Spam Detection Using RPART

Dennis Murray, Jared Law, Julien Bataillard, Cory Nichols

March 20th, 2018

## Abstract

The use of email has grown exponentially since the introduction of the world wide web in the late 20th century. Today, spam email is ubiquotous on every email platform. Spam detection methods to filter out unwanted emails originated in the late 1990s, and while the algorithms have improved, so have spam avoidance methods. In this paper, we explore rpart; a classification and regression tree package in R. Specifically, we explore the effectiveness of spam classification using rpart and its hyperparameters on a dataset of emails previously classified as spam or valid email. After parameter optimization, our custom rpart model outperforms an rpart model using default settings.

## Introduction

Electronic mail is an integral part of our lives. The information spread through the use of email is massive in scale, and often includes unwanted marketing and phishing. Email adoption spread rapidly in the 1980s, and so did the ability for companies to market products to email users. In addition, entities with ulterior motives sprang up, attempting to gain information from people using phishing and social engineering. These unwanted emails were given the term “spam.”

Spam filters were introduced not long after the introduction of email. These filters automatically process the incoming messages, applying certain techniques to identify unwanted emails. Bayesian email filters began to be utilized in 1996 but didn’t become popular until much later. These techniques utilize the probabilities of certain words occurring in regular emails versus spam emails.

Today, many different methods exist for spam email classification. One such spam filtering method is the utilization of decision trees or recursive partitioning. We explore a decision tree package in R called rpart. Our objective is to investigate and optimize key hyperparameters used in rpart in order to classify email messages as spam or valid email. Specifically, we analyze two separate decision trees for classifying spam email using the rpart package. We fit one decision tree on training data using default hyperparameters in rpart. After optimizing the minsplit, maxdepth and complexity parameters, we fit a second decision tree on the training data. We compare each model’s generalization performance on a test dataset containing emails previously classified as spam or valid email. The method used for optimization in our case is AUC or the area under the ROC curve. We use this metric to maximize the true positive rate of our classifiers. However, we also consider false negative and positive rates when comparing models to determine if optimization has resulted in a better rpart model overall.

In the subsquent section, we review research literature and introduce the email dataset. In the methods section, we explore the emails dataset in detail and explain the methods used for exploring rpart decision tree parameters. We implement two different decision trees and compare their generalization performance to determine if hyperparameter tuning results in better performance. Our paper concludes with a discussion of the applications of the improvements in anti-spam email filtering.

## Background

### Literature and Decision Tree Overview

In their paper from the Hawaii International Conference on System Sciences, Cukier and Cody define spam as “including all electronic messages that are unsolicited or unwanted, sent to many users (in bulk), without regard to the identity of the individual user, and usually having commercial purposes” [1]. Spam also includes messages containing attachments that spread viruses through emails. They state that in 2002, spam numbers peaked at one in three email messages. Also, in 2003, approximately 20 billion spam messages were sent daily [1].

Clearly, spam has become a major problem for users, businesses, and the internet in general, which led to the introduction of spam filters. These filters have historically relied on keywords within the message to identify spam. Some of the methods used include list-based filters (classifies the sender), content-based filters (uses words within the message), challenge/response system (sender needs to complete an extra task before the message can be delivered), and collaborative filters (users report spam messages which are stored in a database) [2].

Many in academia focus on creating a web spam taxonomy to prevent spam from spreading. The literature goes into the different types of spamming and the way it is used to collect information from the users. But as spam detection improves, so does the spammers’ techniques to send spam [3].

There are many types of classification methods to detect spam, some of which include support vector machines, naïve bayes classifiers, and decision trees. The latter is the method used in this project. A decision tree algorithm is a tree-like graph used to model decisions and their consequences based on a given set of rules. It can be used to classify unlabeled data [4], such as the dataset of emails we use in this case study to identify spam. In our case, we utilize rpart, a recursive partitioning package in R. The rpart package is one of the most commonly used packages in R. It implements the classic non-parametric CART algorithm, using the Gini index as a default splitting criteria.

### Data Description

The email dataset used for our analysis task is made up of a corpus of emails from SpamAssassin.org (Apache ). In total, there are 9348 unique emails. Each observation is made up of 29 predictor variables and one response variable. Of the 30 total variables, 17 are boolean factor variables and the remaining 13 variables represent numeric variables. Each email has been previously classified as spam or valid. We will use these predictor variables and the isSpam response to create multiple rpart decision tree models. A high level listing of variable names is given in Table 1.

Table 1: Variable Listing *Numeric vars - perCaps, bodyChartCt, numLines, subExcCt, subQuesCt, numAtt, numRec, hour, perHTML, subBlanks, forwards, avgWordLen, numDlr* Boolean vars - isSpam, isRe, isYelling, underscore, priority, isinReplyTo, sortedRec, subPunc, multipartText, isPGPsigned, subSpamWords, noHost, numEnd, isOrigMsg, isDear, isWrote

For context, a sample description of five variables is given below. We omit the rest of the variable descriptions for brevity and give explanations where appropriate in future sections.

