Predictive Modeling in Binary Classification

Steven Futter

2/7/2017

In this assignment I develop a predictive modeling framework for spam email identification using the UCI Machine Learning Repository Spambase data set found here. This classification problem is important for numerous reasons, most notably the loss of time that spam email can waste, the misuse of resources for storing emails that are not important to the receiver, and the invasion of privacy. Before I perform any statistical analysis, let's perform a data quality check and an exploratory data analysis.

(1) Data Quality Check:

Let's start by creating a frequency table for spam and non-spam emails. Below I run the spam email data set through a customized summary function to see how spam ('Yes') emails compare to non-spam emails ('No') over the 57 variables provided in the spam data set. I provide a sample of rows due to the large number of predictors included in the spam data set.

```
                          No 0.05 Yes 0.05 No 0.5 Yes 0.5 No 0.95
char_freq_$                     1        2      5      32      54
char_freq_#                     0        1      0       1       0
capital_run_length_average      0        0      0       0       0
capital_run_length_longest      0        0      0       0       0
capital_run_length_total        0        0      0       0       0
SPAM                            0        0      0       0       0
                          Yes 0.95 No Min Yes Min  No Max  Yes Max
char_freq_$                 194.00 665.00 1596.40 5902.00 15841.00
char_freq_#                   1.00   0.00    1.00    0.00     1.00
capital_run_length_average    0.00   0.19    0.10    4.38     1.12
capital_run_length_longest    0.33   0.44    1.55   32.48     7.84
capital_run_length_total      0.08   0.05    0.67    2.04     6.00
SPAM                          0.00   0.07    0.24    7.41    19.83
                          No Mean Yes Mean No Variance Yes Variance
char_freq_$                161.47   470.62   126549.81    680758.95
char_freq_#                  0.00     1.00        0.00         0.00
capital_run_length_average   0.05     0.02        0.09         0.01
capital_run_length_longest   0.11     0.51        0.67         0.55
capital_run_length_total     0.01     0.17        0.00         0.13
SPAM                         0.02     0.08        0.06         0.37
                          No % Missing Yes % Missing
char_freq_$                          0             0
char_freq_#                          0             0
capital_run_length_average           0             0
capital_run_length_longest           0             0
capital_run_length_total             0             0
SPAM                                 0             0
```

Missing Data

The my.summary() function above confirms that there are no missing observations in either the spam or non-spam email observations.

Data Ranges and Distributions

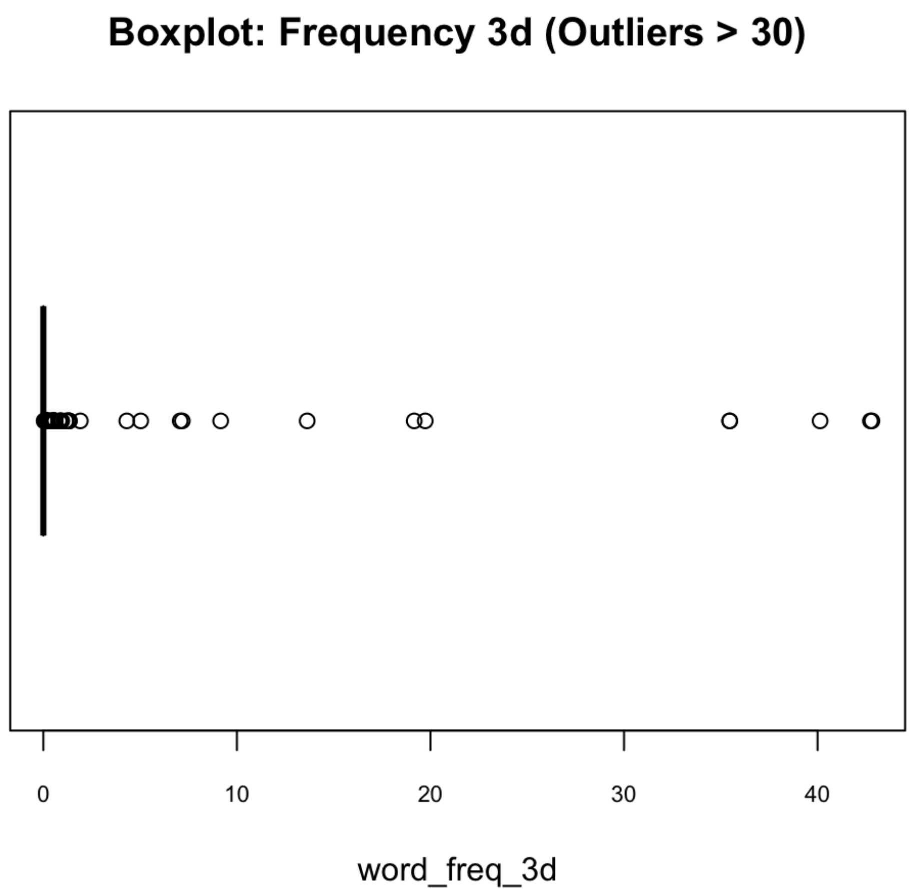
The my.summary() function output also provides insight into the distribution of various different word and character frequencies, as well as the capital letter run length average, run length longest, and run length total. There are 4601 observations and 58 variables in the spam dataset of which 61% (2788 out of 4601) are non-spam emails and 39% (1813 out of 4601) are spam.

