CASE STUDY 6

MSDS 7333 – Quantifying The World

Andrew Mejia, Sabrina Purvis, Sean McWhirter and Brian Gaither

Contents

I. Business Understanding

Spam emails, or unwanted mass-produced emails, are not only bothersome, but they also pose a security threat by potentially containing malicious software or as part of a phishing ploy. Although a greater proportion of the population is becoming more tech savvy, an unintentional click could cause serious computer issues for users. The best way to help defend our users against spam emails is to isolate suspected spam and remove it from the users' general inbox to keep it out of sight.

In addition to the security concerns, spam emails can quickly overrun a user's inbox. This poses an issue for the company operating the domain. If users become overwhelmed with spam, they may simply stop using the platform and try a competitor's product. In order to create an atmosphere that customers love using, keeping spam at bay to the greatest extent possible is a top priority.

While quarantining spam is essential, allowing legitimate messages pass on to the user is of the utmost importance. Important communications taking place via email being misclassified as spam should be avoided at all costs. If a user never sees a bill reminder, a tax document, or even just a letter from a loved one, their trust in the platform could be irreparably damaged. The potential cost of an email being misclassified as spam is greater than the benefit of a 100% spam-free inbox.

With these thoughts in mind, our goal is to build a spam filter that quarantines spam emails to the greatest extent possible and misclassifying non-spam emails as spam to the smallest degree possible. The model will be built on a dataset containing both spam and non-spam emails to train a model to make the distinction on a variety of factors covered below. While training will occur on a static dataset, the resulting model will be able to operate on emails it has never encountered.

Although the model will be designed to operate outside of the dataset it is trained on, it is likely that in even a few months from now, it will not be as effective as its first deployment. Con artists, like viruses and bacteria, mutate rapidly to the surrounding environment and keep proliferating. It is likely that the creators of these spam emails will realize that their spam is becoming less effective and change the angle of their attack accordingly. We are also operating under the assumption that this dataset contains an adequate representation of all spam and non-spam emails spanning the internet. It is likely that there are other features not captured in this dataset that could potentially identify spam more effectively.

II. Data Evaluation / Engineering

The dataset used in constructing this model is made available by SpamAssassin, part of the open-source Apache project. The dataset contains emails that have been identified as spam or 'ham' (not-spam). The target variable is isSpam, a logical variable with a value of TRUE if the email is spam, and a value of FALSE if the email is ham. The dataset contains 31 different variables and 9,348 emails. The variables that will be used to determine if a given email is spam or ham are below in Table 1.

Variable Name	Data Type	Description
isRe	logical	TRUE if Re: appears at the start of the subject.
numLines	integer	Number of lines in the body of the message
bodyChaCt	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 the 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 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.

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) (p. 110). CRC Press. Kindle Edition.

Table 1. Variable names, data types, and descriptions

It should be noted that the variable subSpamWords is a list of spam-related lexicon to assist these types of models identify spam emails.

It is important to highlight some of the assumptions we are working under with this dataset. As stated earlier, we are assuming that this dataset is representative of the actual population of spam and ham emails. An important deduction of that assumption is that we are assuming that a model built on this dataset will be effective on the active email population which is not captured in this dataset. Given that this is a pre-labelled dataset, we are also assuming that the dataset itself is accurate. We have not scoured the 9,000+ emails to ensure the accuracy of every attribute value.

The first thing we want to explore is how many values are missing from the dataset. The number of missing observations will determine the method in which we handle those missing values. If there are few missing values, dropping the observations will likely have little to no impact on the outcome. If there are many missing values, imputation would likely be the best course of action to retain a suitable number of observations. The SpamAssassins dataset contains 303 missing observations. Since that is only 3.2% of the total observations, we will proceed with removing those observations from the dataset leaving us with 9,045 useful observations.

The next area of investigation is to determine the proportion of the observations in our dataset that are spam and how many are not. The proportion of spam vs ham emails in our dataset is important for a couple of reasons. Firstly, it will determine whether this is a suitable dataset. If there is not an adequate number of spam emails, it is likely that any model built would not provide any significant utility. Secondly, the proportion will help determine the model construction process itself. If the number of spam emails is significantly smaller than the number of ham emails, we would want to split our training and test sets accordingly to ensure the same ratio exists in both. A training set with no spam emails would not produce an effective model. Similarly, it can help select the model evaluation metrics. If spam emails only comprise 20% of the dataset, a model with 80% accuracy would be meaningless as it could classify every email as ham. Therefore, more specific would likely need to be addressed such as precision and recall.

Ham Emails	Spam Emails

73.79%	26.21%

Table 2. Ham versus spam proportion

As Table 2 illustrates above, 26.21% of the observations in the dataset are spam emails. Given that the proportion is imbalanced, we will proceed with stratified sampling. Stratified sampling ensures that the proportion of target observations remains consistent in the training and test datasets. We will also use an 70/30 train-test split; 70% of the dataset will be used for training, and 30% will be used for testing.