*perCaps: percentage of capitals in the email body* isYelling: subject alpha characters are all capital *bodyCharCt: number of characters in the body of the email* numEnd: email ends in numbers \*isRe: subject contains reply characters Re:

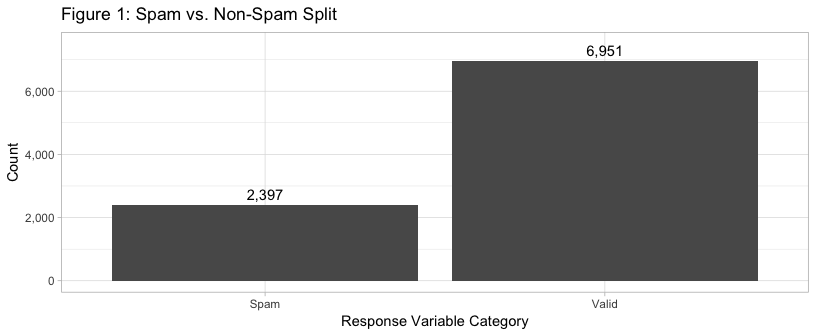
## Methods

Prior to fitting decision trees using rpart, we explore the email dataset in more detail. Specifically, predictor variable relationships are examined using correlation and independence methods. Relationships of predictor variables to the response variable are also investigated visually. We also introduce variable importance, an rpart method that will allow us to efficiently determine optimal variables to use for splitting a decision tree.

### Exploratory Data Analysis

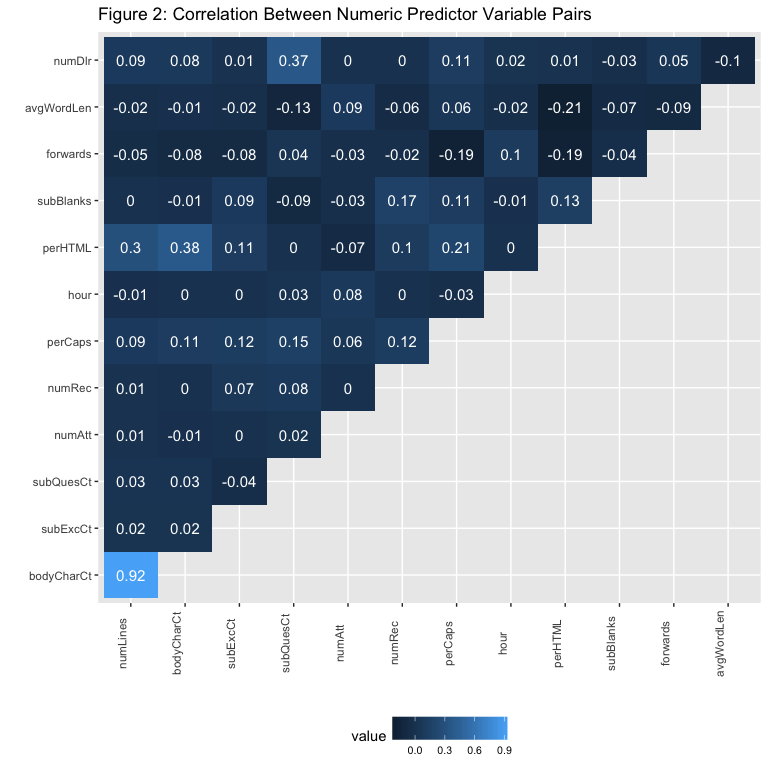
A preliminary evaluation of the dataset found missing observations in 303 unique rows. The availability of values for the following predictor variables was incomplete for: \* subSpamWords \* subQuesCt \* subExcCt \* subBlanks \* numRec \* noHost \* isYelling

Rather than discard over three percent of the dataset, we undertake imputation using random forest regression and classification methods for both numeric and categorical predictor variables.



Our dataset is unbalanced, with 2,371 (26%) spam emails, and 6,674 (74%) valid emails. This imbalance in the dataset could introduce higher false negative rates for our analysis task.

#### Explanatory Variable Relationships

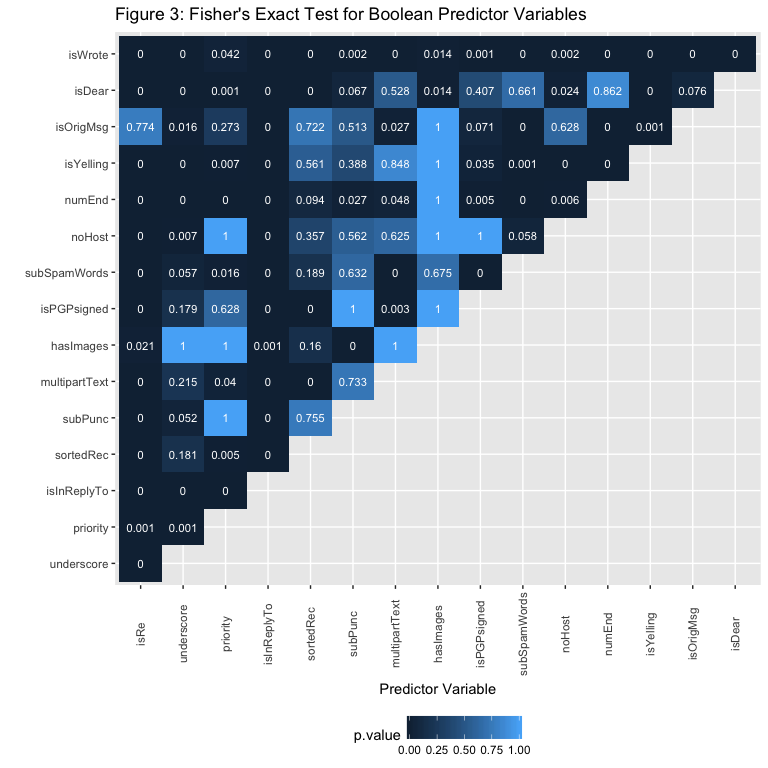


Working with our imputed dataset, the correlation matrix in Figure 2 reveals several significant positive correlations between numerical predictors:

* numLines and bodyCharCt
* perHTML and bodyChartCt
* numDlr and SubQuestCt

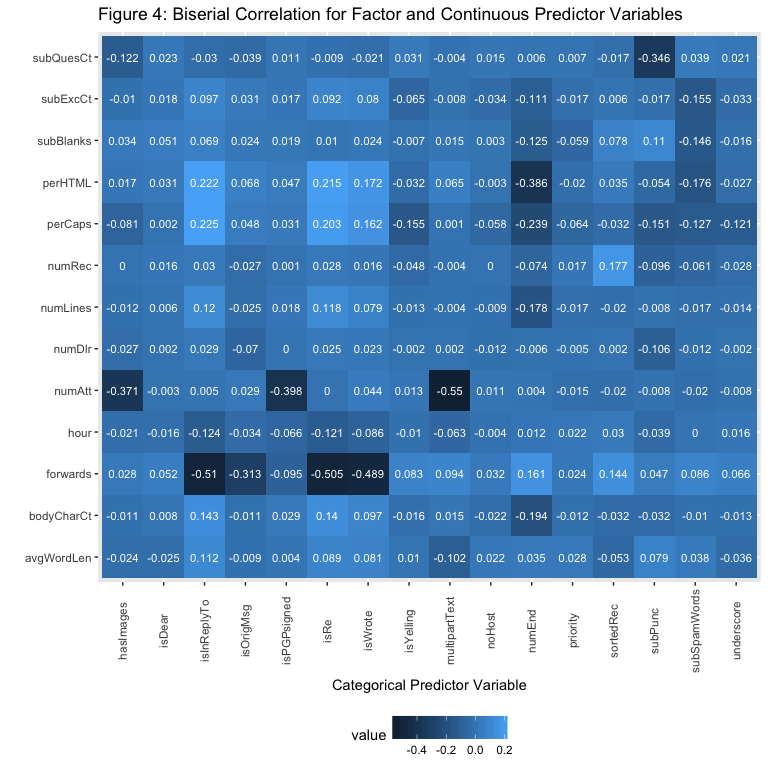
The higher correlation between predictor variables could cause these variables to be collectively overweighted in modeling, as they may not be fully independent from one another. Indeed, bodyChartCt and numLines, which represents the number of distinct lines in the body, are nearly similar variables. However, given we are using recursive partioning with rpart, collinearity issues are usually rectified naturally. The default splitting algorithms in rpart are greedy and will select the most important variable for classification if similar variables are found. We explore this further in the results section.

Given our dataset contains 16 boolean predictor variables as factors, we display a Fisher’s exact p-value matrix in Figure 3. This figure shows the resulting p-values for factor variables only, as correlation is not an appropriate metric for analyzing nominal relationships. Fisher’s exact test allows us to nonparametrically examine the association between our categorical variables. It also allows us to account for small numbers of observations for certain variable frequency counts.



Lower p-values indicate we reject the null hypothesis of random association. Obviously, significant non-random dependence between factor variables exists. This makes logical sense for variables such as isWrote, which indicates if an email is electronically scribed. Since the majority of emails are electronically scribed, we can assume this variable is not as important for predicting spam or valid. Other interesting relationships such as the independence between priority and noHost indicate that these variables may be useful separately for analysis.

In order to be complete, we also visually inspect the biserial correlation between factors and continuous variables. Given our factors are all dichotomous variables, biserial correlation is an appropriate measure to use when exploring relationships with continuous variables. Upon visual inspection, we are able to establish some common sense relationships.



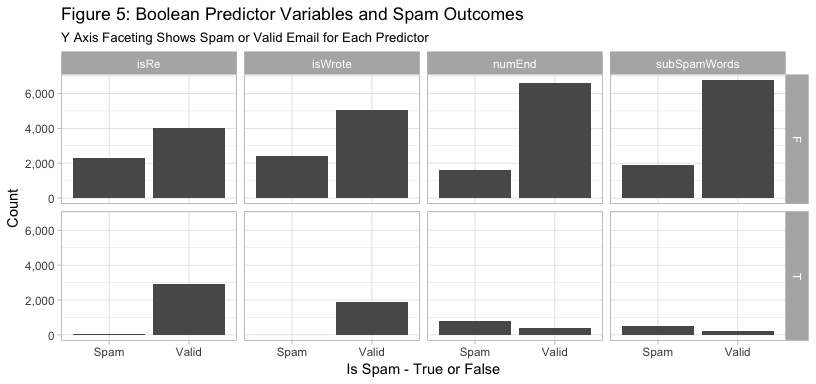
For example, the number of forwards is negatively correlated with isInReplyTo. This makes sense as replies typically do not contain many forwards. Additionally, the number of attachments is negatively correlated with the boolean multipartText. Again, multipart text messages typically do not contain attachments. In general, we see stronger negative relationships when investigating the correlation between factors and continuous variables.

In summary, highly correlated predictor variables could be extraneous to our objective of predicting whether or not an email is spam. To address predictor and response relationships, we visually inspect both continuous and factor variable relationships with the isSpam response variable in the next section. Additionally, we establish variable importances using the rpart package.

