Case Study 10 – Email Classification (Spam or Ham)

Hieu Nguyen, Nithya Devadoss, Ramesh Simhambhatla, Ramya Mandava

Oct 07, 2018

Abstract

Email spam, also known as junk email, is unsolicited messages sent in bulk by email (spamming). Most email spam messages are commercial in nature. Whether commercial or not, many contain disguised links that appear to be for familiar websites but in fact lead to phishing web sites or sites that are hosting malware [1]. In this case study, we examine over 9000 messages that have been classified by SpamAssassin (http://spamassasin.apache.org) for the purpose of developing and testing spam filters. We would be using CART aka classification and regression trees to classify an email message as spam or ham (not spam).

1 Introduction

As humans we spot a spam message easily by looking at the subject and sender information. Many a times we fail to miss a spam leading to phishing and hacking. This process could be automated using some statistical methods. Spam filters used by mail readers examine various characteristics of an email before deciding whether to place it in your inbox or spam folder.

The objective of the case study is to classify and predict whether a given email is spam or not spam based on the features derived from an email. It is known from industry standards that the header information from the emails is often enough to determine whether a message is spam or ham. By using text mining techniques, we try to uncover features such as 'The subject line having all letters capitalized', or 'whether or not the subject line begins Re:' or the quantity of exclamation marks in the body of the message. These characteristics will further be quantified to differentiate between spam and ham. Our goal is to create a classifier based on these characteristics.

Decision tree model has been chosen for this classification purpose, that is trained through recursive partitioning process using 'rpart' package in R and further Hyper-parameter optimization is performed.

The data for this case study is obtained from SpamAssassin website (http://www.rdatasciencecases.org/Spam/) and has 5 directories – 3 of the folders contains Ham (non spam) and 2 folders contain Spam individual messages (totaling about 9000 messages).

Our approach in this case study is:

1) Capture Data from SpamAssasin website
2) Examine Data and perform Data Cleaning using R Text Mining packages

3) Feature extraction and create a derived data frame
4) Analyze Data
5) Create and evaluate the model
6) Compare the performance by tuning the parameters

2 Data Acquisition, Cleaning, Exploratory Analysis and Model Fitting

2.1 Data Acquisition:

We have acquired the data from the spamassasins website http://spamassasin.apache.org. It is saved as 5 folders – easyham , easyham2 , hard_ham , spam , spam_2 as shown below(Code output) :

```
> spamPath = "C:/Nithya/MSDS/Term5/QTW/CaseStudies/CS10/SpamAssassinMessages"
> list.dirs(spamPath, full.names = FALSE)
[1] ""           "easy_ham"   "easy_ham_2" "hard_ham"   "spam"       "spam_2"
```

Each folder has corresponding email messages which are either spam or non-spam. Each of this email message is split into head and body of the message for analysis using various functions.

2.2. Create Tidy Data:

We have extracted the words in the message by removing the stop words so we use the words in the email for analysis. The text is cleaned from punctuations, blanks and stop words and stored in the list for the different email content types.
The frequency is calculated and the different features are created for creating the model. There are various functions which is used on the header and body of the emails to extract the features like finding the capital letters, is the message a reply, is the subject empty, are there words which is a part of the spam Check words list, for example – “Viagra”, “pounds”, “free”, “million” and so on.
As a future work, we can create a separate file to update these words from time to time for reproducibility.
The final derived dataframe - emailDF has about 9348 observations and 30 extracted features as shown in the figure 2.3.1. Among these 30 variables, isSpam is our response variable.

Document# 7501(email) had to be removed from the final data structure to be able to extract features as this email has been corrupt.