Initial investigation of the dataset also includes examination of potentially important features. To conduct this investigation, we use two primary types of plots, kernel density plots and stacked bar charts. Kernel density plots show the probability distribution of a continuous variable. We will be plotting the distributions of spam and ham emails to identify any separation of those distributions. The stacked bar chart will be used to show the logical variables (true/false) and the count of each that are spam or ham emails.

The first to examine is perCaps, the percentage of capital letters in the body of the email. This variable makes sense to investigate, as anecdotally, most spammers are attempting to grab your attention. In Figure 3 below, we can see that the distribution of ham emails is extremely focused around the 5%-6% range of capital letters with a slight skew to the right. Spam emails, however, have a much broader distribution, with relatively significant densities spanning from 12.5% up to approximately 60%.

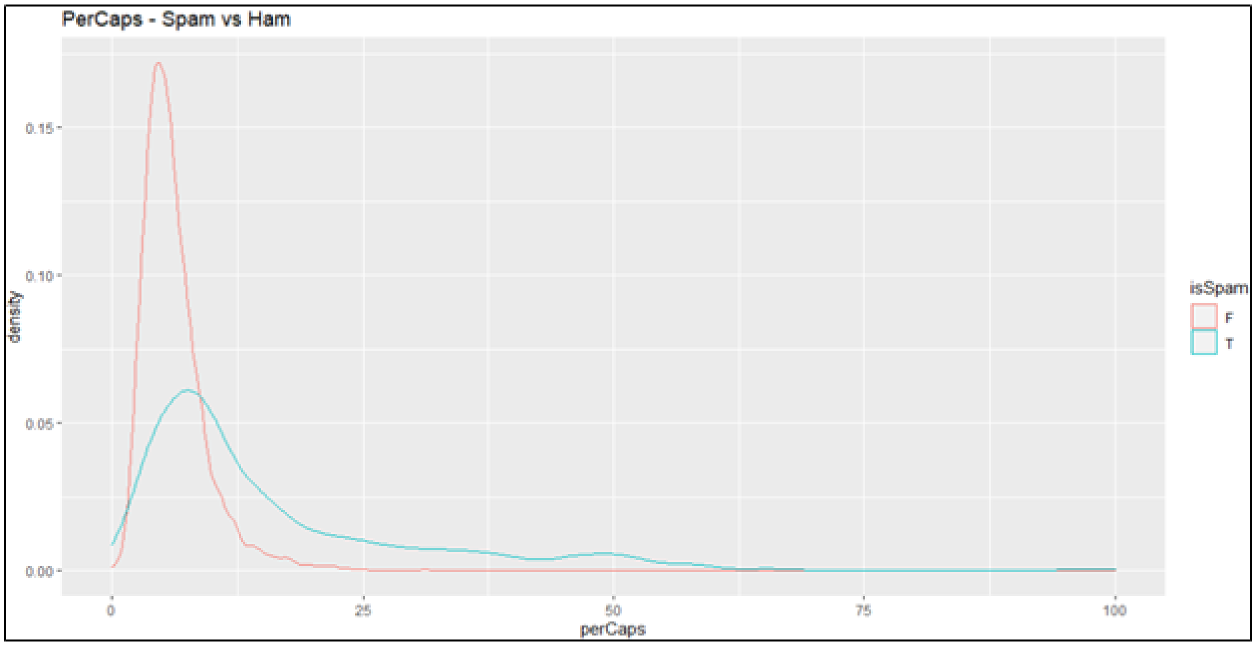


Figure 3: Probability distributions of perCaps and their classification of spam versus ham.

IsRe is another variable worth investigating. IsRe provides a true or false value based on whether 'Re:" appears in the subject title. As "Re:' generally appears in ongoing conversations, this variable could potentially be useful. Figure 4 below shows whether or not the subject lines contained "Re:" on the x-axis and their count on the y-axis. Ham emails are filled in with orange, and spam emails are filled in with turquoise. By referencing Figure 4, it is evident that a greater number of emails did not contain

"Re:" in the subject line, however, a much larger proportion of emails containing "Re:" were ham emails.