#### Response Variable Relationships

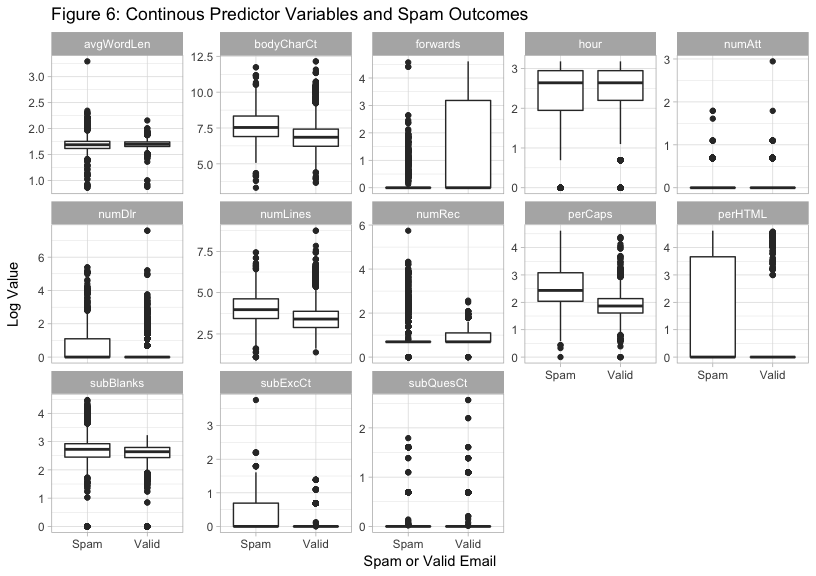
As described earlier, we know that the majority of observations in the overall dataset are classified as spam. However, we can visualize spam and valid email separation for factor and continuous predictors using different plotting techniques.

First, we inspect five factor variables visually to determine viability for spam prediction: isRe, numEnd, subSpamWords, and isWrote. The variable numEnd indicates whether or not the “from” email prefix ends with a number. For example, “[Greer5769@yahoo.com](mailto:Greer5769@yahoo.com)” would be classified as true for numEnd.



The variable subSpamWords is a boolean that is true when a known spam word is contained in the subject. For instance, the word “viagra” would trigger a boolean value of true for subSpamWords. The majority of spam occurs when the four factors above are set to false. On the other hand, we know that mostly valid emails occur when isRe and isWrote are set to true. There are occurences of spam emails in both boolean statuses for factor predictor variables numEnd, and subSpamWords. However, the true cases of each predictor variable have more cases of spam than valid email. Given the splits above, we can see how a decision tree could split on different categorical variables in order to classify an email message as spam or valid.

We can easily identify separation of classes for numeric variables by looking at log values in a box plot. We inspect each numeric value in Figure 6.



We can visually inspect continuous predictors in order to identify potential variable splits. The predictor variable forwards, which quantifies how many times an email has been forwarded, shows a more concentrated distribution of values in the third quartile for messages that are valid. The predictor variable perCaps shows a larger interquartile range for spam. We also know from the boxplot that the median perCaps value for spam messages is higher than for valid messages. Also, roughly 75 percent of valid messages have a perCaps value lower than the median perCaps value for spam messages. PerHTML (percent of HTML in email body) also provides for a decent classification variable, with the majority of the log range occurring for spam predictions.

Examination of these predictor variables gives some idea of how we should expect the rpart model to determine splits. However, we can also expect that some of the lower correlation variables might be involved in the decision of spam versus valid, perhaps providing some finer detailed distinctions between the two classes. We use rpart in the next section to determine which variables are the most important for splitting.

#### Variable Selection and Model Comparison Setup

We explore variable selection and the effect of rpart’s control parameters on classifying spam emails by fitting a default rpart model and comparing it to a separate, optimized rpart model. For the optimized rpart model, we explore four different parameters: complexity penalty (cp), minsplit, maxdepth, and the splitting criteria. Table 2 provides a description for each of these parameters.

Table 2 \* cp - A scaled complexity penalty that ranges from 0 to 1. In a classification setting, cp is compared against the error rate relative to a previous split. Any split that does not decrease the overall lack of fit by cp is not considered. Default is 0.01. \* minsplit - the minimum number of observations that must exist in a node in order for a split to be attempted. Default is 20. \* maxdepth - The maximum depth of any node of the final tree, with the root node counted as depth 0. Default is 30. \* splitting criteria - gini or information. Gini utilizes the gini index to optimize split points, information uses entropy and information gain. Default is Gini.