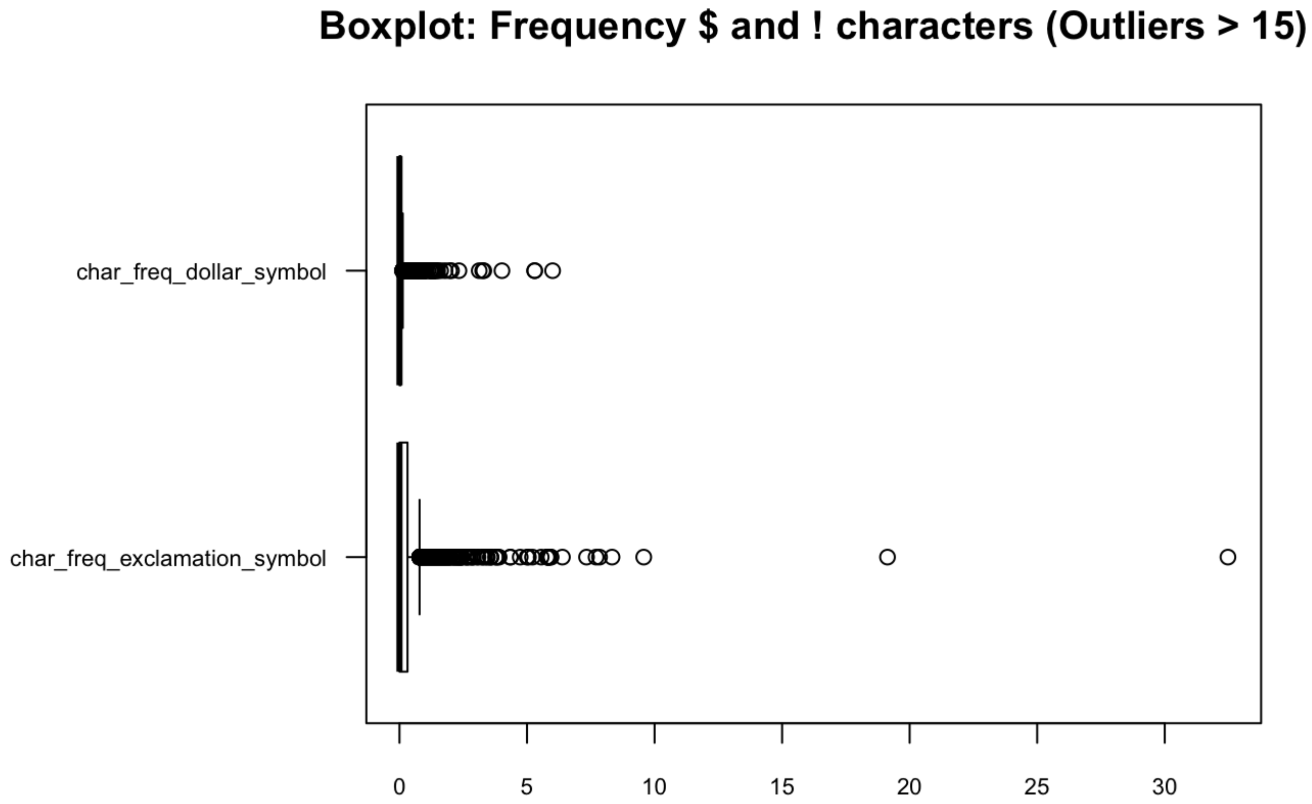
Differences across spam and non-spam observations

The ‘Capital Run Length Average’ measures the average length of uninterrupted sequences of capital letters. Spam emails often need to attract the readers attention so this may not come as a surprise. The mean for this variable differs from 2.38 for non-spam to 9.52 for spam emails. The non-spam minimum average length of uninterupted sequences of capital letters is 4.68 in comparison to the spam minimum average length of uninterupted sequences of capital which is 15.68. Again, for the maximum number of non-spam and spam uninterrupted sequences of capital letters with 251.00 and 1102.50 letters respectively. It is evident that the spam and no-spam observations differ considerably by their use of capital letters. These variables look to be promising predictors, but let’s investigate further.

Outliers

By using a boxplot approach I identify several extreme outliers for the variables word_freq_3d, capital_run_length_total, and capital_run_length_longest. An outlier is defined as a data point that is located outside the fences, or whiskers, of the boxplot (e.g: outside 1.5 times the interquartile range above the upper quartile and below the lower quartile). However, for the purpose of the initial quality check I refer to only the extreme values that stand out amongst the observations. For word_freq_3d the extreme values can be seen with values greater than 30, and for char_freq_dollar_symbol and char_freq_exclamation_symbol for values greater than 15. Other extreme outliers are found in the dataset, notably with the capital_run_length_total and capital_run_length_total variables where values exceed 5,000. I review these in the EDA section below.