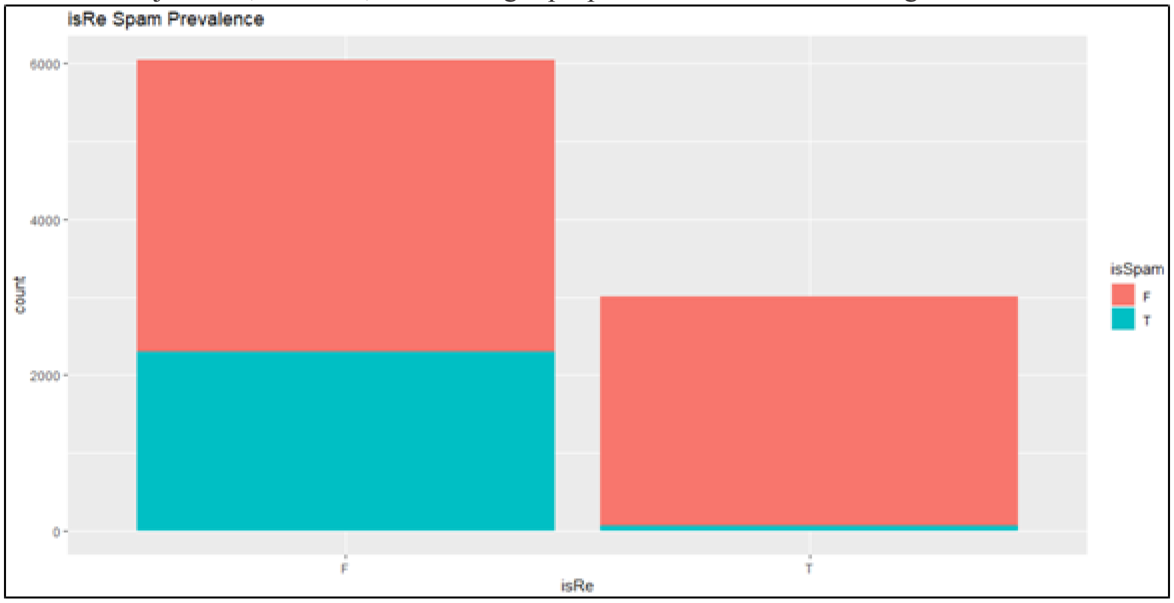


Figure 4: Spam prevalence for the isRe variable.

While perCaps and isRe seem to be potentially useful variables, a large number do not provide such useful information off the bat. Figures 5, 6, and 7 show the drastic discrepancy of ham emails to spam emails. For most of these, the number of spam emails is simply too low to make a definitive decision prior to model training.

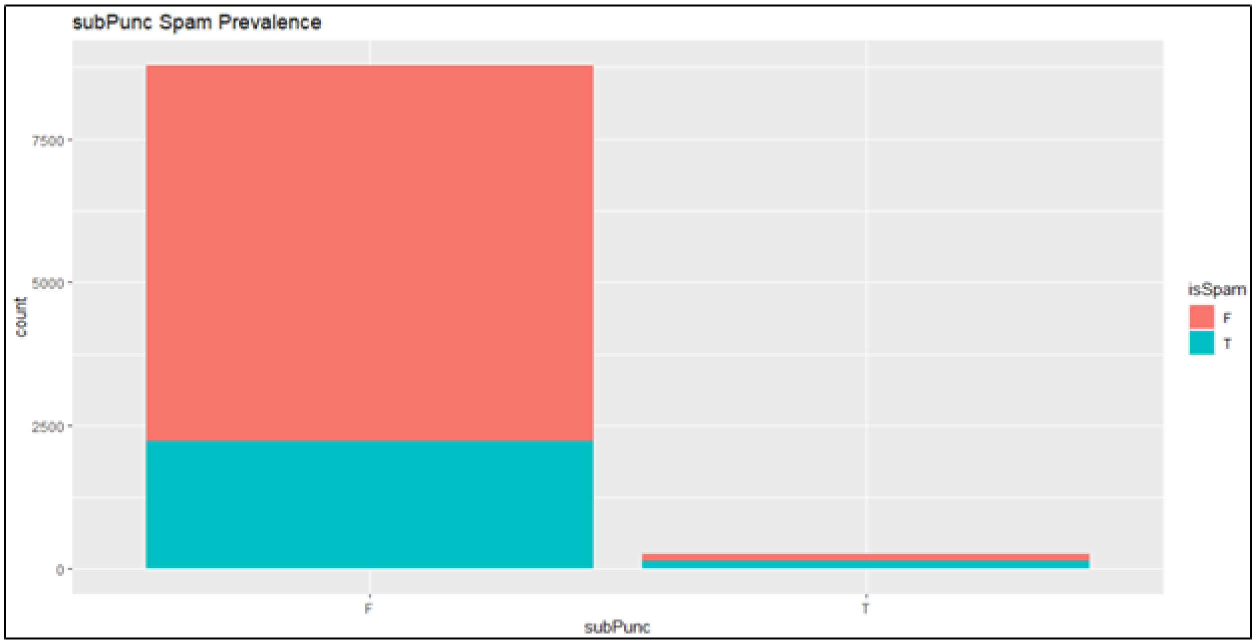


Figure 5: Spam prevalence for the subPunc variable.