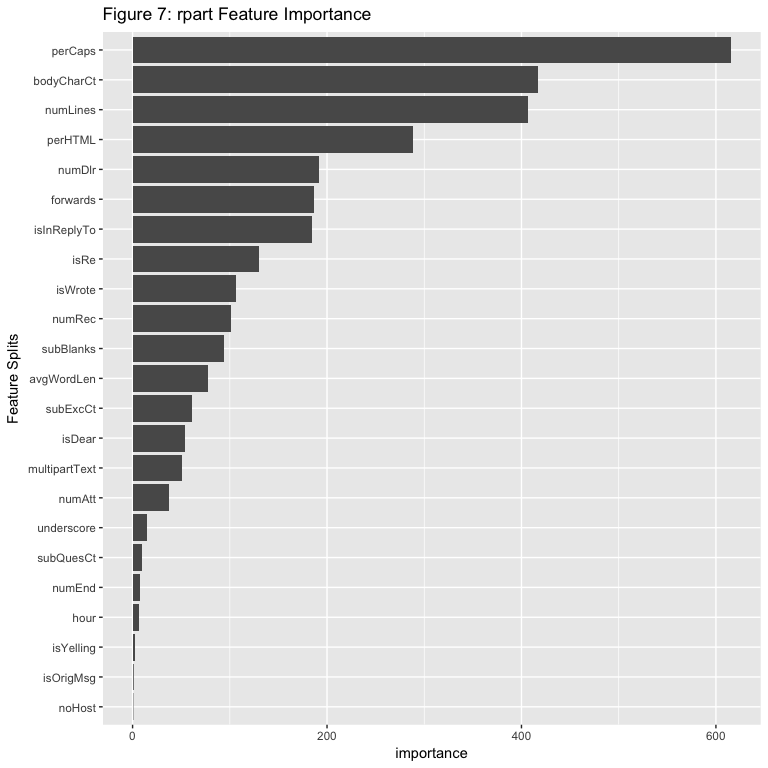
Utilizing the full listing of variables in our email dataset can lead to overfitting, however, decision trees allow us to find the best variables for splitting while pruning extraneous variable splits. Applying the first three parameters in table 2 typically reduces the size of the final tree. This reduction in size can help with model generalization to the test dataset.

The rpart package contains other control parameters used primarily for exploratory purposes (xx reference). However, one point of clarification regarding the parameter xval is warranted. The parameter xval allows a user to optimize the cost penalty (cp) for a tree in a k-fold cross-validation setting. Given rpart does not allow for the tuning of multiple parameters simultaneously, we instead rely on the mlr package (xx reference) for cross-validation and tuning.

We use 80 percent of the email data for training and 20 percent for testing. Spam represents roughly 25 percent of the emails in our original dataset. Therefore, we stratify the observations in our train and test datasets to maintain the original spam/valid ratio. For example, of the 20 percent of data held for testing, 25 percent is made up of spam.

In order to explore the effect of rpart parameters mentioned previously, we first fit an rpart model on the training data using all 29 features and default model parameters. Default parameters for rpart consist of a minsplit of 20, a complexity paramer (cp) of 0.01 and a maxdepth of 30. The Gini index is used as the splitting criterion by default.

We retain this initial fit on the full training set in order to evaluate the model’s generalization capabilities on the test data set. We call this model our “base” model, and use this terminology going forward. However, we are also interested in the variables rpart considers the most important for splitting. We can easily identify these variables with rpart package methods. Figure 7 shows the variables our base model considers the most important for classification.



At the top of the variable importance list is perCaps, or the percentage of capital alpha characters in the body. Importance in Figure 7 is a weighted sum of improvement in impurity for each variable split. The perCaps variable overshadows all other variables from an importance standpoint. Indeed, we saw good separation of spam and valid email in the previous section for this variable. We are able to classify 77 percent of all spam messages based on a split value of 13 percent for perCaps. In this case, the base model predicts non-spam when a message contains less than 13 percent capitals. Additionally, we see email from addresses ending in numeric (numEnd) and priority provide little value for spam classification.

#### Hyperparameter Optimization

Now that we’ve established variable importances and fit our base model, we tackle the objective of exploring key parameters and fitting an optimized model. For comparison purposes, an optimized rpart decision tree model is also fit on the training data set. Optimization is achieved by exploring a discrete list of the four parameters of interest. The grid search method is used in conjunction with ten-fold stratified cross-validation.

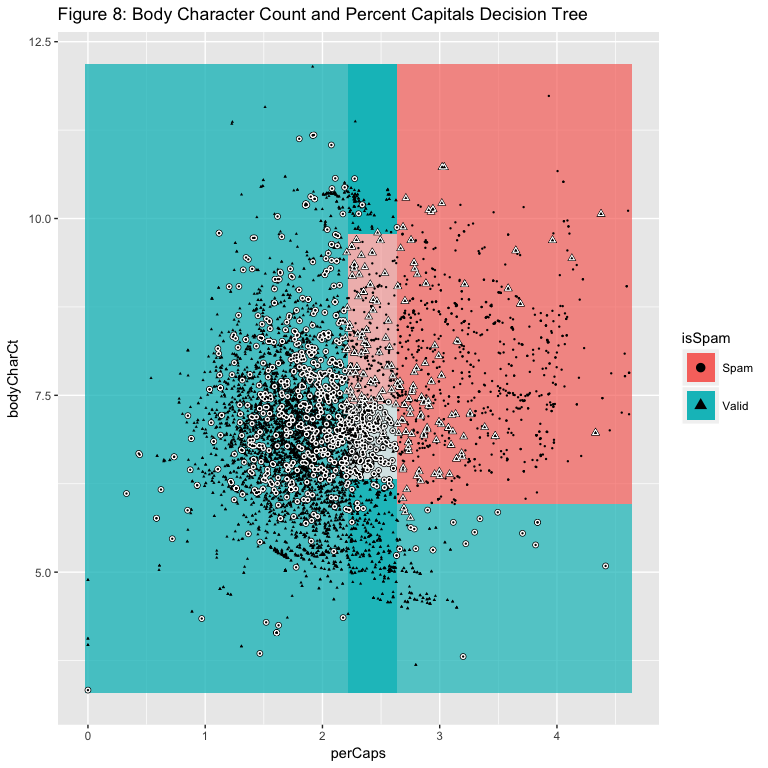
We seek to maximize the true positive classification rate where “spam” is our positive class. We do this by selecting area under the ROC curve (AUC) as our performance metric when tuning hyperparameters. A false positive means an important email may end up in spam or deleted. A false negative means the user may experience unfiltered messages that should be in the spam folder. We consider the former situation a more severe model error. The mean AUCs for all models and associated parameters are compared and the model with the highest cross-validated AUC is chosen as our optimized model.