Derived Features

```
glimpse(emailDF)

Observations: 9,348
Variables: 30
$ isSpam       <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ isRe         <lgl> TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, TRUE, F...
$ numLines     <int> 50, 26, 38, 32, 31, 25, 38, 39, 126, 50, 19, 20, 27, ...
$ bodyCharCt   <int> 1554, 873, 1713, 1095, 1021, 718, 1288, 1182, 5989, 1...
$ underscore   <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ subExcCt     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ subQuesCt    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0,...
$ numAtt       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,...
$ priority     <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ numRec       <int> 2, 1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1,...
$ perCaps      <dbl> 4.451039, 7.491289, 7.436096, 5.090909, 6.116643, 7.6...
$ isInReplyTo  <lgl> TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE,...
$ sortedRec    <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,...
$ subPunc      <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ hour         <dbl> 11, 11, 12, 13, 13, 13, 13, 14, 14, 11, 14, 14, 15, 1...
$ multipartText <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ hasImages    <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ isPGPsigned  <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ perHTML      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ subSpamWords <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ subBlanks    <dbl> 12.50000, 8.00000, 8.00000, 18.91892, 15.21739, 15.21...
$ noHost       <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ numEnd       <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ isYelling    <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ forwards     <dbl> 0.000000, 0.000000, 0.000000, 3.125000, 6.451613, 12....
$ isOrigMsg    <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ isDear       <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
$ isWrote      <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE...
$ avgWordLen   <dbl> 4.376623, 4.555556, 4.817164, 4.714286, 4.234940, 3.9...
$ numDlr       <int> 3, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0,...
```

Feature Explanation

Variable	Type	Definition
isRe	logical	TRUE if Re: appears at the start of the subject.
numLines	integer	Number of lines in the body of the message.
bodyCharCt	integer	Number of characters in the body of the message.
underscore	logical	TRUE if email address in the From field of the header contains an underscore.
subExcCt	integer	Number of exclamation marks in the subject.
subQuesCt	integer	Number of question marks in the subject.
numAtt	integer	Number of attachments in the message.
priority	logical	TRUE if a Priority key is present in the header.
numRec	numeric	Number of recipients of the message, including CCs.
perCaps	numeric	Percentage of capitals among all letters in the message body, excluding attachments.
isInReplyTo	logical	TRUE if the In-Reply-To key is present in the header.
sortedRec	logical	TRUE if the recipients' email addresses are sorted.
subPunc	logical	TRUE if words in the subject have punctuation or numbers embedded in them, e.g., w!se.
hour	numeric	Hour of the day in the Date field.
multipartText	logical	TRUE if the MIME type is multipart/text.
hasImages	logical	TRUE if the message contains images.
isPGPsigned	logical	TRUE if the message contains a PGP signature.
perHTML	numeric	Percentage of characters in *HTML* tags in the message body in comparison to all characters.
subSpamWords	logical	TRUE if the subject contains one of the words in a spam word vector.
subBlanks	numeric	Percentage of blanks in the subject.
noHost	logical	TRUE if there is no hostname in the Message-Id key in the header.
numEnd	logical	TRUE if the email sender's address (before the @) ends in a number.
isYelling	logical	TRUE if the subject is all capital letters.
forwards	numeric	Number of forward symbols in a line of the body, e.g., >>> xxx contains 3 forwards.
isOrigMsg	logical	TRUE if the message body contains the phrase original message.
isDear	logical	TRUE if the message body contains the word dear.
isWrote	logical	TRUE if the message contains the phrase wrote:.
avgWordLen	numeric	The average length of the words in a message.
numDlr	numeric	Number of dollar signs in the message body.

Figure 2.3.1 List of the extracted features

2.3. Data Exploration

Some common traits of spam emails include using all capitals, hash busting to break one word or phrase, adding many pictures, some attachments with executables, html links to phishing sites. In order to get a clear idea and use appropriate features for creating the model, we have to analyze the training data set. We have analyzed the derived data set with various plots like scatter plots, qq plots, box plots.

Scatter plot of number of Lines Vs number of characters:

As seen in figure 2.3.2, We see that there is a linear relationship between the number of characters and the number of lines of messages in the email body. The minimum number of

characters in a line is about around 100 from the below plot. The linear relationship indicates that our function is functioning well in extracting word vectors.