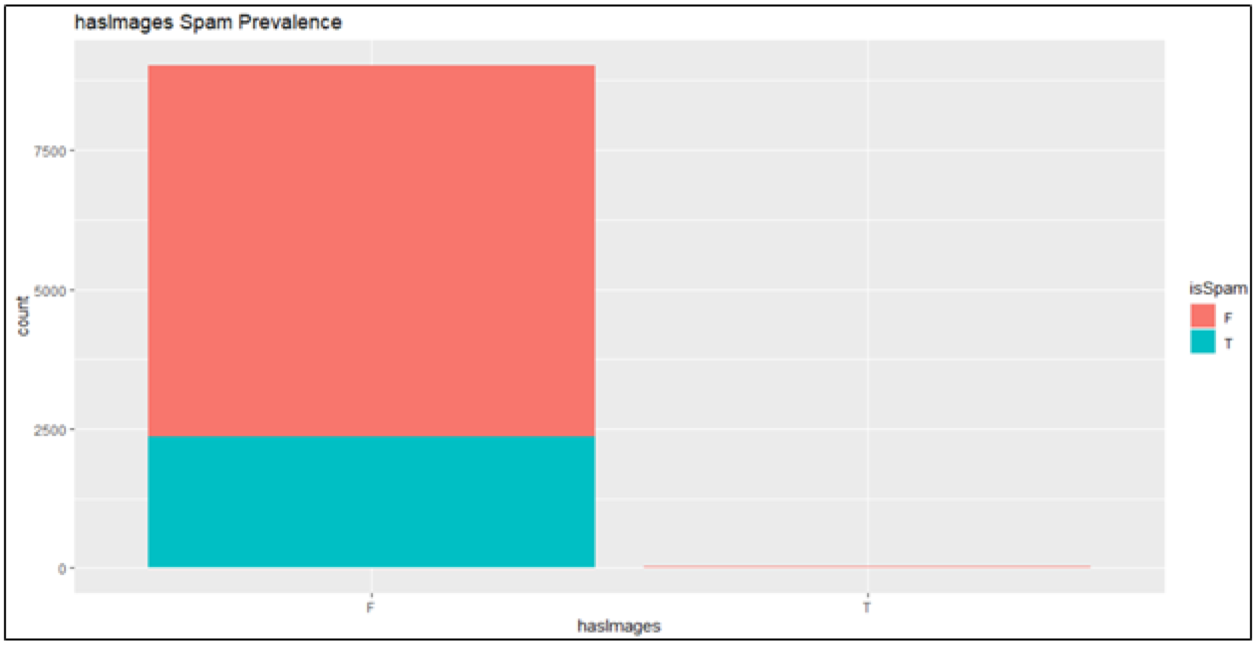


Figure 6: Spam prevalence for the hasImages variable.

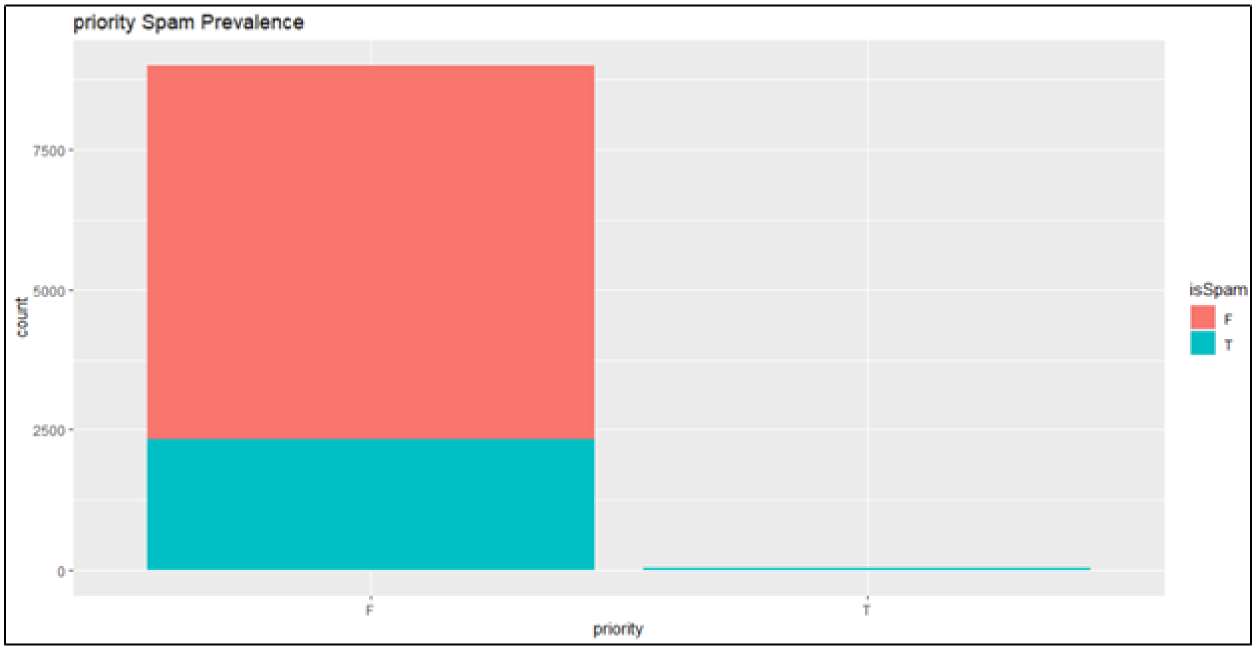


Figure 7: Spam prevalence for the priority variable.