The base rpart model with default parameters is then subsequently compared to the optimized model given the test data set. We explore model performance and comparisons in the next section. We consider final model comparisons using a holistic set of performance metrics including AUC, mean misclassification error, false positive rate, and false negative rate.

## Results

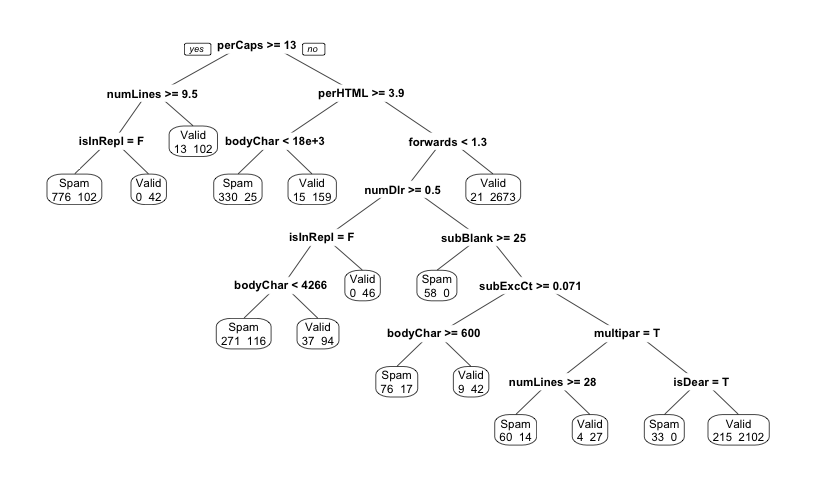
### Base Model Results

Our base model lists perCaps as the most important variable to split in Figure 7. In second place is BodyCharCt, which represents the number of characters in the body of the email message. Given these variables are so important, we visually inspect an rpart model fit on our training data set using only perCaps and BodyCharCt.



We represent values on a log scale in order to provide a clean visualization of classification regions. The observations outlined in white are misclassifications and the color boundaries represent the outcomes spam and valid. The lighter areas of the chart represent lower probabilities for a given class. Obviously, the ligher regions represent areas of higher misclassification. Additionally, utilizing only these two variables, our base model struggles with false positive rate. This indicates our model needs more complexity. However, these two variables alone provide for good separation as part of a more complex model.

Thus, we leave it up to rpart and its default parameters to create a decision tree containing 14 splits represented in Figure 9. This default rpart decision tree was created in the previous section and is considered our base model. We fit this default model on the entire training data set.



Of particular note is the base model’s usage of bodyCharCt. It is used three times in the splitting process for the training data. Additionally, we can see quite a bit of misclassification in the model on the training set, particularly a prevalence of false negatives.

We utilize the test data set of emails to produce a confusion matrix and determine model generalization performance in Table 3.

## predicted  
## true Spam Valid -err.-  
## Spam 403 76 76  
## Valid 77 1313 77  
## -err.- 77 76 153

## auc mmce fpr fnr   
## 0.95038750 0.08186196 0.05539568 0.15866388

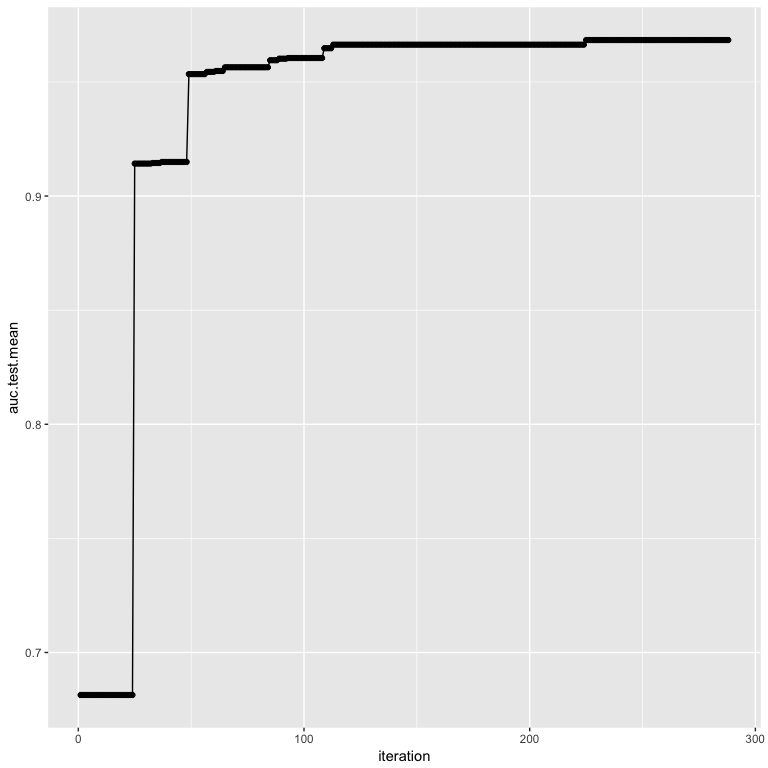
In order to address overall misclassification, we include MMCE, or model misclassification error rate in our metrics set. As seen in the confusion matrix, the base model struggles with false negatives. In total, 76 observations in the test set out of 479 spam records are misclassified for a false negative rate of 15.8 percent. MMCE overall is high at almost eight percent. We explore if we can improve on these metrics by optimizing rpart’s hyperparameters.