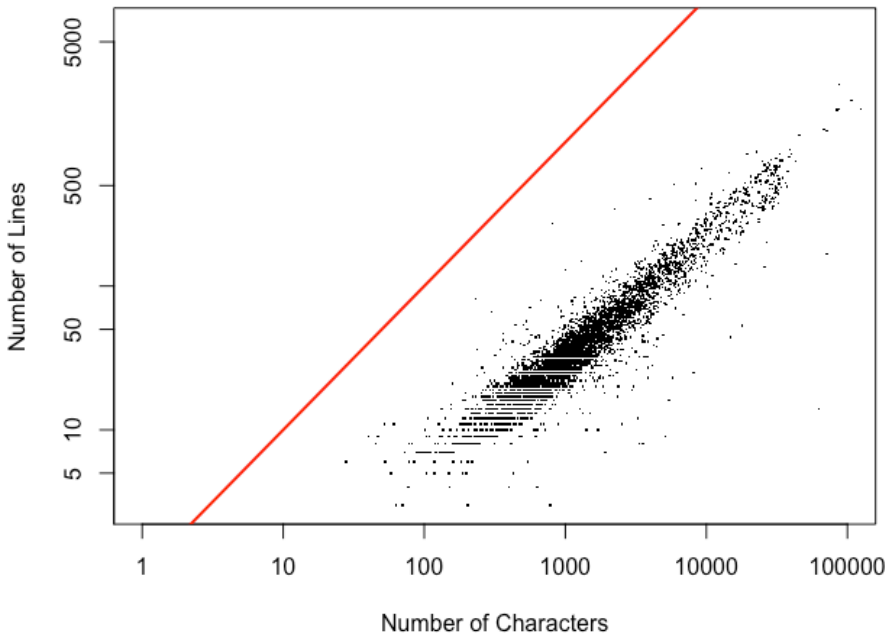

Figure 2.3.2 Num of Lines Vs Num of characters in body of the message

Box Plot:

Let's examine the percentage of capitals in the message body, comparing this percentage between spam and ham messages. Side-by-side boxplots in Figure 2.3.3 help us compare the distributions of these two groups.

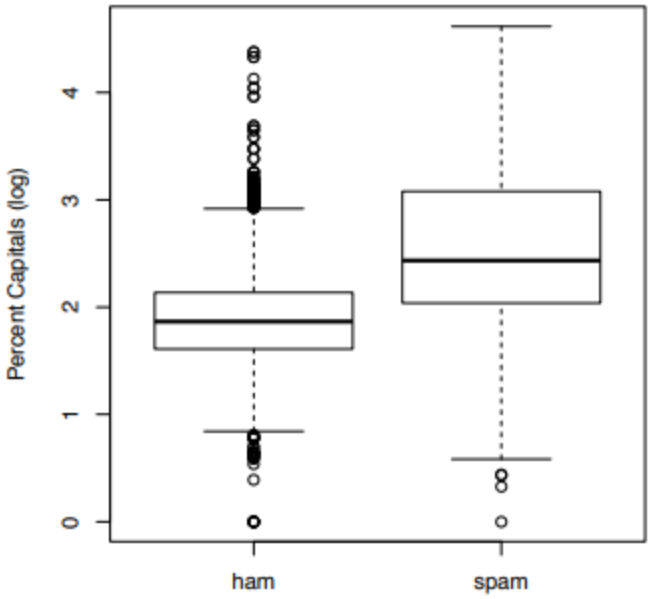

Figure 2.3.3 Box plots comparing usage of capitals in emails

We see from the boxplots that about 75% of the regular email have values below the lower quartile for spam. This variable may be useful for classification. [2]

QQ Plot:

From the QQ plot below in Figure 2.3.4, we can clearly see that the distribution for the percentage of capital letters in the email have different spread for spam and ham messages. A slope other than 1 indicates the distributions have different spreads, and an intercept other than 0 indicates a shift in the mean of the distributions.[2]

