



Spam Email Detection Data Mining Methods Final Project Report

CIDM-6355 Professor: Dr. Liang Chen

Team C Members:

Cecilia Foy
Rex Herndon
Brandon Koch
Mohammdsalim Shaikh
Jordan Unfred

Executive Summary

Email has become a nearly universal form of communication and identification over the past few decades, and it's seemingly tied to just about every single service we use on the web, from newsletters to applications, to account registrations, and so many more options. "Despite the growing popularity of messengers, chat apps, and social media, email has managed to remain at the center of digital communication and continues to grow every year" (Damar, p. 184).

This report was created with the intention to assist with the development of spam filters, and their purpose of reducing spam appearing in people's inboxes. Our goal is to identify the most efficient data mining and classification method for creating an effective spam filter. The data provided below offers insight into the different characteristics of spam/legitimate emails in our "Spambase" dataset. Our data analysis programs, RapidMiner and R/RStudio were used to create different data mining models, run predictive analysis on the testing portion of our dataset, and perform model evaluation to compare and identify the best model/data mining method for our intended goal. Models used such as Decision Trees, Neural Networks, Naive Bayes, and Logistic Regression provide detailed information about spam detection and prediction.

From our efforts, we have concluded that our Naive Bayes model in R/RStudio (Model 4) gave us the best results compared to every other classification method we used in our analysis based on overall accuracy, recall, and precision, especially in determining true values (ie. - where "*class*" = 1, which is our label for a spam email) between the training and testing sections of our dataset. This model was chosen, since it had the highest precision percentage for identifying spam emails compared to our other 7 models. Precision is a more important measure when the cost or stakes of creating a false positive is greater than the cost or stakes of creating a false negative. If our spam filter sent a false positive (ie. - a spam email to a user's main inbox) and the user clicked on it, the consequences could be range from annoying to devastating, since these spam emails could retrieve information about a particular user and either leak it publicly if it is personal information or sell this information to companies with malicious or immoral intent, such as targeted advertising or phishing. If these spam emails were also a phish that the user fell for since it would appear as a legitimate email, these senders could obtain usernames and passwords, financial information, or other sensitive personal information to use for blackmail, harassment, or identity theft.

Based on our findings, we would like to reach out to larger email providers, such as Google (Gmail), Yahoo/RocketMail, or ProtonMail, and collaborate with them to integrate our findings into their existing spam filters. The war against spam and phishing emails is one that we can never truly win, as companies work tirelessly to find different ways to reduce or eliminate spam emails from user's inboxes every single day. We hope that our contributions through our data mining methods defined in this report can help with the reduction of spam in user's inboxes.

Some limitations with our data were with the overall size of our data (4601 records) and with the age of the dataset itself. This might make our data irrelevant from some perspectives, but for the case of this report, the data that we have is still a great starting point for the development of our data mining models throughout the entire data mining process, and this can be backed up by the fact that this dataset is still cited in several peer-reviewed articles, with several of the most recent articles and reports being published as recent as 2019.

Introduction

In 1971, Ray Tomlison, of ARPANET, was credited for inventing email as we know it today (Steinbrinck, 2021). The idea that connected computers could receive messages from each other has evolved into one of the most used communication tools in the world. With the rise of social media and other avenues to communicate, email still stands near the top. A study found that in 2023, more than four billion users were active on email. Compared to Facebook, there were 2.96 billion users at the end of 2022 (Kindness, 2023). Almost half of the world's population 8.1 billion are actively using email.

This also means there are significantly more spam emails sent to users. Another study found “Phishing is the most common form of Cyber-crime, with an estimated 3.4 billion spam emails sent every day” (Griffiths, 2023). This is a very staggering number once we consider there are four billion active users of email. With email usage being extremely common among both personal and business settings, one of the biggest challenges we have to face is minimizing and filtering out spam. “Personal users and companies are affected by Spam due to the network bandwidth wasted receiving these messages and the time spent by users distinguishing between Spam and normal (legitimate or ham) messages” (Guzella & Caminhas, p. 10206).