### Optimized Model Results

As stated previously, we explore the complexity parameter, minimum split for each node, maximum depth of the tree and the splitting criterion via a grid search. The parameter list is given in Table 4 below:

Parameter Search Criteria *complexity parameter (cp) -0.001, 0.01, 0.1, 0.2, 0.5,* minsplit - 1, 5, 10, 15, 20, 30 *maxdepth - 1, 5, 10, 15, 20, 30* splitting criterion - gini, information

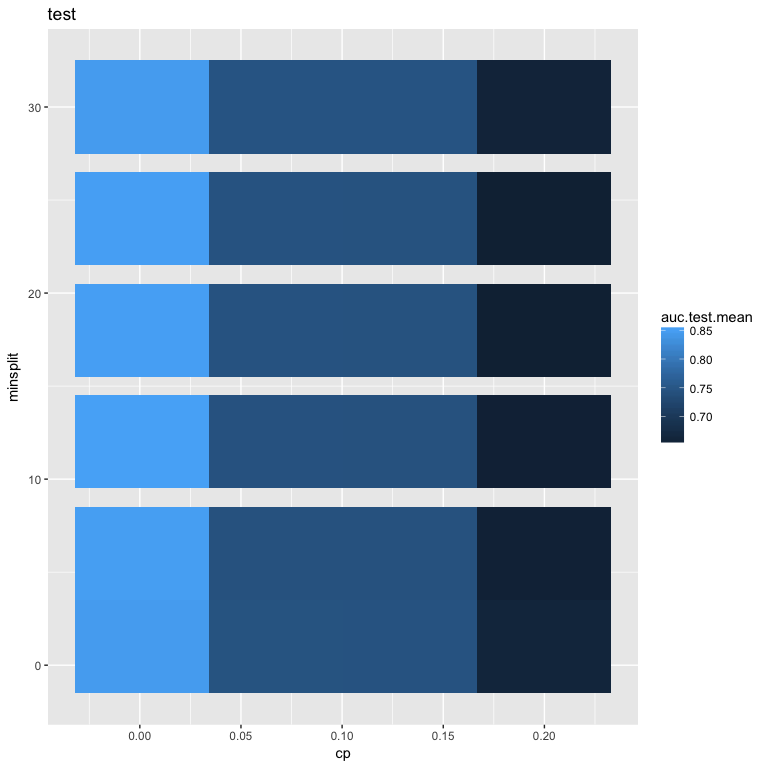
Each combination of parameters is cross-validated on 10 folds of the training data set using AUC as the performance metric. After the grid search and associated cross-validations are complete, the model with the best AUC performance has a complexity penalty of 0.001, a minimum split of 10, a maximum tree node depth of 15, and uses information as its splitting criterion. We can see the optimization path for our rpart model clearly by viewing the AUC score in sorted order.



Optimization begins to stall just under an AUC of 0.97 for our training data. Our optimized tree is much larger than our base model. In fact, 76 total splits are used for our optimized tree. This is because the optimal complexity penalty is set lower than the default (0.01) at 0.001. This may introduce risk for overfitting our test set, which we will explore shortly.

Of the 29 models fit during the tuning process with a resulting cross-validated AUC of greater than 0.96, all have complexity parameters of 0.001. Additionally the maxdepth for a decision tree node is always 10 or higher. 14 out of the 29 models use the information splitting criterion and the majority of models have a minsplit value of 10 or higher. Optimization results favor more complex models with medium to deep nodes. A single value result indicates the complexity parameter dominates when it comes to maximizing AUC results for our analysis task. The splitting criterion is nearly a toss up between Gini and information.

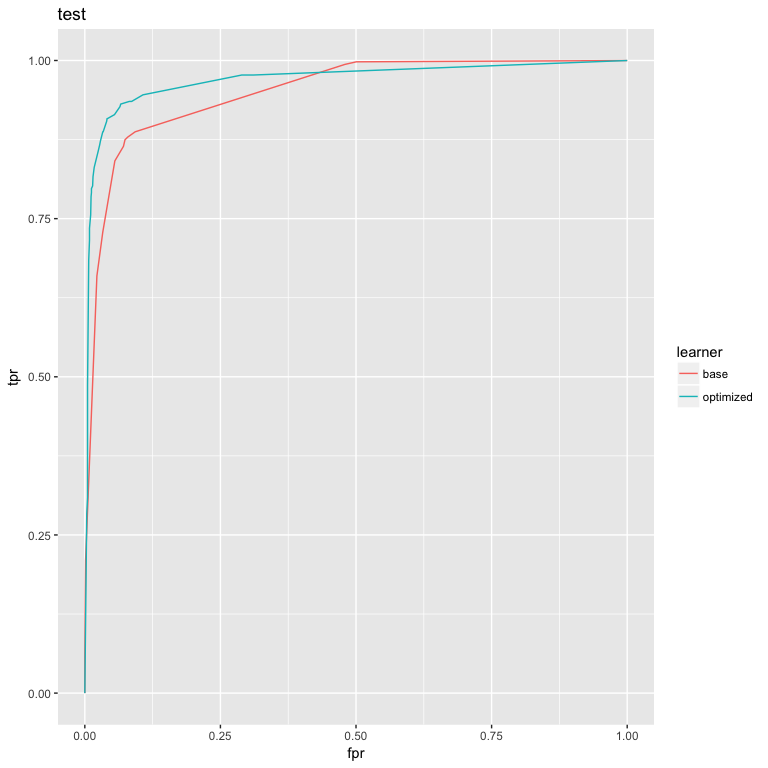
We can easily see the complexity parameter’s dominance in Figure XX below, where the mean AUC across validation splits is maximized as long as the complexity parameter remains small.