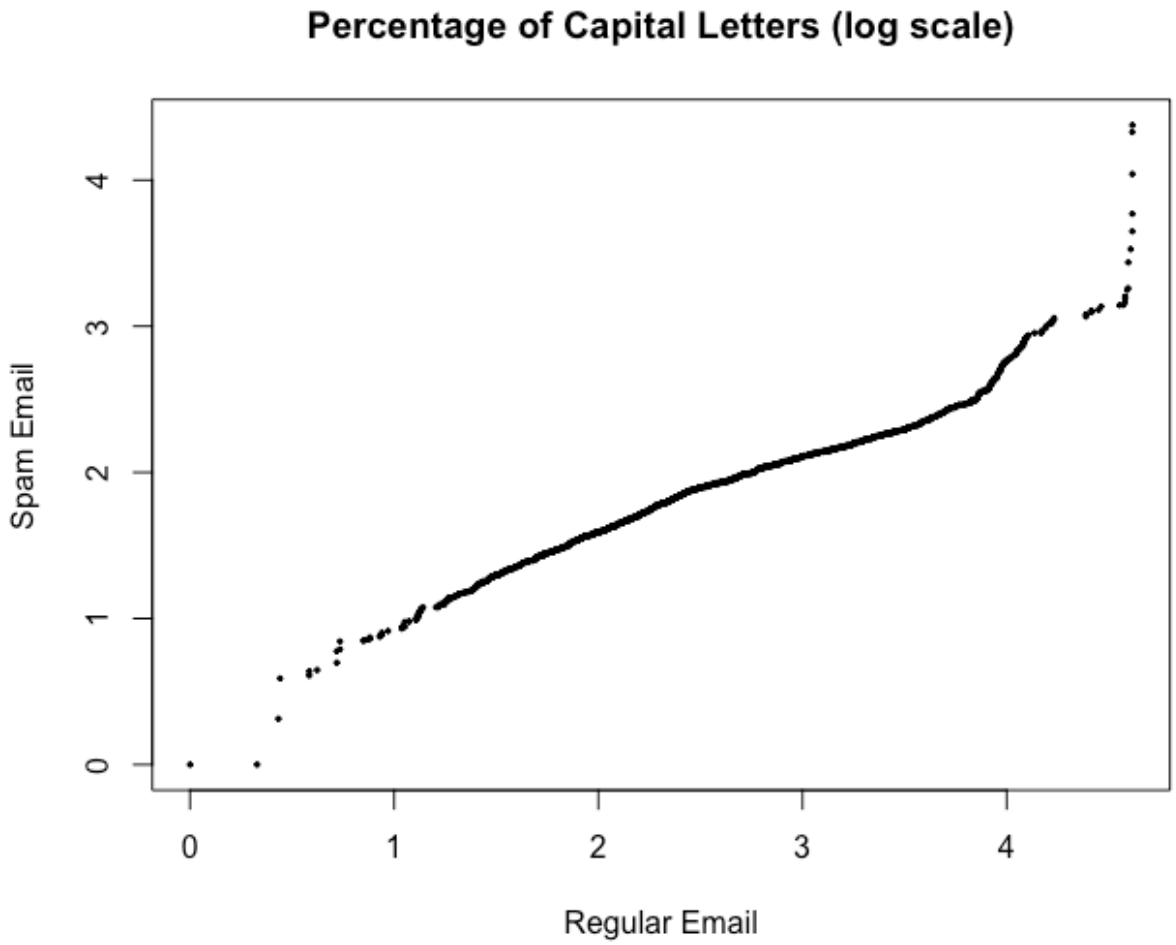

Figure 2.3.4 QQ Plot of the capital letters Spam Vs Regular.

Scatter plot:

Scatter plot examines the relationship of the percentage of capital letters in a message and the total number of characters in the message. Spam is marked by purple dots and ham by green. The darker color indicates over plotting. We see here that the spam tends to be longer and have more capital letters than ham.

