Case 3

Matt Chinchilla, Drew Larsen, Rikel Djoko

10/4/2020

# Business Understanding

Email is a service that has all but replaced traditional mail in the 21st century. Unfortunately, junk mail, called spam mail in the context of email, has filled inboxes as email becomes more ubiquitous. Spam mail is significantly cheaper to send than junk mail, so there is very little stopping advertisers from sending spam emails to as many people as possible. Spam mail can range from mostly harmless advertising to phishing or malware links. The range of outcomes from clicking a spam link requires a system to properly identify spam. A human educated about spam mail can often spot spam emails at a glance, but what about a less tech savvy operator? An automated system to detect spam mail can be used to unclutter inboxes and help protect those who have trouble identifying spam mail.

In this analysis, we will create an automated system to identify spam emails using machine learning. More specifically, we will use information about the email such as the number of characters in the body, whether the email a reply email, or if the email body has images, among other characteristics. Using the extracted information, we create a tree based model in order to detect if an email is spam. Since we were given the data and did not perform the data extraction ourselves, we are assuming that the data extraction was done in a robust manner. The following analysis is only as robust as the data that we were provided.

# Data Evaluation and Engineering

### Data Description

The dataset is made up of a corpus of emails from SpamAssassin.org (Apache 4). In total, there are 9348 unique emails. It contains 29 predictor variables and one response variable named isSpam. Of the 30 total variables, 17 are boolean factor variables and the remaining 13 variables are numeric variables. Our goal is to use the existing email previously classified to classify future incoming email based on the value of the predictor variable. See table below for the list of the variables and the data dictionary.

|  |  |  |
| --- | --- | --- |
| 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. |

The output below is the structure of the data that we will be using.

## 'data.frame': 9348 obs. of 30 variables:  
## $ isSpam : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isRe : Factor w/ 2 levels "F","T": 2 1 1 1 2 2 1 2 1 2 ...  
## $ underscore : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ priority : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isInReplyTo : Factor w/ 2 levels "F","T": 2 1 1 1 1 2 1 1 1 2 ...  
## $ sortedRec : Factor w/ 2 levels "F","T": 2 2 2 2 2 2 2 2 2 2 ...  
## $ subPunc : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ multipartText: Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ hasImages : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isPGPsigned : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ subSpamWords : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ noHost : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ numEnd : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isYelling : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isOrigMsg : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isDear : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 1 1 1 ...  
## $ isWrote : Factor w/ 2 levels "F","T": 1 1 1 1 1 1 1 2 1 1 ...  
## $ numLines : int 50 26 38 32 31 25 38 39 126 50 ...  
## $ bodyCharCt : int 1554 873 1713 1095 1021 718 1288 1182 5989 1554 ...  
## $ subExcCt : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ subQuesCt : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ numAtt : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ numRec : int 2 1 1 0 1 1 1 1 1 2 ...  
## $ perCaps : num 4.45 7.49 7.44 5.09 6.12 ...  
## $ hour : num 11 11 12 13 13 13 13 14 14 11 ...  
## $ perHTML : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ subBlanks : num 12.5 8 8 18.9 15.2 ...  
## $ forwards : num 0 0 0 3.12 6.45 ...  
## $ avgWordLen : num 4.38 4.56 4.82 4.71 4.23 ...  
## $ numDlr : int 3 0 0 0 0 0 0 0 0 3 ...

### NA Values

The evaluation of the data set found 357 missing observations. Rather than doing a preliminary imputation we will remove the missing values as there are only a small number of rows with NA’s in comparison to the larger data set.

## [1] "number of missing value: 357"

Figure 1: Shows the distribution of spam vs not spam email in our dataset. We can clearly see the number of not spam is 3 X the number of spam.