Given the overlap of spam and ham email attributes, a tree-based model will likely provide the most reliable outcome. This will allow us to use all the variables in the dataset without degrading performance. A tree-based model will also provide an interpretable result that can provide clarity on which attributes are the most important when it comes to determining ham vs spam. The most important attributes can then be compared to future model outcomes to identify measures of the filter that spammers could be adapting to.

III. Modeling Preparations

As mentioned previously, the primary objective of this research is to develop a binary classification model that can correctly classify an email message as either spam or ham. As this is a non-linear problem, we will be utilizing a tree-based model approach to perform our classification. Since the application of our model will be used to essentially block unwanted spam emails from reaching the inbox, we will want to be careful to optimize for minimal false positives. A false positive in this case would result in a legitimate and possibly important email from appropriately reaching the inbox. Therefore, a false positive scenario could potentially be very problematic for a user.

To ensure we develop a fit for purpose model, we will emphasize the precision of the mode whereby we maximize the true positives and minimize the false positives. To do this, we will need to make a tradeoff for FN's or misses. We will also perform cross validation that aims to identify the model that maximizes AUC such that we get the maximum TP for tradeoff in FP.

IV. Model Building & Evaluation

We utilized an XGBoost tree-based model due to its advanced features for model tuning and its gradient boosting abilities. The XGBoost is an ensemble of trees that aims to create a strong classifier based on weak classifiers. By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor until the training data is accurately predicted or reproduced by the model. Gradient boosting fits a new model to new residuals of the previous prediction and minimizes the loss when adding the latest prediction. The model is therefore updated using gradient descent which is where the name gradient boosting is derived[i].

We have leveraged seven hyperparameters for our model using the XGBoost implementation in R. These seven hyperparameters are listed below:

Parameter	Description
nrounds	Number of boosting iterations
max_depth	Maximum depth of the tree. Increasing this value will make the model more complex and more likely to overfit. Therefore, by keeping the tree depth at a minimum we can combat overfitting.
eta	Step size shrinkage prevents overfitting. After each boosting step, we can directly get the weights of new features and eta shrinks the feature weights to make the boosting process more conservative.
gamma	Minimum Loss Reduction required to make a further partition on a leaf node of a tree. The larger the gamma is, the more conservative the algorithm will be.
colsample_bytree	Subsample Ratio of Columns for each level. Subsampling occurs once for every new depth level reached in a tree. Columns are subsampled from the set of columns chosen for the current tree.
min_child_weight	Minimum Sum of Instance Weight. If the tree partition step results in a leaf node with the sum of instance weight less than min child weight, then the building process will give up further partitioning.
subsample	Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees and this will prevent overfitting.

Table 1: Hyperparmeters explicitly utilized in the XGBoost model.

We have used a grid search approach to perform hyperparameter tuning. This allows us to build models using the various permutations of hyperparameter values specified to identify the optimal model based on our specified objective which is AUC. The various parameter values utilized in the grid are as follows:

Parameter	Values
nrounds	50, 75, 100
max_depth	4, 5, 6, 7
eta	0.05, 0.3, 0.075
gamma	0
colsample_bytree	.3, .4, .5
min_child_weight	2.0, 2.25
subsample	1

Table 2: Hyperparmeter values used for grid search.

We used a 70/30 split into a training and test set using a stratified approach to ensure we have an equal proportion of the target variable (isSpam) in both the training and test sets. We performed a 5-fold cross validation repeated 3 times solving for maximum AUC. This was performed using a package called Caret and we leveraged a package called doSNOW to allow us to utilize multiple cores for parallel processing to improve resource utilization and overall time efficiency.

The optimal hyperparameters (best tuned model) resulted in the following hyperparmeters:

Parameter	Optimal Values
nrounds	100
max_depth	7
eta	.3
gamma	0
colsample_bytree	0.5
min_child_weight	2
subsample	1

Table 3: Optimal hyperparmeters for best XGBoost model.

The ROC curve for the optimal model is output below showing an AUC of 0.9984. Note, this ROC curve has been zoomed in to look at FPR between 0 and .10 so that we can see the various threshold ranges that would give us the tradeoff between TPR and FPR.