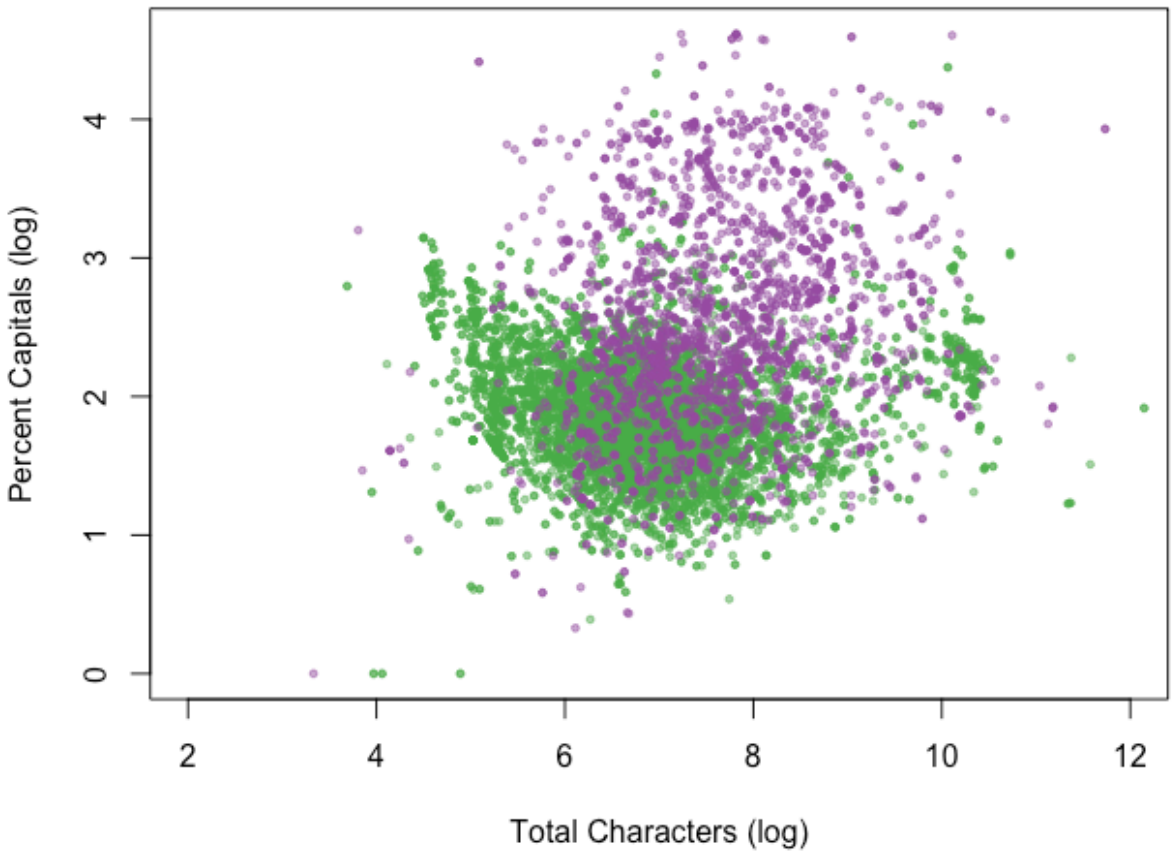

Figure 2.3.5 Scatter plot comparing Cap Amount and Size of the message

Table:

The table below as in figure 2.3.6 indicates very little difference as majority of the messages have no attachments.

```
> table(emailDF$numAtt, isSpamLabs)
    isSpamLabs
      ham spam
  0  6624 2157
  1   314  230
  2    11    6
  4     0    1
  5     1    2
  18    1    0
```

Figure 2.3.6 Table indicating number of attachments against being Spam Vs Ham label

Mosaic Plots:

The two mosaic plots as shown in Figure 2.3.7 use area to denote the proportion of messages that fall in each category – Spam or Ham. The plot on the left shows those messages that have an Re: in the subject line tend to be ham. The right plot shows that those messages that are from a user with a number at the end of their email address tend to be spam. However, few messages are sent from such users so it is not clear how helpful this distinction will be in our classification problem.