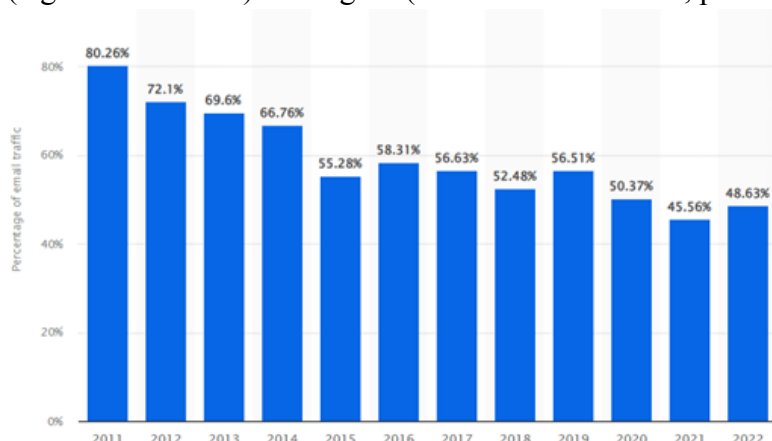


Figure 1 Details: Worldwide; Kaspersky Lab; 2011 to 2022 © Statista 2023

In addition, “Results from a survey [Rainie & Fallows] in March 2004, with over 2000 American email users, report that over 60% of them are less trusting of email systems, and over 77% of them believe being online has become unpleasant or annoying due to spam [4]” (Gomes, *et al.*, p. 690). Spam email, or junk mail, is an unwanted or unsolicited email sent to one or more recipients. “Spam messages accounted for over 45 percent of email traffic in December 2022. Russia generated the largest share of unsolicited spam emails in 2022, with 29.82 percent of global spam emails originating from the country” (Petrosyan, 2023).

These emails can be sent for a broad variety of reasons with varying degrees of severity, but these emails are often sent with malicious intent to phish for the recipient's data, such as passwords, debit/credit cards, phone numbers, or other personal and sensitive information. “These spam emails not only consume users' time and energy to identify and remove the undesired messages, but also cause many problems such as taking up limited mailbox space, wasting network bandwidth and engulfing important personal emails” (Yu & Zu, p. 355).

Spam email also can be used for advertising/commercial purposes or to install viruses or

malware on a user's device. "The United States has the highest number of spam emails sent daily as of January 2023. On average, about 27% of businesses worldwide experience four to six successful cyberattacks in a year, with 15% saying they have been the victim of more than 50 mass phishing attacks" (Damar, p. 201-202).

Although most users have seen spam in their inboxes, spam filters are often created by email providers or users to automatically filter unwanted messages. Google equipped Gmail with a user-guided learning mechanism and "as a result, the large number of Gmail active accounts (more than 900 million in 2015 [²⁸]) allows Gmail anti-spam filtering system to achieve a classification accuracy up to 99%" (Pérez-Díaz, *et al.*, p. 2). "Automatic email filtering seems to be the most effective method for countering spam at the moment and a tight competition between spammers and spam-filtering methods is going on" (Awad & ELseuofi, p. 173).

For this project, our plan included investigating how we can use data mining and classification to detect future emails as spam or legitimate in order to contribute to the reduction in phishing or unwanted emails in a given user's inbox. "Spam filtering is also increasingly considered as a benchmark for testing newly developed machine learning algorithms, not specifically designed for this problem (e.g. Camastra & Verri, 2005; Gadat & Younes, 2007; Qin & Zhang, 2008) " (Guzella & Caminhas, p. 10206).

Data Description

The dataset used was obtained from the UC Irvine Machine Learning Repository, the 1999 dataset includes over 4600 records/instances with the following attributes (Hopkins, et. al.):

- Number of Instances: 4601 (1813 Spam = 39.4%)
- Number of Attributes: 58 (57 continuous, 1 nominal class label)
- Attribute Information: The last column of 'spambase.data' denotes whether the email was considered spam (1) or not (0), i.e. unsolicited commercial email. Most of the attributes indicate whether a particular word or character was frequently occurring in the email. The run-length attributes (55-57) measure the length of sequences of consecutive capital letters.

Here are the definitions of the attributes:

- 48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the email that match WORD, i.e. $100 * (\text{number of times the WORD appears in the email}) / \text{total number of words in the email}$. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type char_freq_CHAR = percentage of characters in the email that match CHAR, i.e. $100 * (\text{number of CHAR occurrences}) / \text{total characters in email}$
- 1 continuous real [1,...] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_longest = length of the longest uninterrupted sequence of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_total = sum of length of

- uninterrupted sequences of capital letters = total number of capital letters in the email
- 1 nominal $\{0,1\}$ class attribute of type spam = denotes whether the email was considered spam (1) or not (0), i.e. unsolicited commercial email.

“The email consists of two parts, one is the body message and another part is called the header. The job of the header is to store information about the message and it contains many fields, for example, tracing information about which a message has passed (Received:), authors or persons taking responsibility for the message (From:), intending to show the envelop address of the real sender opposed to the sender used for replying (Return-Path:)” (Yu & Zu, p. 356).