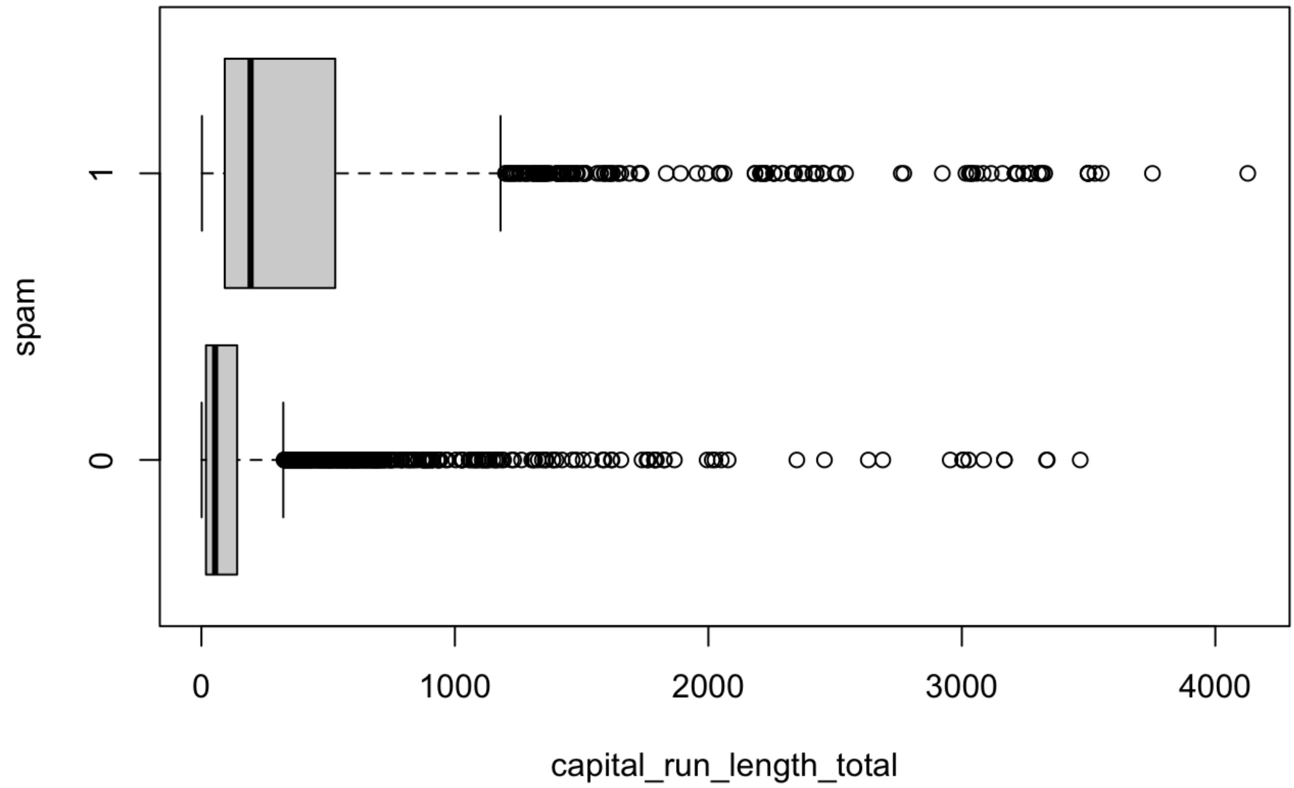


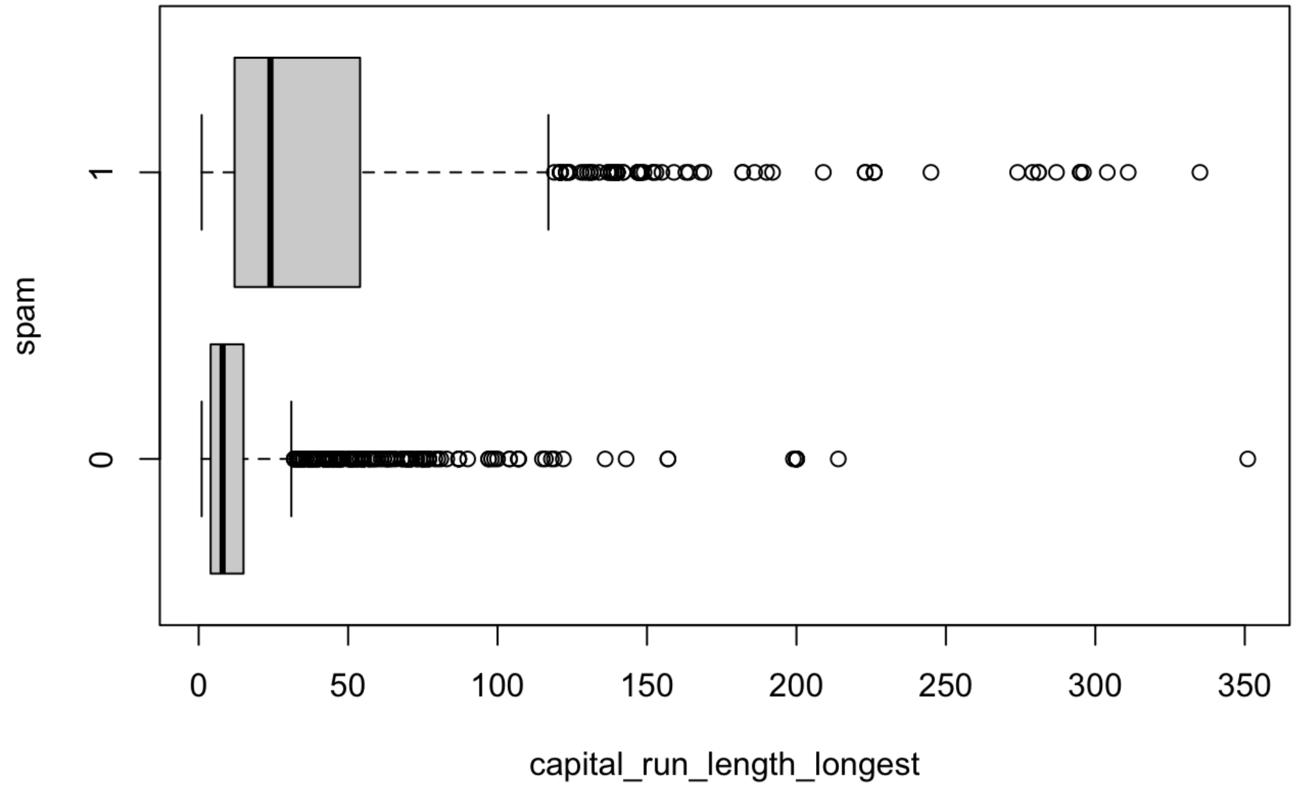


(2) Exploratory Data Analysis:

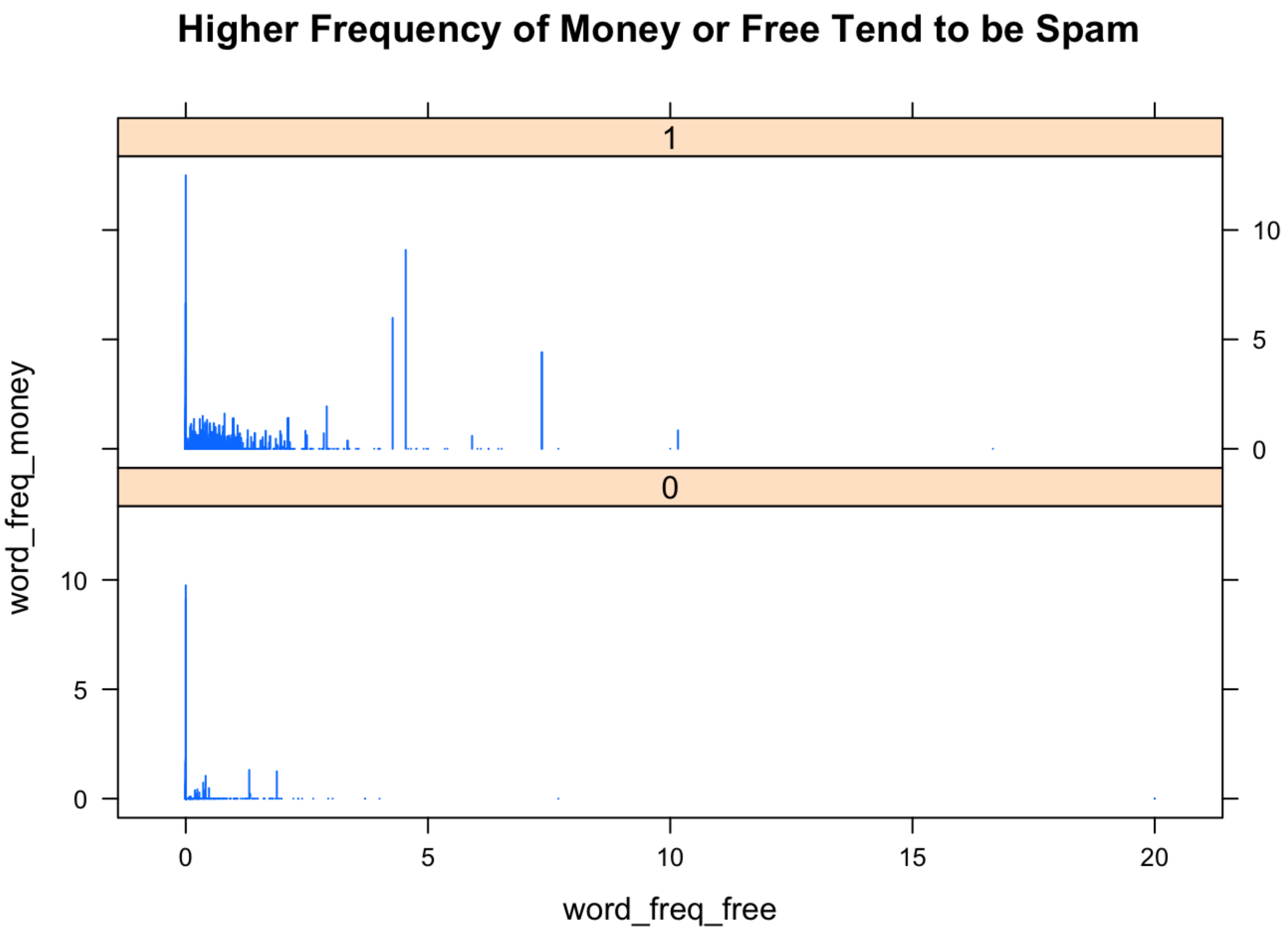
With the spam data set we have a binary classification problem: either an observation is spam, or it is not. After temporarily removing the extreme outliers with values in excess of 5,000 we can see that capital_run_length_total is higher for the middle 75% (grey shaded bar) of values for spam emails. There is some overlap with non-spam emails for the capital_run_length_total variable, however, the whiskers of the boxplot for spam emails is far higher (i.e. extends further to the right in the plot below). The same finding can be made by looking at the capital_run_length_total variable in that the middle 75% (grey shaded bar) of values for spam emails is higher and the upper whisker extends far higher for spam in comparison to non-spam emails. I exclude values here that are in excess of 500 to make this distinction clearer.