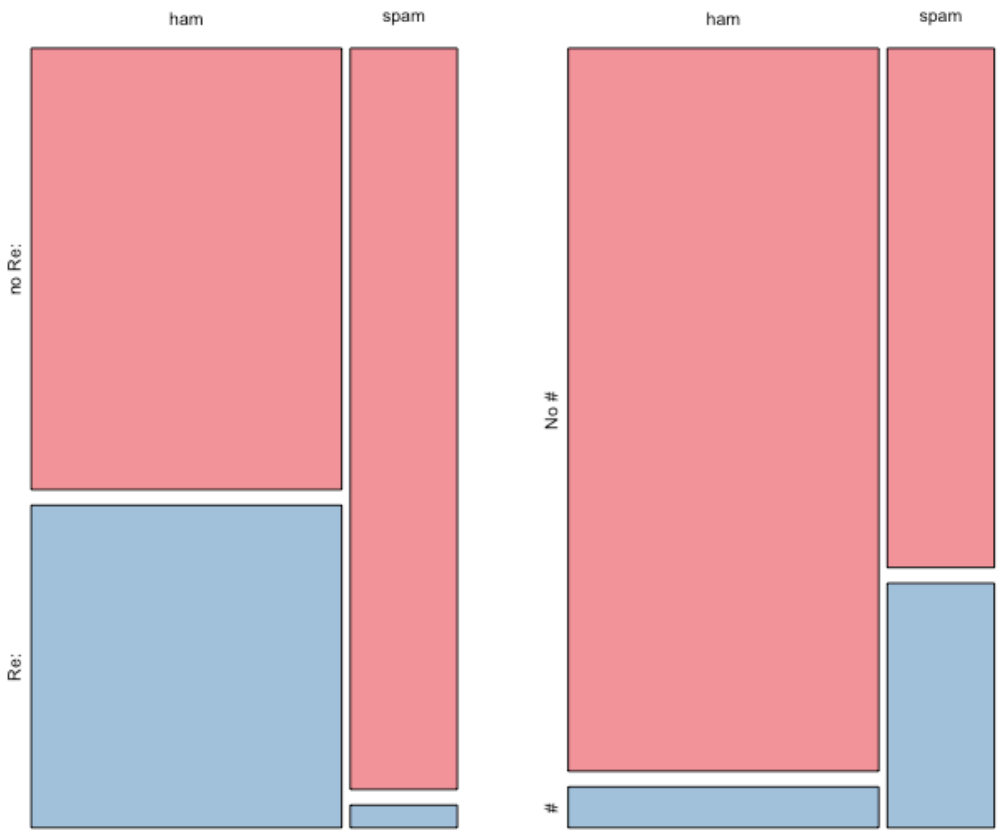

Figure 2.3.7 Exploring Categorical Measures Derived from email

2.4. Fitting the rpart() model to our data set

We split the dataset to train and test using the 1:3 ratio with a seed value for repeatability. We fit the data using rpart() function and method = "class" with default parameter values to get the following tree as shown in Figure 2.3.8