Data Preparation

Regarding the data preprocessing and cleaning before the actual mining and analysis, a lot of the initial formatting was actually done by the original curators of the “spambase” dataset. The main preparation activities we had to do were done in order to work more effortlessly with RapidMiner (RM) and R/RStudio (R), our respective data analysis platforms. Even though the initial “spam base” repository gives us some level of information about how clean their data is, we still need to double-check the validity of this with our own data cleansing and preparation steps. This meant we ultimately needed to verify if our data contained any discrepancies or errors, if it contained any outliers, if there were any redundant or uninteresting attributes, or if any of our data needed to be recorded to account for qualitative attributes.

It is also important to note we performed nearly all of the pre-processing steps in Microsoft Excel as opposed to R/RStudio or RapidMiner due to it’s “low-code/no-code” solutions for moving columns, renaming attributes, and functions to check for outliers or null/missing values. We verified our pre-processed dataset(s) was imported correctly in our respective data analysis software through a brief look at the summary statistics of our data.

To start with, we had to consider how we were going to deal with any missing values. Ideally, we would just delete the rows that had missing information if we just had a handful of missing values. We could perform imputation by replacing the missing values with the mean, median, or mode values of the respective attribute since that could fit well into the context of our data. However, taking this data into one of our data analysis platforms showed us that we, in fact, had no missing values on any of our rows and columns in the original dataset. In RStudio, this was done easily by importing the dataset and running the summary function to identify key summary statistics, including the number of missing values. This is demonstrated in the screenshot below, which shows the summary function on our dataset and a preview of the results. The R output showed the statistics for all columns, but only a handful is shown in the screenshot.

```

> summary(SpamData)
      class      capital_run_length_average capital_run_length_longest
Min.   :0.000   Min.   : 1.000      Min.   : 1.00
1st Qu.:0.000   1st Qu.: 1.588      1st Qu.: 6.00
Median :0.000   Median : 2.276      Median : 15.00
Mean   :0.394   Mean   : 5.191      Mean   : 52.17
3rd Qu.:1.000   3rd Qu.: 3.706      3rd Qu.: 43.00
Max.   :1.000   Max.   :1102.500    Max.   :9989.00
capital_run_length_total word_freq_make word_freq_address word_freq_all
Min.   : 1.0           Min.   :0.0000      Min.   : 0.000      Min.   :0.0000
1st Qu.: 35.0          1st Qu.:0.0000      1st Qu.: 0.000      1st Qu.:0.0000
Median : 95.0          Median :0.0000      Median : 0.000      Median :0.0000
Mean   : 283.3         Mean   :0.1046      Mean   : 0.213      Mean   :0.2807
3rd Qu.: 266.0         3rd Qu.:0.0000      3rd Qu.: 0.000      3rd Qu.:0.4200
Max.   :15841.0        Max.   :4.5400      Max.   :14.280      Max.   :5.1000

```

Figure 2 Details: Summary Statistics of “Spambase” Data in R/RStudio

This view also shows us any outliers or discrepancies, such as typos or incorrectly inputted values in our dataset. Ultimately, human judgment is needed to identify if a particular attribute had any discrepancies we needed to address, and our team of data analysts found everything to be within the appropriate thresholds and context of our dataset. As an example, we can see the range for the first attribute, “*class*”, only has values within the range “ $0 \leq \text{class} \leq 1$ ”. This is appropriate because the class is a coded variable where the only two values it can be are 0 and 1, where 0 is identified as a legitimate email, and 1 is identified as a spam email record. That same methodology and line of reasoning were applied to all attributes to come to logical and sensible conclusions on the parameters and thresholds of each attribute.

Regarding redundant or uninteresting attributes, our team came to the conclusion that all existing attributes were needed in order to fully understand this dataset. When working with an extremely broad concept like wanting to identify spam emails, only having attributes for the frequencies of specific words, character frequencies and attributes to identify the size of a particular email, can give us enough variety and uniqueness to develop our own “spam filter” based on the content of the message alone (even though more metadata, such as the sender or timestamp, could also help support our analysis).

We also needed to rename and reorder some of the variables to integrate better into our analysis platforms, RStudio and RapidMiner. This was primarily because our team encountered some issues with importing and manipulating the attributes or columns with the original dataset. Whenever our data analysis was working with any of the attributes that had a special character in the name, it caused syntax issues and errors with our code. The following table shows the initial attribute names found in the “spam base” dataset and what our team renamed them to in order to work with them properly in R/RStudio.

