reviewed for relevance, and the model itself adjusted to thwart spammers.

In terms of method, I decided to use the Random Forest algorithm on the validation set as it had delivered the most accurate result and lowest false positive rate on test_set. One limitation of the Random Forest method is the long computation time, in fact the longest amongst the methods tried. If the dataset was much larger, I might have used the Support Vector Machine (SVM) method which was close to the accuracy to Random Forest and had a low false positive rates, but much faster.