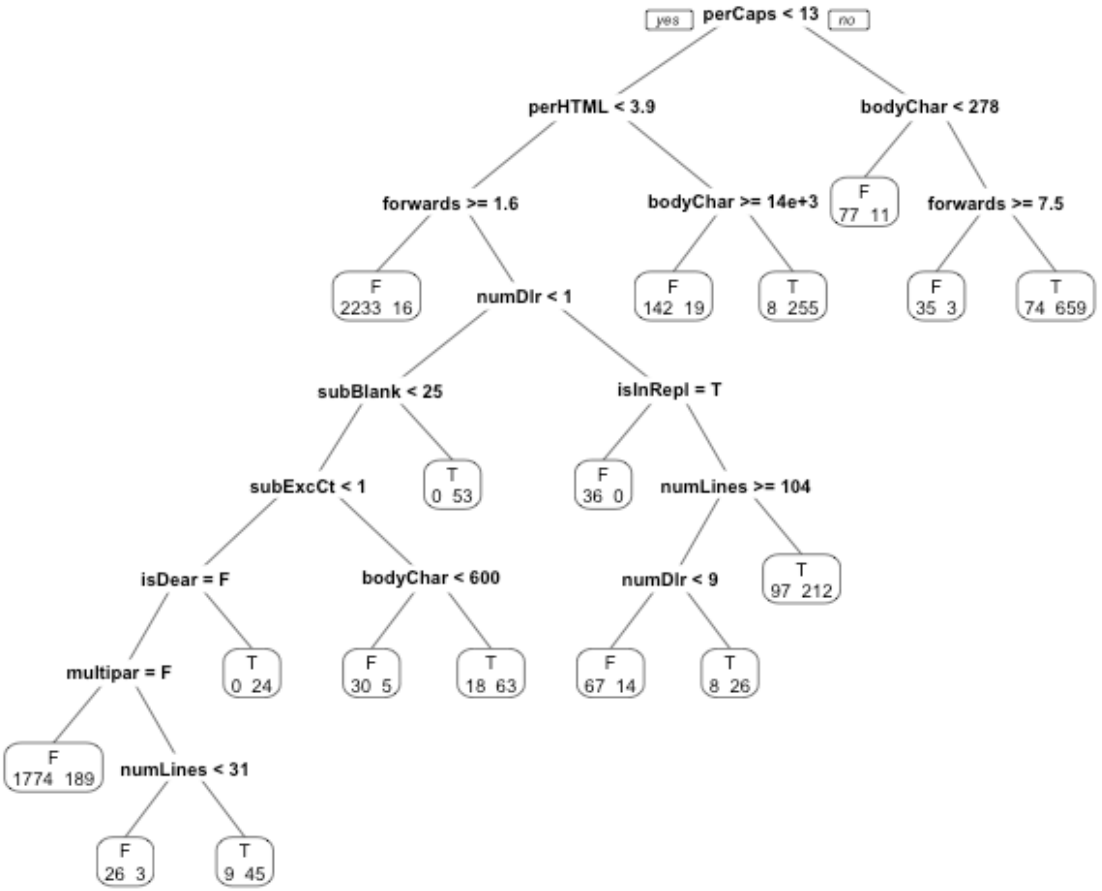

Figure 2.3.8 Tree for partitioning emails to predict spam with default parameter values

This tree was fitted using rpart() on the training data set . The default values for all of the arguments to rpart() were used. The leftmost leaf classifies as ham those messages with fewer than 13% capitals, fewer than 4% HTML tags, and at least 1 forward. Eighteen spam messages fall into this leaf and so are misclassified, but 2233 of the ham is properly classified using these 3 yes–no questions [2]

To find out how well our tree has performed in classifying these test messages, we compare the predictions from the fitted rpart object to the hand classifications. First we find the Type I error, the proportion of ham messages that have been misclassified as spam. The TypeI and TypeII errors as calculated as below:

```
0.0539490720759603
0.156445556946183
```

The decision tree building process can be tuned by using the control parameter of the rpart function. The control parameter itself is given as a rpart.control() function. The rpart.control() function allows us to control various aspects of the tree-fitting procedure. Some of the key parameters that can be used to tune the performance are:

- **minsplit:** the minimum number of observations that must exist in a node in order for a split to be attempted.
- **minbucket:** the minimum number of observations in any terminal <leaf> node. If only one of minbucket or minsplit is specified, the code either
  sets minsplit to minbucket*3 or minbucket to minsplit/3, as appropriate.
- **cp:** complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. We have done hyperparmeter optimization by running grid search on cp values ranging from 0 to 0.01 with increments of 0.0005 and the best results were noticed with a cp = 0.001.
- **maxdepth**: Set the maximum depth of any node of the final tree to 30, with the root node counted as depth 0

F1 score: This is a balance between precision and recall and is given by the below formula.

$$F1 = ((precision* recall)/ (precision + recall))$$

Precision: This is the fraction of relevant instances among the retrieved instances. It answers the question “How many selected items are relevant?” and is a measure of predicted True positives given false positives.

$$Precision = TP / (TP + FP)$$

Recall: This is the fraction of relevant instances that have been retrieved over the total amount of relevant instances – “How many relevant items are selected?” and is measure of True positives given false negatives.

$$Recall = TP/ (TP + FN)$$

In our scenario, ‘Precision’ would be a more suitable metric to evaluate the model that classifies the email to ham or spam. Having false positives can have a very bad effect in classifying emails especially in a corporate setting where an extremely important email could be lost.