Initial Attribute Name	Renamed Attribute
char_freq_;	char_freq_semicolon
char_freq_(char_freq_parenthesis
char_freq_[char_freq_bracket
char_freq_!	char_freq_exclamation
char_freq_\$	char_freq_dollar
char_freq_#	char_freq_pound

One other important step we did in our data preprocessing was creating the splits between the training and testing data for the analysis methods that required a split dataset (such as our Decision Tree models). Our team decided to perform the holdout method where nearly 70% of our dataset was reserved for training, and the remaining 30% for testing and understanding the effectiveness of our predictive modeling. In order to randomly split the dataset, we created an additional “*rand_num*” column in our “spambase.csv” file to generate a random number between 0 and 1 for every record in our dataset. The dataset was then filtered by our “*rand_num*” column in order to perform simple random sampling and get randomized distributions of spam and legitimate emails in both our testing/training datasets. The breakdown for our splits is shown in more detail in the table below.

	Total Records	Testing	Training
Amount	4601	690	3911
Percentage	100%	30%	70%

Data Modeling

Using the dataset above, we sought to understand how the language within the email can be detected as spam or as a regular email, which is why we created several different models for our analysis. The following is a list of models we created and evaluated with our “Spambase” dataset:

- **Model 1:** Decision Tree (with RapidMiner)
- **Model 2:** Decision Tree (with R/RStudio)
- **Model 3:** Naïve Bayes (with RapidMiner)
- **Model 4:** Naïve Bayes (with R/RStudio)
- **Model 5:** Logistic Regression (with RapidMiner)
- **Model 6:** Logistic Regression (with R/RStudio)
- **Model 7:** Neural Network (with RapidMiner)
- **Model 8:** Neural Network (with R/RStudio)

Also, we needed to look for and identify what metrics were needed to evaluate our model, which will be covered in the Model Evaluation section of this report. “Discarding a legitimate email is of greater concern than classifying a spam message as legitimate. This means that high SP [Spam Precision] is particularly important” (Clark, *et al.*, p. 704). In the end, accuracy, recall, precision, and lift charts were used to evaluate all the models and select the model with the best performance. We used RapidMiner and R Studio for classification model building and testing.

Model 1: Decision Tree (with RapidMiner)

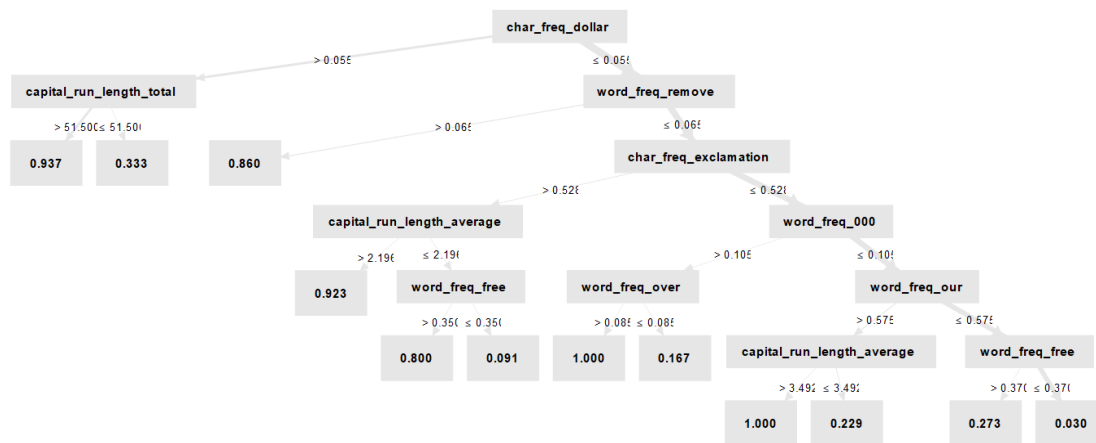


Figure 3 Details: Decision Tree Graph in RapidMiner

Using the preprocessed dataset Spambase training set and testing set described in Data Description, a Decision Tree model in RM was utilized as a supervised learning algorithm to analyze the attributes. The datasets were reviewed to verify the import configuration. In the training set, the attribute “class” was designated as the label attribute. In the testing set, the attribute “class” was designated as the prediction attribute. The parameters in the Decision Tree used were *least_square*, maximal depth 7, minimal gain 0.05, minimal leaf size 5, minimal size for split 5, number of preparing alternatives 3. The three attributes that had the most significant appearance in the spam emails were “capital_run_length_average”, “capital_run_length_longest” and “capital_run_length_total” (for more details see Appendix 1 and 2). With this valuable information, a filter could be created noting those emails with an “x” number of capital letters/words should be considered spam. The next closest attribute “word_freq_you” had a significantly lower average (for more details see Appendix 3 and 4).