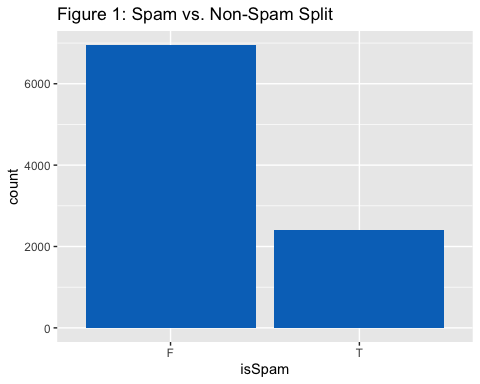


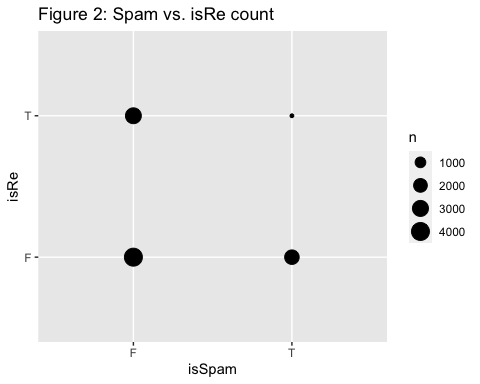
Figure 2: Shows the pairwise distribution between isSpam and isRe. Most of the of spam emails do not have Re: in the subject line. 

Figure 3 : Shows the pairwise distribution between isSpam and Priority level. We can clearly see the priority classification is barely used across spam and not spam.

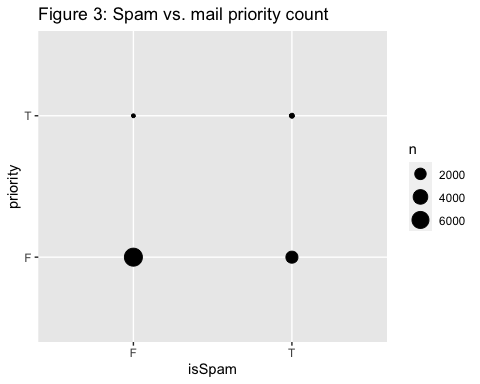
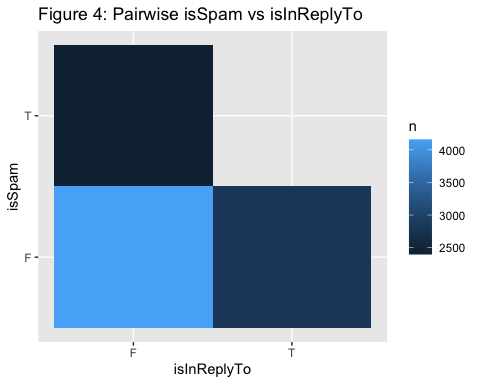
 The

Figure 4 : Shows the pairwise distribution between isSpam and isInReplyTo. We can see that most spam emails do not have the In-Reply-To key in the header.



# Modeling Preparation

As stated in the Business Understanding, we will be using a tree based algorithm in order to determine whether an email is spam or not. Decision trees are fairly useful for binary classification problems. The direction an observation flows through the tree structure is determined by decision nodes. If the decision node is based on a binary variable, the observation flows right or left based on the value of the variable for the observation. If the decision node is based on a continuous variable, the observation flows right or left based on if the observation’s value is above or below a certain threshold. The tree model has a binary structure and is a non-parametric model, so the variables used in the model don’t need to follow any distribution. This makes a tree model extremely useful for binary classification problems such as this one.

In order to find the “best” decision tree, you need to test every possible tree for the model. In order to avoid testing every possible tree, we decided to leverage the random forest model. The random forest takes a number of decision trees, which each make a decision if the email is “spam” or “ham”. If enough of trees classify the email as “spam”, then the email is classified as spam. If the number of trees classifying the email as spam is below a certain threshold, then we classify the email as not spam. Utilizing random forest allows us to test fewer trees in the model creation step and allows us to have a more robust model. What makes these forests “random” is how variables are chosen. In our model call, we specify how many variables are randomly chosen for each tree. This allows for many different trees to be created, instead of many similar trees with the same variables. In order to aid our ability to interpret these models we are going to set the maximum number of final nodes allowed per-tree in our Random forest to 10. Doing this will allow us to interpret the final model and how the trees are being classified using a single data point.