To get the readers attention senders of spam email will use capital letters. On average spam emails contain a proportionately larger percentage of capital letters. As below we can see that there is a marked difference between spam and non-spam emails for the two variables.



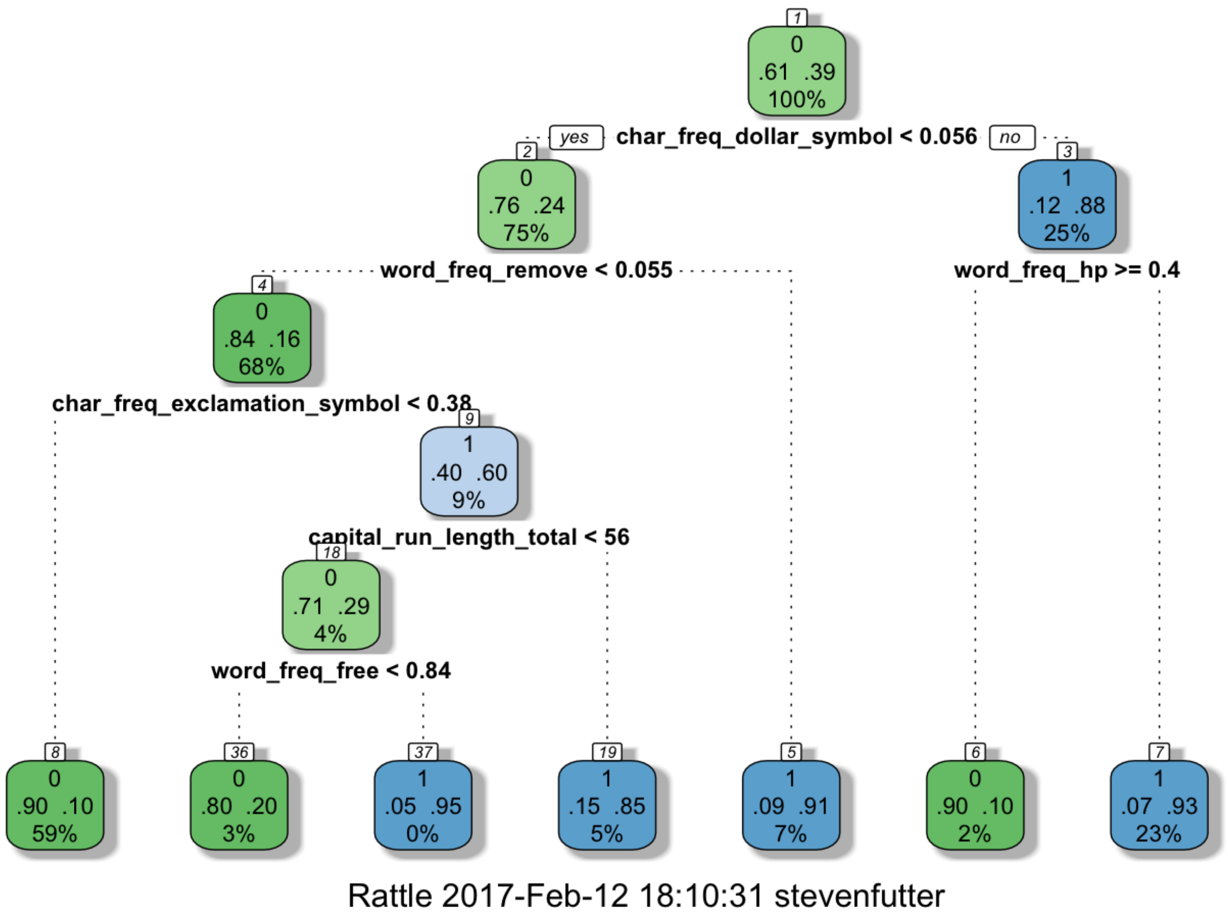


Additionally, emails received with higher frequencies of the words ‘free’ and ‘money’ are more likely to be spam. This can be seen below by the xyplot from the lattice() library below. The frequency of the words money or free are higher for spam emails. See the top section of the plot below (Spam=1) in comparison to the bottom section (Spam=0).



I now use a tree model to explore the data further and evaluate missclassification error.

Tree Model



At the root of the tree the 0.39 denotes that 39% of the 4601 observations are spam emails. The root is split at char_freq_$<0.056 and this highlights that the frequency with which the $ sign appears in the email is the most predictive factor in determining spam emails. Of the 4601 observations 25% have a char_freq_$ greater than 0.056. Of these observations 88% are spam. Other variables that appear in the top five most important predictive variables from looking at summary(tree) of the tree are: char_freq_$, word_freq_remove, word_freq_money,word_freq_000, and char_freq_!.

(3) The Model Build

Let's begin the model building process by splitting the spam data set into a training and test data set.

1. **Logistic Regression Model using Variable Selection**

For this first model I use the glm() library to create a forward, backward, and stepwise variable selection logistic regression model. The AIC value for each model produced is AIC=1346.28. I present the R code for all models in the appendix and provide the formula from the the logistic model created via the stepwise variable selection algorithm below.

Final Model Selected by the stepwise variable selection algorithm has an AIC of 1346.28.