Model 2: Decision Tree (with R/RStudio)

In R/RStudio, we started by importing the training and testing datasets we created from the previous data preprocessing steps as separate variables, and then we verified if our data was imported correctly with a few different R functions. We used “*names()*” to check if all of our column headings appeared, “*dim()*” to identify if the correct dimensions appeared on our rows

The only pre-processing steps we have done specifically within R/RStudio in these datasets was removing the “*rand_num*” column, since the only intention we had with making this column was to create the splits between our training and our testing dataset. Afterwards, we used the “*party*” package in R to create our decision tree model along with the “*ctree*” function.

Below is our initial output without any initial pruning or parameter adjustment with our decision tree model. There is a text output of this as well, and will be included in the appendix of this report (see Appendix #2 in Model 2).

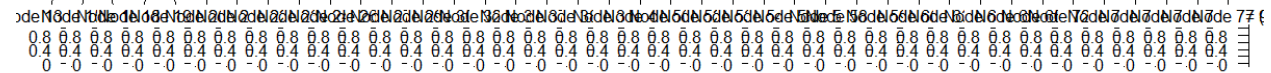


Figure 4 Details: Initial Decision Tree Graph in R/RStudio

From a first look at this graph, we can see that, even though we have a larger dataset to work with, the output is too detailed and cluttered to identify any sort of conclusions at a glance if we have a given record, which was one of the main benefits of a decision tree model. To go further in-depth, we created three different iterations of our decision tree model. The “*cree*” function in R/RStudio had two different parameters we will be adjusting in order to make comparisons with our model - “*mincriterion*” and “*minsplit*”. The “*mincriterion*” parameter specifies how pure a particular leaf node needs to be. With this attribute, our intentions with our model were to have our leaf nodes split up to be as pure as possible in order to ensure a higher level of confidence with our predictions. For each of our three models, we set this parameter at “*mincriterion = 0.99*” where our leaf nodes each have a 99% purity in the fact that they result in either spam or legitimate emails. There is no feasible reason to study different decision tree models that have a lower purity threshold, since models that give us impure nodes are not reliable enough for future predictions or educated decision making.

Our second parameter, “*minsplit*”, was the one we adjusted between our three models. This parameter describes the minimum amount of records our model will take to create a split node. In general, this attribute gave us the ability to prune our model by minimizing the amount of branches and leaf nodes created, with less decisions to focus on in our model and a higher support level for each split node. It’s important for our model to still have a certain level of detail so it can still be feasible for future predictions, but not so detailed that it overfits with our model and it can not be feasible for predictions outside of our training dataset (even though the model may perform very well with our training dataset). Each of our model parameters are described below:

- SpamTree1 \leftarrow mincriterion = 0.99, minsplit = 1000
- SpamTree2 \leftarrow mincriterion = 0.99, minsplit = 1500
- SpamTree3 \leftarrow mincriterion = 0.99, minsplit = 2000

After creating each of our models, we were able to generate graphs for each of them, which are also shown below:

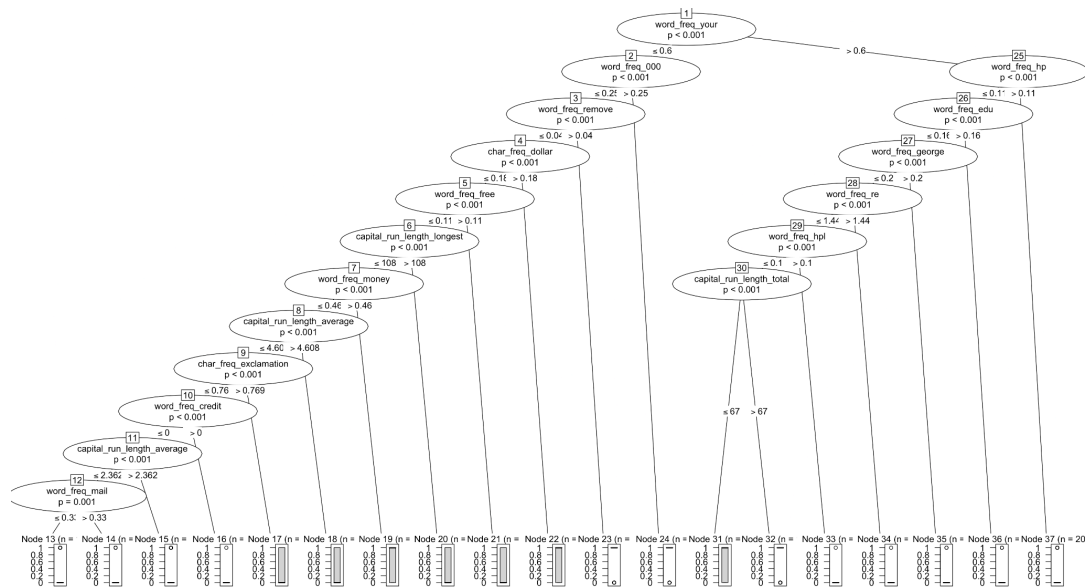


Figure 5 Details: SpamTree1 Decision Tree Graph in R/RStudio

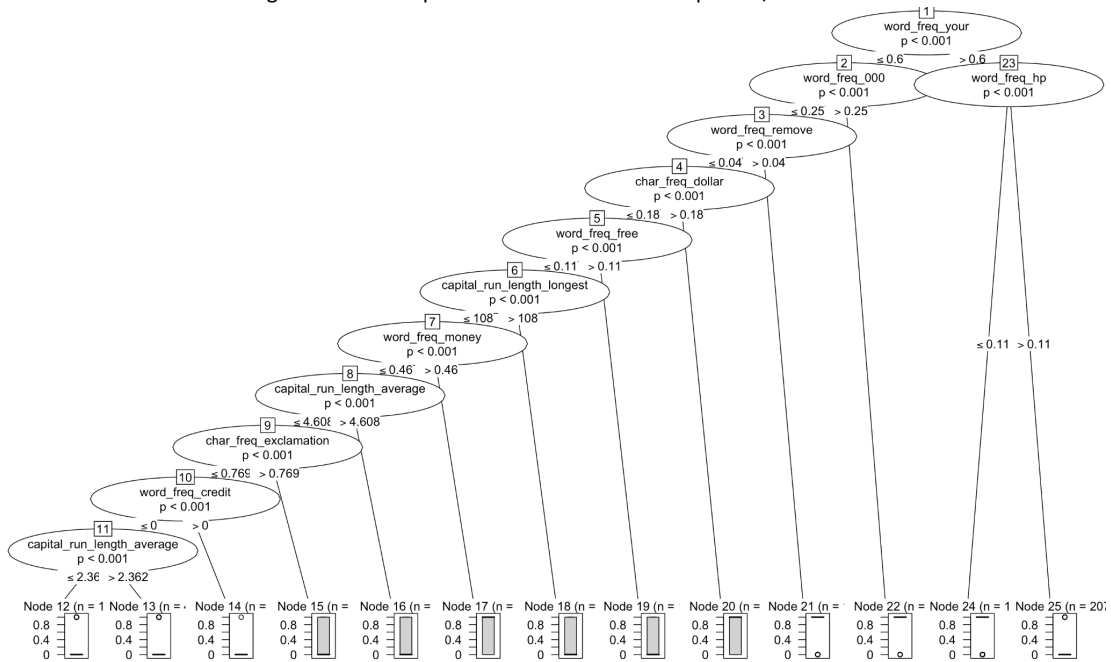


Figure 6 Details: SpamTree2 Decision Tree Graph in R/RStudio

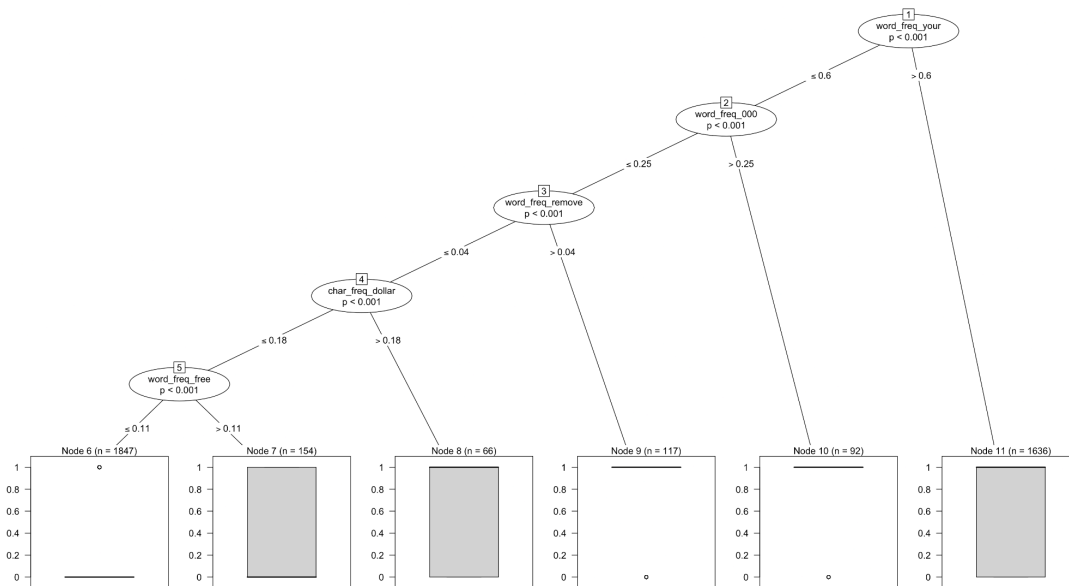


Figure 7 Details: SpamTree3 Decision Tree Graph in R/RStudio

As seen here, overall, these models were much more accessible to work with if we had to evaluate a particular record manually. We were able to confidently determine a result for a record that was not in our original training dataset from each of these models. In the Model Evaluation section, we will go further into the performance metrics for each of our models.