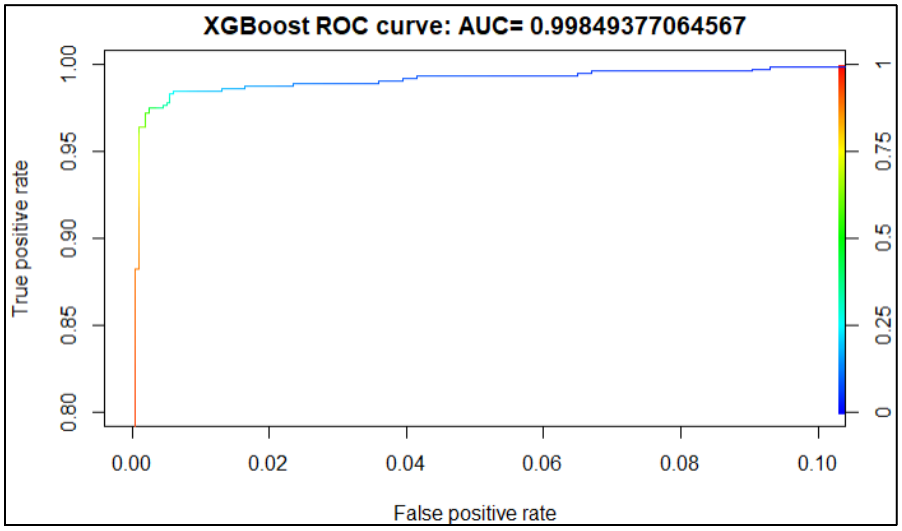
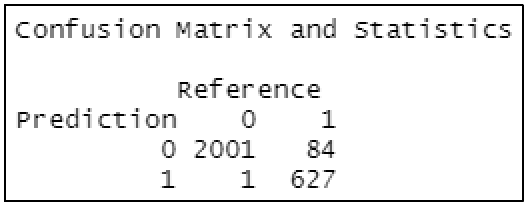


Figure 1: ROC curve for final XGBoost model.

The ROC curve includes a color gradient that helps visualize the threshold at which different true positive rates (TPR) and false positive rates (FPR) can be achieved. We want to maximize the TPR and minimize the FPR. Therefore, we want to select a threshold that is in the dark red area of the threshold before the curve starts darting off to the right.

We selected a threshold of 0.88. When applying this threshold on the test, we were able to achieve a precision of .9984 (TP/(TP+FP)). The confusion matrix for the test set is shown below:

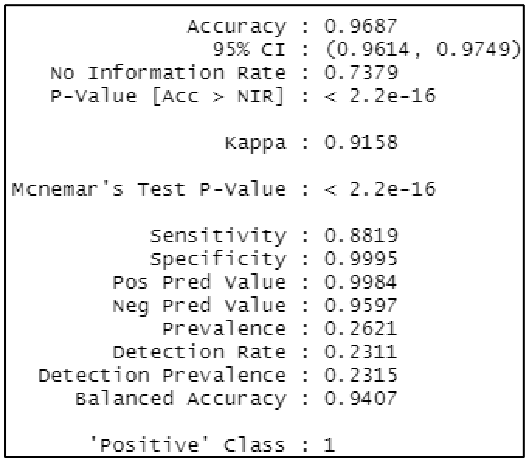


```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2001   84
         1    1  627
```

Figure 2: Confusion Matrix for best XGBoost model.

Additional model metrics for the best XGBoost model on the test set are below:



```
               Accuracy : 0.9687
                 95% CI : (0.9614, 0.9749)
    No Information Rate : 0.7379
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9158

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8819
            Specificity : 0.9995
         Pos Pred Value : 0.9984
         Neg Pred Value : 0.9597
             Prevalence : 0.2621
         Detection Rate : 0.2311
   Detection Prevalence : 0.2315
      Balanced Accuracy : 0.9407

       'Positive' Class : 1
```

Figure 3: Model metrics for best XGBoost model.

With the model metrics above, we can accurately classify 627 spam emails out of a total of 711 in our test set while only falsely classifying 1 ham email as spam. This means that out of all the messages our model classified as spam, only 0.2% are incorrectly classified. Even if we were to move to a threshold of .99, we would still have 1 false positive. Using a threshold of 0.875 would result in 2 false positives. Therefore, .88 is the optimal threshold that minimizes the false positives and maximizes the volume of true positives.

It should be noted that when comparing the test accuracy against the training accuracy, we do see evidence of overfitting of the model. The test accuracy is 0.9687 while the training accuracy is 0.9722 with a difference of 0.0035 between the two. We have experimented with early stopping and adjusting the max tree depth to identify an optimal model whereby the test accuracy is slightly better than training accuracy, however, this model was chosen due to its performance and the fact that the overfit is quite marginal, in fact nearly equal. From this analysis, we believe this model will generalize well.

V. Model Interpretability & Explainability

As mentioned, we utilized XGBoost to generate our model. XGBoost produces a few metrics when quantifying split decisions. Gain describes the relative contribution of the associated feature to the overall model by ascribing each feature's contribution for each tree in a model. When assessing gain, higher comparative values imply more importance when generating predictions. Coverage illustrates the relative number of observations related to the feature across all trees. Each time that a feature is used to decide the leaf node of an observation in any tree, it is counted once. The Coverage metric is expressed as a percentage for all features.

Figure 4 shows a summary of the key features when predicting spam across all threes in the model.