After thoroughly exploring rpart parameters, we turn to generalization performance. Given an AUC of 0.95 for our base model, we fit our optimized model to the same training data set of emails.

|  |  |  |
| --- | --- | --- |
|  | optimized | base |
| auc | 0.967 | 0.950 |
| mmce | 0.054 | 0.082 |
| fpr | 0.034 | 0.055 |
| fnr | 0.113 | 0.159 |

Our optimized model outperforms the base model on our key metric: AUC. Given our base model struggled with false negative rates, we also include false positive and false negative rates for the classification of spam email on the test data set. Our model significantly improves the false negative, false positive and misclassification error rates as well. However, no analysis using AUC would be complete without viewing an ROC curve for comparison.



We can now visually attest our optimized rpart model outperforms the base rpart model using default parameters.

## Conclusions and Future Work

Through detailed data exploration, extensive parameter investigation, and tuning, we are able to improve upon rpart’s default settings in an email spam classification setting. We determined that the complexity penalty in rpart dominates when optimizing for AUC given our analysis task. Further, we were able to effectively generalize to a test data set using a relatively complex rpart decision tree.

Additional improvements could be achieved given a more balanced data set. Oversampling of spam emails could reduce the tendency for rpart models to give false negative results. Additionally, empirial investigation of decision thresholds and learning curves could assist in optimizing the classification criteria and amount of training data necessary to fit an optimal rpart model.

Supreme Court Justice Potter Stewart is famously credited with the quotation: “But I know it when I see it.”[1] In the context that he was in, Justice Stewart was referencing a film that was thought to be outside the standards of decency accepted in the era of the 1960’s. In the present day, most internet and mobile phone users could also apply it to the subject of spam, or commercial, unsolicited e-mail: they know it when they see it.

The borders of so-called “Spam” e-mail are hard to delineate versus e-mail accepted by users. It would be easy to set extremely broad borders - that might include sweeping up any commercial e-mail for disposal of, before delivery to the recipient. It would be easy as well to be too lenient - filtering only the most egregious examples, and leaving the blatant examples of spam to the refuse.

Early efforts at filtering spam were centered on the assumed validity of identity of e-mailers: recipients were expected to “white list” their favored senders, and the senders validity would be assumed valid if the e-mail was shown to be from the same address. In the last 20 years, many major e-mail handling systems have moved from this type of certainty label, to an ensemble of factors that can be labeled with a final “TRUE” or “FALSE” to the e-mail’s spam content indication. Additional aide is given in creation, or validation of this level by the input predictor set.

Google, via it’s gmail product, as well as other providers, have had an ample data set over the past several years. All have searched for algorithms that reduce the number of false positives (labeled as spam, when it’s actually valid) and false negatives (labeled as valid, when it actually is spam).

A 2012 article in the Journal of Economic Perspectives[2] cites that up to 3% of the 50 billion pieces of spam e-mail sent each today are successful in reaching the recipient. The authors of the paper develop a cost of $20 billion annually to American consumers from spam.

The value of intercepting spam communications, across mediums, is also growing. Most mobile phone numbers have experienced the spoofed-number attack, where a number similar to your own places a call with a robot caller on the opposite end. A number of smart phone applications are currently trying to address this problem by machine learning. Likewise, SMS text messaging spam also provides a risk to users of mobile phones. In both cases, the originator of the call or message is not the number that is listed on the phone, but is a false or spoofed phone number used with the intent of deceiving the recipient. Phishing attacks via SMS skirt the boundaries of security in several ways. First, traditonally there have not been products to address it. Second, the perceived higher level of intimacy by users given to SMS versus e-mail seems to reduce the users’ awareness of threats. Third, the smaller screen may reduce users’ awareness to visiting spoofed websites.

As e-mail filtering continues to move in the direction of machine learning, development of new variables is likely an area for improvement in accuracy and performance. This could move further into the area of natural language processing, and perhaps more specificity to different languages. One can also imagine moving some of the service closer to the user, to further allow for customization. Of course, spam senders will also try to defeat the measures equally as much - and one can imagine training an adversarial model to allow for spam e-mail to pass through filters and land in your inbox. Engineers and developers will need to continue to walk a fine line - avoiding false positives (thereby filtering legitimate e-mails) with ensuring that users’ experience and time is not absorbed by spam e-mails.

## References

<http://spamassassin.apache.org>

1. Nolan, D., Temple Lang, D. DATA SCIENCE IN R: a Case Studies Approach to Computational Reasoning and Problem Solving. CRC PRESS, 2017.

<https://cran.r-project.org/web/packages/rpart/rpart.pdf> <https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf> <https://stats.stackexchange.com/questions/191842/how-does-the-complexity-parameter-correspond-to-the-number-of-splits-in-cross-va>