Model 3: Naive Bayes (with RapidMiner)

PerformanceVector (Performance)				
SimpleDistribution (Naive Bayes)				
Criterion	Table View Plot View			
accuracy	accuracy: 81.52%			
precision				
recall				
AUC (optimistic)				
AUC				
AUC (pessimistic)				
	true 0	true 1	class precision	
pred. 0	606	25	96.04%	
pred. 1	230	519	69.29%	
class recall	72.49%		95.40%	

Figure 8 Details: Naive Bayes Confusion Matrix in RapidMiner

The above indicates that the Naive Bayes model has the following values for spam messages:

Accuracy: 81.25%

Recall: 72.49%

Precision: 96.04%

The ratio of correctly predicted observation to the total observation is: 81.25%.

So, for spam messages in this model we have a total percentage of 72.49 % the correct total of actual positive numbers for the spam messages.

And 96.04% for the exactness values of spam messages. However, as our recall decreases, our precision increases because in addition to decreasing the negativity, we decrease the false positives too.

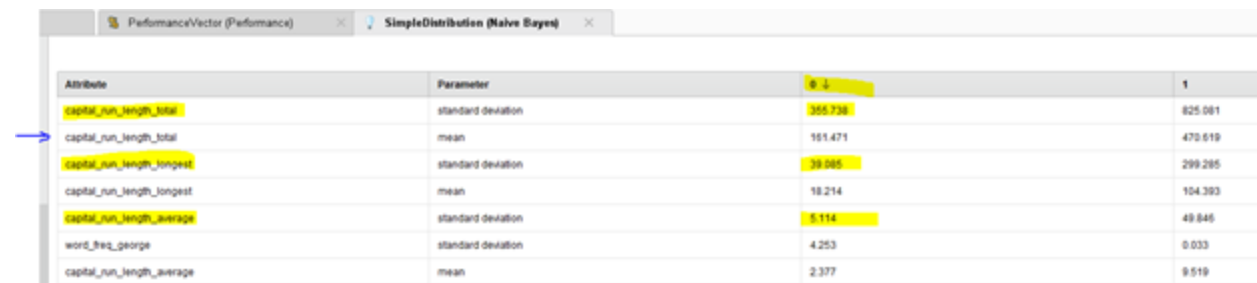
In this model we have relatively high recall which means that the filtering algorithm

returned an acceptable amount of the relevant results. So, which works better, recall or precision? To answer this question, we should know that it is not possible to maximize both these matrices at the same time, as one comes at the cost of the other.

For simplicity, we could use an F-Score metric, which is a harmonic mean of precision and recall. For problems where both Precision and Recall are important the equation as below:

$$F1 \text{ Score} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall}) = 2 * (96.04 * 72.49) / (96.04 + 72.49) = 82.62\%$$
 is the weighted average of precision and recall meaning about 83% takes both false positives and false negatives into account.

Simple Distribution:



Attribute	Parameter	0	1
capital_run_length_total	standard deviation	355.738	825.081
capital_run_length_total	mean	161.471	470.619
capital_run_length_longest	standard deviation	39.045	299.285
capital_run_length_longest	mean	18.214	104.393
capital_run_length_average	standard deviation	5.114	49.846
word_freq_george	standard deviation	4.253	0.033
capital_run_length_average	mean	2.377	9.519

Figure 10 Details: Simple Distribution View for Logistic Regression in RapidMiner

A Naive Bayes distribution table displays the ‘capital_run_length_total’ attribute as the number one (356) when it comes to spamming with an average (mean) value of 161. The second and the third-ranking goes to the ‘capital_run_length_longest’ and the ‘capital_run_length_average’ respectively.