SPAM ~ char_freq_dollar_symbol + word_freq_hp + capital_run_length_longest + word_freq_george + word_freq_free + word_freq_edu + word_freq_remove + word_freq_business + word_freq_meeting + word_freq_000 + word_freq_money + word_freq_cs + word_freq_internet + word_freq_re + char_freq_exclamation_symbol + word_freq_your + word_freq_conference + word_freq_credit + word_freq_our + word_freq_project + word_freq_order + word_freq_original + word_freq_technology + word_freq_over + word_freq_650 + word_freq_pm + word_freq_you + word_freq_lab + word_freq_85 + word_freq_3d + word_freq_address + word_freq_data + word_freq_will + capital_run_length_total + char_freq_semi_colon_symbol + char_freq_number_symbol + word_freq_addresses + word_freq_hpl + word_freq_make + word_freq_table + word_freq_direct

2. **Tree Model**

The tree is constructed with a reduced subset of eight variables producing 13 terminal nodes. In the context of a regression tree, the deviance (Residual mean deviance = 0.5034) is simply the sum of squared error for the tree. The misclassification error is 8.2%. The variables included in the tree are: “char_freq_dollar_symbol”, “char_freq_exclamation_symbol”, “word_freq_remove”, “word_freq_hp”, “capital_run_length_longest”, “word_freq_our”, “word_freq_george”, “word_freq_free”, and “word_freq_edu”.

```
Classification tree:
tree(formula = SPAM ~ word_freq_make + word_freq_address + word_freq_3d +
    word_freq_our + word_freq_over + word_freq_remove + word_freq_internet +
    word_freq_order + word_freq_will + word_freq_addresses +
    word_freq_free + word_freq_business + word_freq_you + word_freq_credit +
    word_freq_your + word_freq_000 + word_freq_money + word_freq_hp +
    word_freq_hpl + word_freq_george + word_freq_650 + word_freq_lab +
    word_freq_data + word_freq_85 + word_freq_technology + word_freq_pm +
    word_freq_direct + word_freq_cs + word_freq_meeting + word_freq_original +
    word_freq_project + word_freq_re + word_freq_edu + word_freq_table +
    word_freq_conference + char_freq_semi_colon_symbol + char_freq_exclamation_symb
ol +
    char_freq_dollar_symbol + char_freq_number_symbol + capital_run_length_longest
+
    capital_run_length_total, data = spam.train)
Variables actually used in tree construction:
[1] "char_freq_dollar_symbol"      "char_freq_exclamation_symbol"
[3] "word_freq_remove"             "word_freq_hp"
[5] "capital_run_length_longest"   "word_freq_our"
[7] "word_freq_free"               "word_freq_edu"
Number of terminal nodes:  13
Residual mean deviance:  0.5034 = 1615 / 3207
Misclassification error rate: 0.08199 = 264 / 3220
```

3. **Support Vector Machine Model**

Create SVM Model and associated confusion matrix. I revert back to the model in the model comparison section below.

```
Call:
svm.default(x = x.train, y = y.train, data = spam.train)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.01754386

Number of Support Vectors:  964

 ( 453 511 )


Number of Classes:  2

Levels:
 0 1
```

4. **Random Forest Model**

```
Call:
 randomForest(formula = SPAM ~ word_freq_make + word_freq_address +      word_freq_3d + word_freq_our + word_freq_over + word_freq_remove +      word_freq_internet + word_freq_order + word_freq_will + word_freq_addresses +      word_freq_free + word_freq_business + word_freq_you + word_freq_credit +      word_freq_your + word_freq_000 + word_freq_money + word_freq_hp +      word_freq_hpl + word_freq_george + word_freq_650 + word_freq_lab +      word_freq_data + word_freq_85 + word_freq_technology + word_freq_pm +      word_freq_direct + word_freq_cs + word_freq_meeting + word_freq_original +      word_freq_project + word_freq_re + word_freq_edu + word_freq_table +      word_freq_conference + char_freq_semi_colon_symbol + char_freq_exclamation_symbol +      char_freq_dollar_symbol + char_freq_number_symbol + capital_run_length_longest +      capital_run_length_total, data = spam.train, mtry = 3, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 5.12%
Confusion matrix:
     0    1 class.error
0 1895   56  0.02870323
1  109 1160  0.08589441
```

(4) Model Comparison

In this next section I compare each model performance by its predictive accuracy performance. By providing each model's misclassification error we can see how many spam emails each model incorrectly classified. I provide the training and test data set errors below so that we can see how each model performs in- and out-of-sample.

Results

```
                          model train.error test.error  diff.error
1 Logistic Regression (Stepwise)  0.07080745 0.07241130 0.001603843
2                     Tree Model  0.08198758 0.09630702 0.014319446
3         Support Vector Machine  0.05310559 0.06951484 0.016409254
4                  Random Forest  0.01428571 0.05430847 0.040022758
```

The Random Forest model performs the best in terms of accuracy with an misclassification training error rate of 1.4% and testing error rate of 5.4%. The tree model performed worse than the others with a training and test error rate of 8.2% and 9.6% respectively. The SVM model had the second lowest error rate with a training and test error rate of 5.3% and 7.0% respectively. The logistic regression model performed the third best in terms of accuracy. Of all the models the Random Forest model saw the widest difference in errors between the training and test data sets. This is likely to be due to some overfitting of the random forest model to the training data. To further evaluate the model we can run additional out-of-sample tests for each model to see which model performs consistently better.