After running the model through the caret grid with various cp values, we found an optimal cp value of 0.001 where the TypeI and TypeII errors were low and also showed best F1 score, precision and recall as shown in figure 2.3.9

```
> model_rpart
CART

9347 samples
  29 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 7478, 7478, 7478, 7476, 7478
Resampling results across tuning parameters:

  cp      F1         prec       rec        Type_I_err  Type_II_err
  0.0000  0.9578286  0.9548450  0.9608704  0.03380824  0.02909858
  0.0005  0.9588388  0.9555839  0.9621651  0.03327297  0.02813584
  0.0010  0.9596332  0.9562904  0.9630274  0.03273803  0.02749493
  0.0015  0.9586726  0.9545108  0.9628836  0.03412858  0.02760183
  0.0020  0.9572731  0.9554284  0.9591436  0.03327331  0.03038292
  0.0025  0.9564000  0.9549995  0.9578494  0.03359479  0.03134520
  0.0030  0.9545174  0.9520965  0.9569859  0.03584164  0.03198748
  0.0035  0.9528068  0.9486790  0.9569855  0.03851561  0.03198794
  0.0040  0.9519660  0.9489895  0.9549711  0.03819458  0.03348607
  0.0045  0.9506865  0.9487637  0.9526707  0.03830205  0.03519627
  0.0050  0.9508939  0.9532520  0.9486420  0.03466420  0.03819241
  0.0055  0.9485033  0.9551379  0.9420233  0.03295205  0.04311483
  0.0060  0.9456388  0.9480628  0.9433186  0.03851653  0.04215140
  0.0065  0.9450410  0.9478802  0.9423114  0.03862353  0.04290047
  0.0070  0.9448879  0.9478687  0.9420236  0.03862353  0.04311449
  0.0075  0.9442864  0.9463752  0.9423115  0.03980063  0.04290035
  0.0080  0.9440291  0.9455843  0.9425992  0.04044269  0.04268633
  0.0085  0.9432041  0.9402604  0.9461948  0.04472042  0.04001282
  0.0090  0.9432041  0.9402604  0.9461948  0.04472042  0.04001282
  0.0095  0.9431270  0.9392567  0.9470581  0.04557638  0.03937077
  0.0100  0.9431270  0.9392567  0.9470581  0.04557638  0.03937077

F1 was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.001.
```

Figure 2.3.9 Parameter grid after tuning cp value

3 Conclusion

In this case study, we acquired email messages data from Spam Assassins website, scrubbed the data manually to create a data frame with derived features for classification of spam and ham messages. We have further analyzed the spam messages dataset using various text mining and statistical techniques such as: box plots, mosaic plots, Summary statistics, decision tress.

We tried to get the best performance by tuning the rpart parameters in which the train control function enables us to control each parameter input and compare results.
The parameters we used for tuning is cp – control parameter. After tuning the values for these parameters and calculating prediction F1 Scores for each parameter combination, we narrowed down to one combination of tuned tree model with and a cp =0.001 as the best performed tuned tree with an F1 score of 96%.

Overall our model performance improved because of parameter tuning. We see that the classification done by our tuned tree is more accurate compared to that of the default tree based on the drastic drop in both Type I and Type II errors.

4 References

[1] https://en.wikipedia.org/wiki/Email_spam

[2] Nolan, Deborah; Lang, Duncan Temple. Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving (Chapman & Hall/CRC The R Series) (Pages 45, 101)

[3] Prof Slater's Jupyter Notebook sample and class material