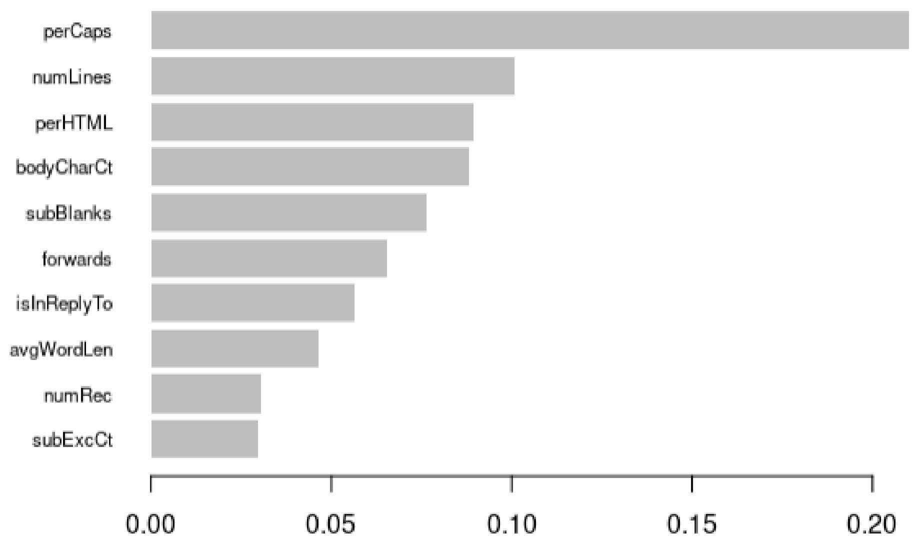


Figure 4: Top 10 features for best XGBoost model.

As we explore Tree 0, we can see how each of these features is considered when determining the final classification. As was noted above, our final parameters determined we would have a depth of 7; as such, we broke down the tree to aid in interpretability. You will see in Figure 5, the tree begins by splitting on the count of perCaps in each email at 12.08%. This split created coverage over 1583 observations across the trees and produced a gain of 1125.43. As you look down the tree, you can see that preliminary gain is 3 to 5 times larger than the next split. Next, if we follow the upper branch, splitting on subBlanks at 24.78 created a gain of 361.47 and coverage on 1337 observations. This split produces a leaf from one split with a final value (log likelihood) of 0.58. This transforms to a probability of being Spam of 0.6415. Again following the upper branch, we have one more split we will discuss on isInReplyTo. Recall that this is a True/False attribute, so less than 1.5 is true and greater than 1.5 is false. This split produces another leaf, covering 438 observations where the email is in reply to something. This classifies as ham (0.3546; transformed from the log likelihood of –0.5986)

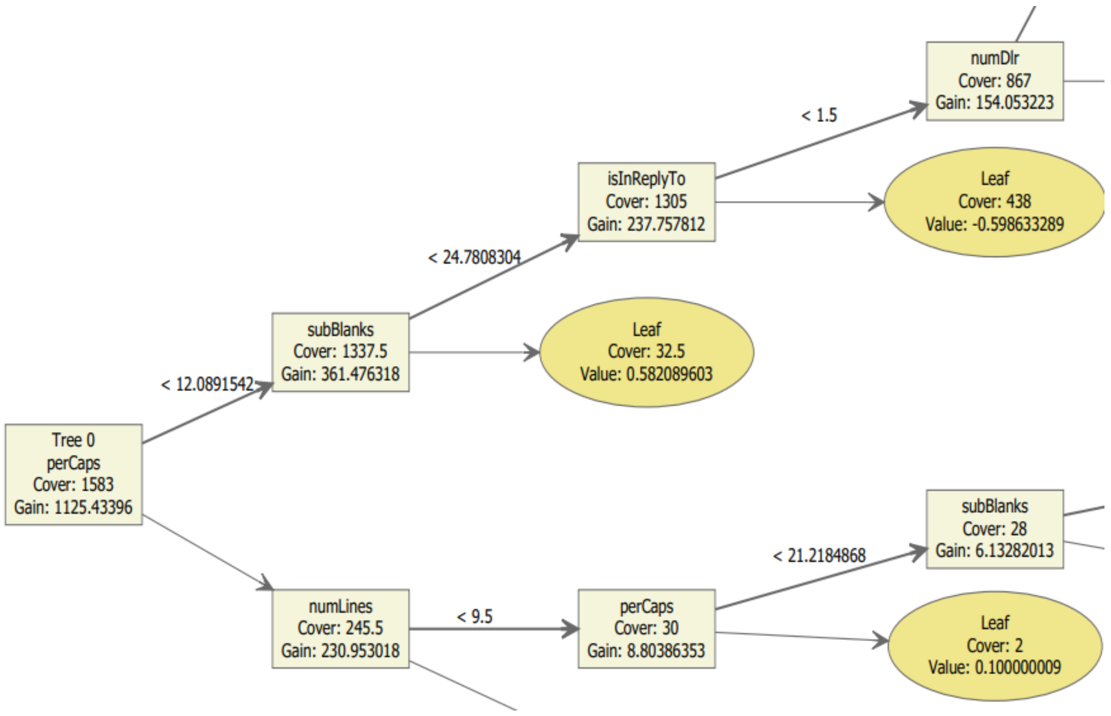


Figure 5: XGBoost Model Tree representation (part 1 of 2)