(5) APPENDIX for R Code

```
#### 1. Data Quality Check: Custom function to calculate min, 0.05, median, 0.95, m
ax, % missing for each variable
maxfun             <- function(x){round(max(x,na.rm = TRUE),2)}
minfun             <- function(x){round(min(x,na.rm = TRUE),2)}
quantilefun        <- function(x){round(quantile(x,c(0.01,0.05,0.25,0.5,0.75,0.95,0
.99)),2)}
meanfun            <- function(x){round(mean(x),2)}
varfun             <- function(x){round(var(x),2)}
percentMissingfun <- function(x){round(100*(sum(is.na(x))/length(x)),2)}

my.summary = function(df){
  max = colwise(maxfun)(df)                            #create rows of summarized valu
es using colwise function from plyr package
  min = colwise(minfun)(df)
  quantile = colwise(quantilefun)(df)
  mean = colwise(meanfun)(df)
  variance = colwise(varfun)(df)
  percentMissing = colwise(percentMissingfun)(df)

  # bind data.frame and rename rows
  mydf = rbind(min, quantile, max, mean, variance, percentMissing)
  rownames(mydf)  = c('min',0.01,0.05,0.25,0.5,0.75,0.95,0.99,'max','mean','variance
','% missing')
  return(t(mydf)) # take transpose
}

#### Create new temp data frame to hold quantile and missing values
# split out spam data set into spam and non-spam data.frames
yes.spam = subset(spam, SPAM=='1')
no.spam  = subset(spam, SPAM=='0')

# create data frame of spam and non-spam
spam.summary = my.summary(spam)
spam.summary = data.frame(spam.summary)

yes.spam.summary = my.summary(yes.spam)
yes.spam.summary = data.frame(yes.spam.summary)

no.spam.summary = my.summary(no.spam)
```

```
no.spam.summary = my.summary(no.spam)
no.spam.summary = data.frame(no.spam.summary)

df1 = cbind(no.spam.summary$min,      yes.spam.summary$min,
            no.spam.summary$X0.05,    yes.spam.summary$X0.05,
            no.spam.summary$X0.5,     yes.spam.summary$X0.5,
            no.spam.summary$X0.95,    yes.spam.summary$X0.95,
            no.spam.summary$max,      yes.spam.summary$max,
            no.spam.summary$mean,     yes.spam.summary$mean,
            no.spam.summary$variance, yes.spam.summary$variance,
            no.spam.summary$X..missing, yes.spam.summary$X..missing)

df1 = data.frame(df1)
colnames(df1) = c('No 0.05','Yes 0.05',
                  'No 0.5','Yes 0.5',
                  'No 0.95','Yes 0.95',
                  'No Min','Yes Min',
                  'No Max','Yes Max',
                  'No Mean','Yes Mean',
                  'No Variance','Yes Variance',
                  'No % Missing','Yes % Missing')
rownames(df1) = column.names
df1[53:58,]


#### Outliers
# Resize the borders so that the long variable names will fit neatly into the boxpl
ot outputs
op <- par(mar = c(5, 10, 4, 2) + 0.1)
boxplot(spam[,c('word_freq_3d')], data = spam, horizontal = TRUE, las = 1, cex.axi
s = 0.7, xlab='Frequency',main='Boxplot: Frequency 3d')
par(op)                                # reset the plotting parameters

# Example of outliers for the capital_run_length_total and capital_run_length_longe
st variables. Example outliers with values > 5,000.
op <- par(mar = c(5, 10, 4, 2) + 0.1)
boxplot(spam[,56:57], data = spam, horizontal = TRUE, las = 1, cex.axis = 0.7, xla
b='Frequency',main='Boxplot: Frequency $ and ! characters')
par(op)                                # reset the plotting parameters




#### 2. Exploratory Data Analysis
# Boxplots usng dplyr to filter out extreme outliers
temp = spam %>% filter(capital_run_length_total<5000)
boxplot(capital_run_length_total~SPAM,   data=temp, col='lightgray',xlab="capital_r
un_length_total", ylab='spam',horizontal=TRUE)

temp = spam %>% filter(capital_run_length_total<500)
boxplot(capital_run_length_longest~SPAM, data=temp, col='lightgray',xlab="capital_r
un_length_longest", ylab='spam',horizontal=TRUE)

# xyplot to show how words free and money effect the likelihood that an email is sp
am
xyplot(word_freq_money ~ word_freq_free  | factor(SPAM), data = spam, type = "h",
layout = c(1, 2), xlab = "word_freq_free", main='Higher Frequency of Money or Free
Tend to be Spam')
```

```
tend to be spam")

# Plot a more reasonable tree model for EDA purposes
form <- as.formula(SPAM ~ .)
tree <- rpart(form,spam)             # A more reasonable tree
fancyRpartPlot(tree)                 # A fancy plot from rattle





#### 3. Model Build

# Split data between training and testing dataset
smp.size = floor(0.7 * nrow(spam))
set.seed(1)
train = sample(seq_len(nrow(spam)), size = smp.size)
test = -train
spam.train = spam[train,]
spam.test  = spam[-train,]


### MODEL 1 Forward:
fullmod = glm(SPAM ~ .,data=spam.train, family=binomial)
summary(fullmod)

nothing <- glm(SPAM ~ 1,data=spam.train, family=binomial)
summary(nothing)