To determine the best model, we are using a few evaluation metrics. Accuracy, which is the number of correct predictions divided by the number of total observations, is important, but it doesn’t necessarily tell the entire story. It is important to minimize the number of false positives, but a large number of false negatives is more problematic. It is a bigger problem to classify a real email as spam, rather than classifying a spam email as not spam. If a spam email gets through our filter, then a human will waste a couple seconds deleting the email from their inbox. If an email gets incorrectly classified as spam, a potentially important email may not be seen. Therefore, we are also interested in looking at sensitivity and specificity. Sensitivity answers the question “how many of the positives classified by the model are actually positive,” while specificity answers the question “how many of the negatives in this model are actually negative.” “F”, or not spam, is considered to be our positive class, so more weight will be given to sensitivity, but specificity will also be considered.

If you are interested in reading more about the algorithms and accuracy measures used in this analysis, here are a few useful links:

Decision Tree Learning: <https://en.wikipedia.org/wiki/Decision_tree_learning>

Accuracy Measures: <https://en.wikipedia.org/wiki/Sensitivity_and_specificity>

Random Forest: <https://en.wikipedia.org/wiki/Random_forest>

# Model Building and Evaluation

### Model selection

There are many Random Forest algorithms that are available to use but for this exercise we are going to use the classic RandomForest package for R. This package implements Breiman’s random forest algorithm (based on Breiman and Cutler’s original Fortran code) for classification and regression. We will use this algorithm to classify SPAM and not Spam.

### Splitting the data training and test

In order to train and validate our model we are going to use a stratified split meaning that the data will be split in a way that preserves the proportions of True and False values observed in the complete data set. Training data will consist of 80% of our data, and the remaining 20% of the data will be a holdout set used to validate the model.

## [1] "Number of rows in Training data"

## [1] 7237

## [1] "Number of rows in Testing data"

## [1] 2111

### Baseline random forest model

First we will create a baseline Random Forest (RF) model so we can assess how well the RF model performs without any parameter tuning to classify our data. In the baseline model we will be using all the 29 available predictors in the data in order to classify “isSpam” as True or False. The baseline RF model does an fairly good job at classify email with an Accuracy of .8393, sensitivity of .9895 and, specificity of .5587.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction F T  
## F 1324 316  
## T 14 400  
##   
## Accuracy : 0.8393   
## 95% CI : (0.8227, 0.855)  
## No Information Rate : 0.6514   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6078   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9895   
## Specificity : 0.5587   
## Pos Pred Value : 0.8073   
## Neg Pred Value : 0.9662   
## Prevalence : 0.6514   
## Detection Rate : 0.6446   
## Detection Prevalence : 0.7984   
## Balanced Accuracy : 0.7741   
##   
## 'Positive' Class : F   
##

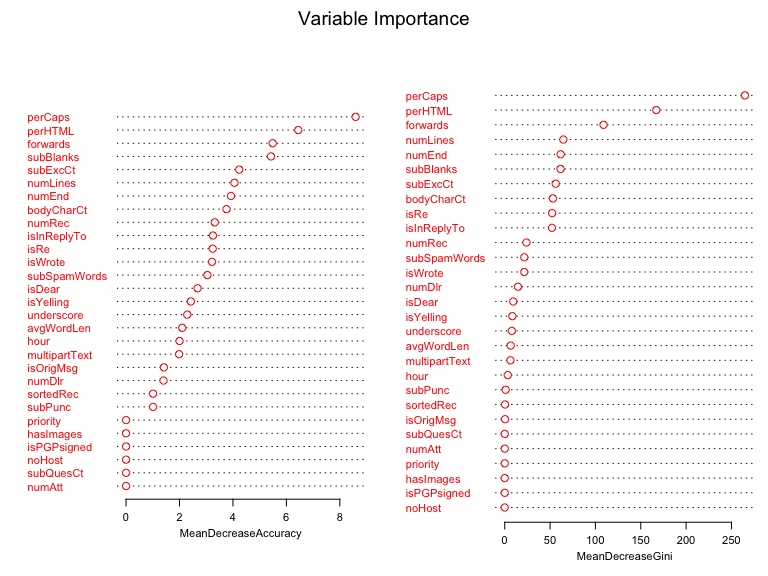
Time for this code chunk to run: 0.488492012023926

### Baseline model most important variables

After generating our basic RF model we can create variable importance plots in order to see what predictors are the most important within the basic RF model. The two plots below are:

**MeanDecreaseAccuracy:** gives a rough estimate of the loss in prediction performance when that particular variable is omitted from the training set.