Next in Figure 6, we restart at the base of the tree and split down the branch. We follow perCaps next, and have a final leaf where the percentage of capitalized letters in the message is greater than 21.21%. Where true, the model concludes probability that it isSpam of 0.524 (from log likelihood of 0.100) with coverage of 2. We next split on subBlanks percentage of 10.23%, where less than creates a leaf with a .5 probability. Greater than 10.23% has coverage on 26 observations and a 0.36 probability of being spam. We again see on the lowest branch that isInReplyTo of true splits to create a leaf with 0.36 probability of being spam. The final leaf presented is split from another True/False classification. Wrote, as a keyword, is important in the body of the text. If the word wrote appeared in this subset of observations of the tree, it produced a probability of 0.39 of being spam.

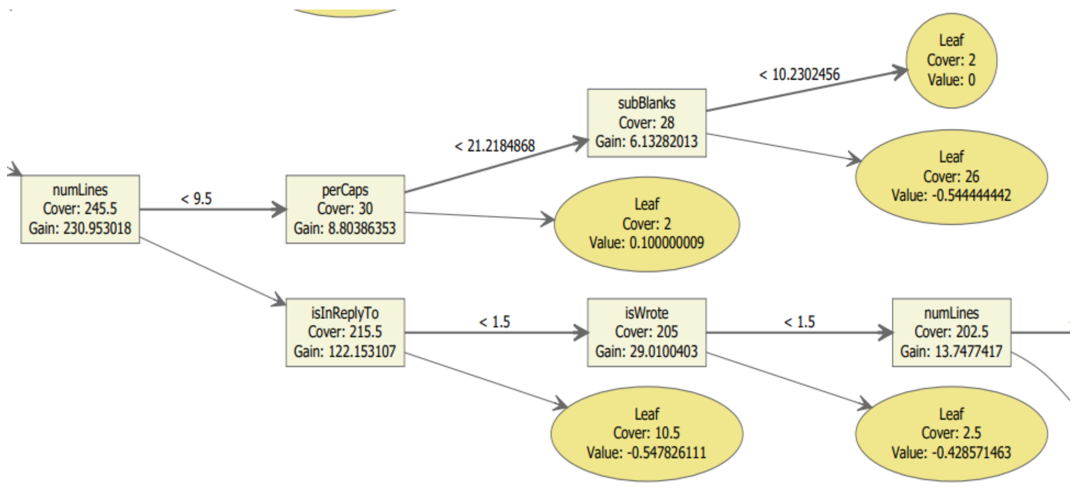


Figure 6: XGBoost Model Tree representation (part 2 of 2)

Even in this focused examination, we can see that the attributes defined have significance to the classification, and at various thresholds as well.

VI. Case Conclusions

A key advantage to using tree and boosted gradient tree classification methods is their high level of interpretability. From figure 4, we can see the top 10 features from the best XGBOOST model. The trees voted Percentage of Capitals characters in the Body, Percentage of blanks in the subject line and Percentage of characters in HTML tags in the message among the top feature to predicting whether an email is spam or not. It is important to note that many of the top features the model picked are engineered features. Now having these individual features alone in an email does not make an email instance spam, however, when these features are present and cooccur, the likelihood of an email being present does increase.

This means the model has picked up the human observations that a spam email will likely contain a higher percentage of HTML tags, have a longer character count in the body and will likely have a higher percentage of capital case characters. When looking at the structure of a SPAM email, all these elements are present more than not, usually due to SPAM being some advertisement of sorts with links in a type of HTML table.

As discussed earlier to gain better performance from the model would not necessarily be found in tuning the model’s hyper parameters or even the cutoff likelihood probability. From the feature importance, it would be more prudent to continue to explore further feature engineering perhaps in the form of the percentage of either misspelled or unknown words in the body and header of the email or looking at the percentage of forwards in relation to the overall volume of the emails in a given email queue.

A potential limitation to the utility of tree-based methods is generalization. In fact, the reader will notice a slight difference in the tree node diagrams in figures 5 and 6 from the top ten features in figure 4 in terms of ranked feature importance. This is due to not seeding the test train partitions and using a gradient boosting technique that randomly selects features in order to gain better performance from the weaker learner trees. This observation shows, while not impacting the practical importance of the features of the model, it is evident that tree-based models are highly sensitive to changes in the data. Meaning, if there are any changes in sampling techniques, such as a period of heavy receipt of spam email or the opposite, the tree-based models will be susceptible to these changes in the data and will shift the feature importance's to what is most prevalent in the training data. This could lead to a statistical occurrence known as under specification. Which translates to, features in the training data may not exist in the test and ultimately in the data the model is trying to predict against, which will degrade the model’s performance. Hence, the model should be further refined to account for other engineered features in order to keep up to date with the latest trends in the composition of spam.

[i] KDnuggets: XGBoost, a Top Machine Learning Method on Kaggle, Explained
https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html