### MODEL 1 Forward
forwards = step(nothing,scope=list(lower=formula(nothing),upper=formula(fullmod)),
direction="forward")
formula(forwards)

### MODEL 1 Backward
backwards = step(fullmod) # Backwards selection is the default
formula(backwards)

### MODEL 1 Stepwise
bothways = step(nothing, list(lower=formula(nothing),upper=formula(fullmod)),direc
tion="both")
formula(bothways)


### MODEL 2 Tree
set.seed(1)
tree.fit=tree(SPAM~
    word_freq_make + word_freq_address + word_freq_3d + word_freq_our +
    word_freq_over + word_freq_remove + word_freq_internet +
    word_freq_order + word_freq_will + word_freq_addresses +
    word_freq_free + word_freq_business + word_freq_you + word_freq_credit +
    word_freq_your + word_freq_000 + word_freq_money + word_freq_hp +
    word_freq_hpl + word_freq_george + word_freq_650 + word_freq_lab +
    word_freq_data + word_freq_85 + word_freq_technology + word_freq_pm +
    word_freq_direct + word_freq_cs + word_freq_meeting + word_freq_original +
    word_freq_project + word_freq_re + word_freq_edu + word_freq_table +
    word_freq_conference + char_freq_semi_colon_symbol + char_freq_exclamation_symb
```

```
ol + char_freq_dollar_symbol +
    char_freq_number_symbol + capital_run_length_longest + capital_run_length_total
    ,data=spam.train)
summary(tree.fit)


### MODEL 3 SVM
x.train = subset(spam.train, select=-SPAM)   # Divide spam data to x (the variable
s) and y the classes
y.train = spam.train$SPAM

x.test = subset(spam.test, select=-SPAM)
y.test = spam.test$SPAM

svm.fit = svm(x.train,y.train,data=spam.train)    # Create SVM model
summary(svm.fit)


### MODEL 4 Random Forest
set.seed(1)
randomForest.fit = randomForest(SPAM~
    word_freq_make + word_freq_address + word_freq_3d + word_freq_our +
    word_freq_over + word_freq_remove + word_freq_internet +
    word_freq_order + word_freq_will + word_freq_addresses +
    word_freq_free + word_freq_business + word_freq_you + word_freq_credit +
    word_freq_your + word_freq_000 + word_freq_money + word_freq_hp +
    word_freq_hpl + word_freq_george + word_freq_650 + word_freq_lab +
    word_freq_data + word_freq_85 + word_freq_technology + word_freq_pm +
    word_freq_direct + word_freq_cs + word_freq_meeting + word_freq_original +
    word_freq_project + word_freq_re + word_freq_edu + word_freq_table +
    word_freq_conference + char_freq_semi_colon_symbol + char_freq_exclamation_symb
ol + char_freq_dollar_symbol +
    char_freq_number_symbol + capital_run_length_longest + capital_run_length_total
,
    data=spam.train, mtry=3, importance=TRUE)
randomForest.fit



#### 4. Model Comparison
###### 1. Stepwise Logistic Regression Model
# Training accuracy
glm.probs = predict(glm.stepwise.fit, newdata=spam.train, type='response')
glm.pred = ifelse(glm.probs > 0.5, '1','0')
table(glm.pred, spam.train$SPAM)
glm.train.error = 1 - mean(glm.pred==spam.train$SPAM)

# Testing accuracy
glm.probs = predict(glm.stepwise.fit, newdata=spam.test, type='response')
glm.pred = ifelse(glm.probs > 0.5, '1','0')
table(glm.pred, spam.test$SPAM)
glm.test.error = 1 - mean(glm.pred==spam.test$SPAM)


###### 2. Tree Model
# Training accuracy
tree.pred=predict(tree.fit, spam.train, type='class')
```

```r
table(tree.pred, spam.train$SPAM)
tree.train.error = 1 - mean(tree.pred==spam.train$SPAM)

# Testing accuracy
tree.pred=predict(tree.fit, spam.test, type='class')
table(tree.pred, spam.test$SPAM)
tree.test.error = 1 - mean(tree.pred==spam.test$SPAM)


###### 3. Support Vector Machine Model
# Training accuracy
svm.pred=predict(svm.fit, x.train)
table(svm.pred, y.train)
svm.train.error = 1 - mean(svm.pred==y.train)

# Testing accuracy
svm.pred=predict(svm.fit, x.test)
table(svm.pred, y.test)
svm.test.error = 1 - mean(svm.pred==y.test)


##### 4. Random Forest Model
# Training Accuracy
randomForest.pred = predict(randomForest.fit, spam.train)
table(randomForest.pred, spam.train$SPAM)
randomForest.train.error = 1 - mean(randomForest.pred==spam.train$SPAM)

# Testing Accuracy
randomForest.pred = predict(randomForest.fit, spam.test)
table(randomForest.pred, spam.test$SPAM)
randomForest.test.error = 1 - mean(randomForest.pred==spam.test$SPAM)

# Create results table
model = c('Logistic Regression (Stepwise)', 'Tree Model', 'Support Vector Machine', 
'Random Forest')
train.error = c(glm.train.error, tree.train.error, svm.train.error, randomForest.tr
ain.error)
test.error  = c(glm.test.error, tree.test.error, svm.test.error, randomForest.test.
error)
diff.error  = c(glm.test.error-glm.train.error,
                tree.test.error-tree.train.error,
                svm.test.error-svm.train.error,
                randomForest.test.error-randomForest.train.error)
results = data.frame(model, train.error, test.error, diff.error)
results
```