**MeanDecreaseGini:** GINI is a measure of node impurity. Think of it like this: if you use this feature to split the data, how pure will the nodes be? Highest purity means that each node contains only elements of a single class. Assessing the decrease in GINI when that feature is omitted leads to an understanding of how important that feature is to split the data correctly.



### Rerunning baseline RF just using Best Predictors

**Remove low importance Variables**

After reviewing the best predictors in our baseline model we will remove variables with low importance and create a new cut of the training and test data. The top 15 variables in each list are our best predictors and we settle on a reduced set of variables that are a combination of the top 15 found in both charts and the remaining values unique to both lists. Our reduced set of variables is below:

Columns

isSpam

perCaps

perHTML

subExcCt

avgWordLen

subBlanks

isDear

numEnd

subSpamWords

bodyCharCt

numLines

hour

forwards

isYelling

numDlr

numRec

bodyCharCt.1

isInReplyTo

isRe

**Rerun baseline RF model with best predictors**

After creating a training and test data set using only our best predictors we rerun the baseline model. The purpose is to determine if we can achieve as good or better results than our baseline model using the best predictors as indicated by our Mean decrease accuracy and mean decrease GINI plots. After rerunning the baseline model we find that it did improve the baseline models accuracy to .8535, Sensitivity to .9933, and Specificity to .5922. Based on this we will choose to use our best predictors in building the final Random Forest Model.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction F T  
## F 1329 292  
## T 9 424  
##   
## Accuracy : 0.8535   
## 95% CI : (0.8374, 0.8685)  
## No Information Rate : 0.6514   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6447   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9933   
## Specificity : 0.5922   
## Pos Pred Value : 0.8199   
## Neg Pred Value : 0.9792   
## Prevalence : 0.6514   
## Detection Rate : 0.6470   
## Detection Prevalence : 0.7892   
## Balanced Accuracy : 0.7927   
##   
## 'Positive' Class : F   
##

Time for this code chunk to run: 0.414118051528931

### Model tuning

The random forest model has a number of options and parameters that can be adjusted but the two that we will focus on to tune our model are:

**nTree-**Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.

**mtry-**Number of variables randomly sampled as candidates at each split.

### K fold Cross validation and grid search

The methods we will be using to tune these parameters are K fold cross validation and grid search.

**K-fold Cross Validation** Cross validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

The three steps involved in cross-validation are as follows:  
1. Reserve some portion of sample data-set.  
2. Using the rest data-set train the model.  
3. Test the model using the reserve portion of the data-set.

The K fold part of cross validation denotes the number of times that we are going to perform the three cross validation steps. For our purposes we will perform a 10 fold cross validation.

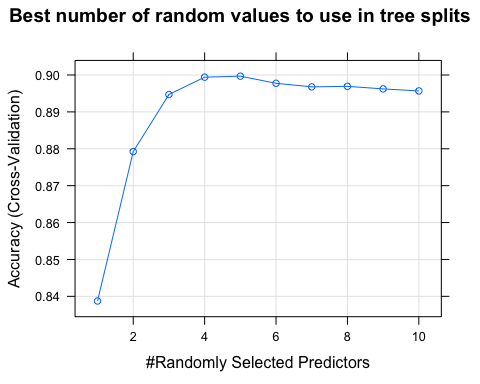
**Grid Search** A grid search is a technique that can be used to test specific values to be used for a certain parameter. In our case we are going to use grid search to find the best value for ‘mtry’ we’ll try a range of values from 1-10. In the grid, each algorithm parameter can be specified as a vector of possible values. These vectors combine to define all the possible combinations to try.

**Pre-Processing** For our final model we will also be performing a pre-processing step on the training and test data. For pre-processing we will center and scale our continuous predictors variables. Centering and scaling our data can help improve classification results by normalizing and standardizing the continuous variables in the data. All normalization means is scaling a data set so that its minimum is 0 and its maximum 1. Standardization or centering is slightly different; it’s job is to center the data around 0 and to scale with respect to the standard deviation.

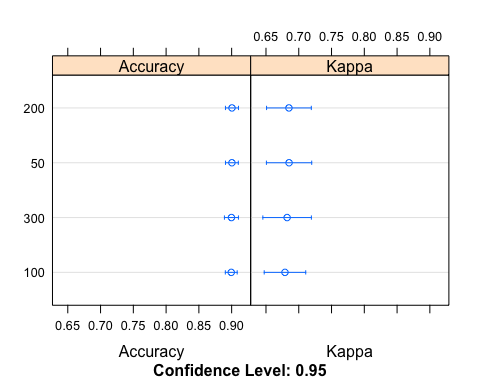
# Final model assessment

Our final model after tuning the parameters shows an improvement over our baseline model. Using cross validation, we find that our best mtry parameter is 5 and the best ntree value or number of trees in our random forest is 200. The output below shows the results of the cross validation a chart showing the accuracy by number of predictors used and the final output is a chart that shows accuracy and Kappa by the number of trees in our forest.

## Random Forest   
##   
## 7237 samples  
## 18 predictor  
## 2 classes: 'F', 'T'   
##   
## Pre-processing: centered (18), scaled (18)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6512, 6513, 6512, 6514, 6513, 6514, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 1 0.8387446 0.4047286  
## 2 0.8792343 0.5942248  
## 3 0.8947116 0.6573012  
## 4 0.8994088 0.6799408  
## 5 0.8996839 0.6822425  
## 6 0.8977504 0.6769958  
## 7 0.8967826 0.6738235  
## 8 0.8969213 0.6743510  
## 9 0.8962303 0.6722280  
## 10 0.8956782 0.6713231  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 5.



##   
## Call:  
## summary.resamples(object = results)  
##   
## Models: 50, 100, 200, 300   
## Number of resamples: 10   
##   
## Accuracy   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## 50 0.8769018 0.8929195 0.9013117 0.9003744 0.9044160 0.9253112 0  
## 100 0.8824343 0.8912293 0.8958621 0.8994100 0.9073654 0.9211618 0  
## 200 0.8755187 0.8940654 0.8998619 0.9005134 0.9044160 0.9266943 0  
## 300 0.8727524 0.8921906 0.8991035 0.8996839 0.9051386 0.9253112 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## 50 0.5988817 0.6628203 0.6885311 0.6852048 0.6981261 0.7715800 0  
## 100 0.6205896 0.6490041 0.6654567 0.6789620 0.7042428 0.7572325 0  
## 200 0.5992338 0.6603942 0.6832806 0.6850073 0.6960013 0.7752986 0  
## 300 0.5883554 0.6546552 0.6788893 0.6822425 0.6998244 0.7705355 0

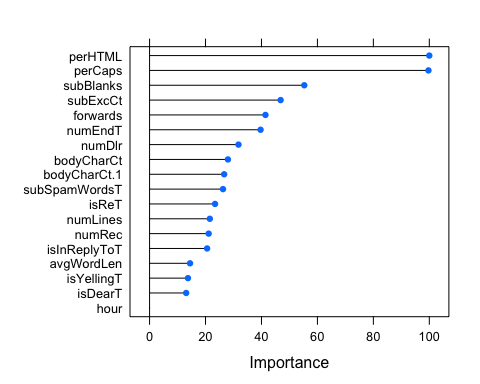
Time for this code chunk to run: 8.20329934755961

### Final model results

Below we see that our final model results are an Accuracy of .8579, sensitivity of .9849, and specificity of .6141. The hyper-parameter tuning has indeed improved our final models results.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction F T  
## F 1367 279  
## T 21 444  
##   
## Accuracy : 0.8579   
## 95% CI : (0.8423, 0.8725)  
## No Information Rate : 0.6575   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.655   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9849   
## Specificity : 0.6141   
## Pos Pred Value : 0.8305   
## Neg Pred Value : 0.9548   
## Prevalence : 0.6575   
## Detection Rate : 0.6476   
## Detection Prevalence : 0.7797   
## Balanced Accuracy : 0.7995   
##   
## 'Positive' Class : F   
##

### Most important variables used in final model

The most important variables used by our final model are perCaps and perHTML. The perCaps variable is the percentage of letters that are capitalized in the email and perHTML denotes the percentage of characters in HTML tags in the message body in comparison to all characters. These two variables make sense as the most important to the model because a spam or junk email usually has lots of capitalized letters meant to denote emphasis and excitement. Spam emails also generally contain links and other HTML formatting which would add more HTML tags to the email. 

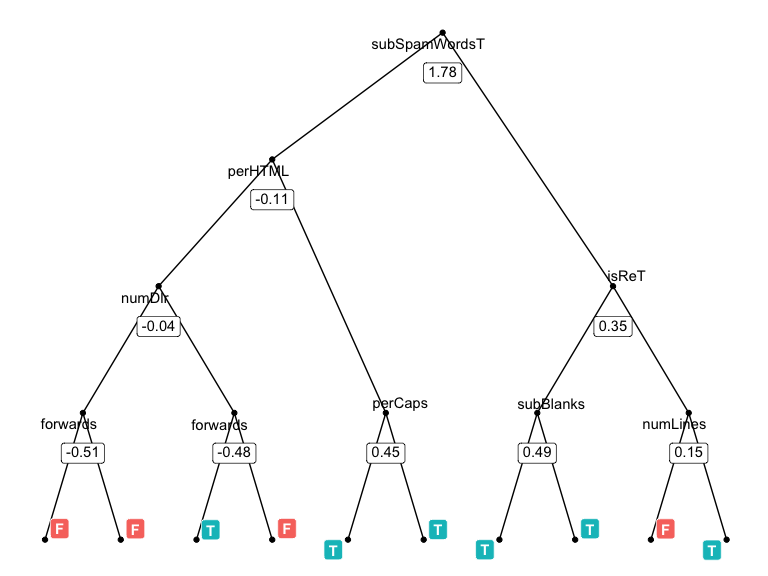
# Visualize Random Forest Tree

We will use the following data point as an example of how data would flow through the tree. The output of the data line below did not format correctly in this output but we can still use the values to illustrate how the tree works.

## isSpam perCaps perHTML subExcCt avgWordLen  
## ./Unit6//messages/easy\_ham7 F 6.343714 0 0 4.051402  
## subBlanks isDear numEnd subSpamWords bodyCharCt  
## ./Unit6//messages/easy\_ham7 17.02128 F F F 1288  
## numLines hour forwards isYelling numDlr numRec  
## ./Unit6//messages/easy\_ham7 38 13 0 F 0 1  
## bodyCharCt.1 isInReplyTo isRe  
## ./Unit6//messages/easy\_ham7 1288 F F

### Following the decision Tree

Below is the first tree that was produced in our final random forest model. We can see that if we follow the data point above we see that subSpamWords is “F”. This would mean that in our tree it would traverse to the left. The next node we hit is perHTML. The value is 0 so we would move to the right because our tree is using -.11 as the threshold. The final node we hit in our first tree is perCaps. We can see that regardless of the perCaps value the first tree would classify our sample data point as TRUE which corresponds to SPAM. Thus, this tree would classify our first data point as SPAM.



# Conclusion

After evaluating all three of the Random Forest models built for this exercise we find that our baseline model using our best predictors performs very well. With hyper-parameter tuning we are able to slightly improve the Random Forest model from our baseline model by adjusting the number of Trees that are in the forest to 200 and by updating the number of random variables used to split in each tree to 5. The table below shows the accuracy, sensitivity and specificity achieved by each model that we tested.

|  |  |  |  |
| --- | --- | --- | --- |
| Measure | Baseline\_RF | Baseline\_RF\_Best | Final\_RF |
| Accuracy | .8393 | .8535 | .8579 |
| Sensitivity | .9895 | .9933 | .9849 |
| Specificity | .5587 | .5922 | .6141 |

We believe that the Random Forest model is an excellent model for predicting if an email is spam or not. Moving forward we would suggest creating a data pipeline that allows the retraining of the random forest model as spam emails may change over time and the model would then need to be retrained in order to maintain quality. We would also recommend rerunning this analysis but allowing the trees to have more than 10 nodes. We did not increase this number in our RF model testing as it would make interpreting the model more difficult but reducing the number of nodes likely reduced the accuracy, sensitivity, and specificity of our models. We expect the model will perform substantially better if it’s not restrained to only 10 nodes per tree.