Model 4: Naive Bayes (with R/RStudio)

In this analysis, A Naive Bayes classification model was utilized to predict spam in the dataset. To prepare for this model, we first had to import the data with a read.csv function and had to identify our function called “Spamtrain” as this was our training set. Then we had our second function with a read.csv called “SpamPredict” as this was our prediction set. At this point, we determined the dimension of the files using the dim function and then viewed a summary of the dataset. Furthermore, we installed the correct package in order to properly model the Naive Bayes model. This library included “e1071”. At this point, we were now ready to build our NB model and apply it to the prediction set. Once we received the results of the NB model and its predictions, we built a confusion matrix in order to determine the accuracy, precision and recall of the dataset. Overall, the results of these evaluation metrics offer a comprehensive understanding of the Naive Bayes model and how it performs against other models utilized by RapidMiner and Rstudio. Starting with the accuracy of the model, a result of 58.6% was provided. The precision of the model, which accurately predicts the legit messages resulted in 97.1% which is very precise. Moving on to the recall, this resulted as 56.7%, indicating the model may not capture all legit messages. Finally, moving on to the F1-score which consists of both precision and recall resulted in 71.0%, reflecting a trade-off between the two metrics. Overall, the NB model indicates certain strengths and weaknesses positively predicting the legit messages and spam messages.

Model 5: Logistic Regression (with RapidMiner)

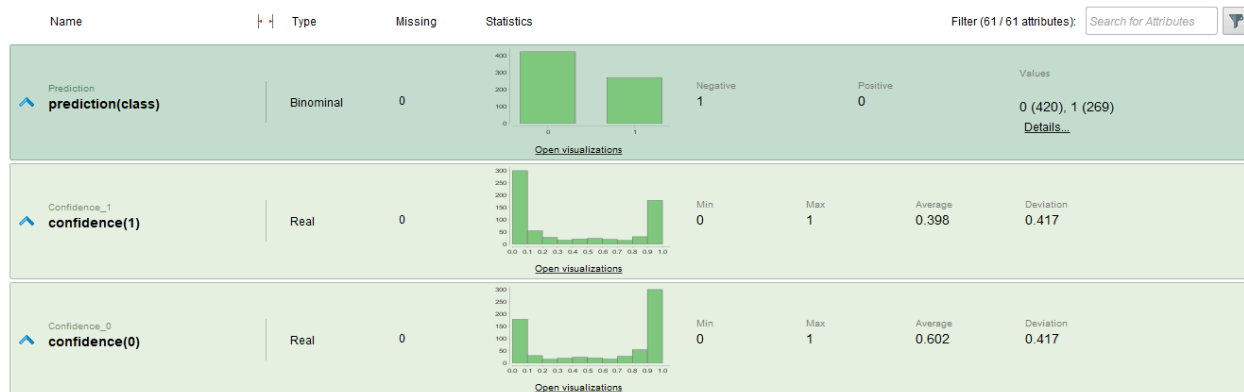


Figure 9 Details: Statistics View of Logistic Regression Predictions in RapidMiner

A logistic regression model was used to analyze the email security. This model was employed to analyze the thousands of email messages, aiming to distinguish between legitimate and malicious ones. The study reported the model used two critical indicators, "confidence(1)" and "confidence(0)," both represented as real numbers. "Confidence(1)" pertains to the model's confidence in an email being phishing or spam, with an average value of 0.398 and a standard deviation of 0.417. "Confidence(0)" represents the model's confidence in an email being legitimate, with an average value of 0.602 and a standard deviation of 0.417.

This offers insights into the model's performance. A lower "confidence(1)" value suggests a higher likelihood of an email being malicious, while a higher "confidence(0)" value indicates the email being genuine. The narrow standard deviation for both measures indicates consistent model performance. The logistic regression model's ability to weigh these confidence levels helps in improving the overall email security system.

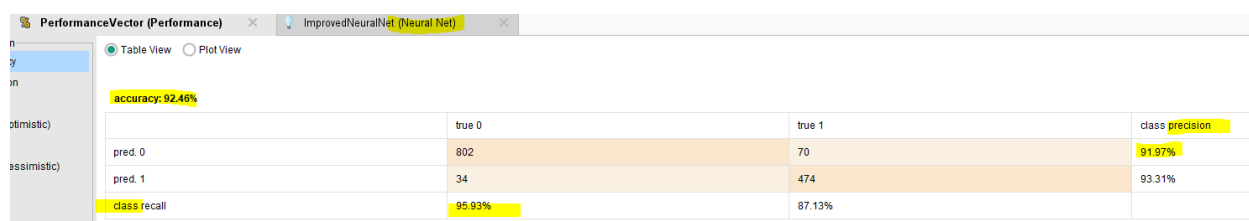
Looking at the confusion matrix for the spam base database, the accuracy result is 81.52%, Precision is 96.04% and recall is 72.49%. Here, we were interested in predicting True (that is 1) as the spam in any given email. The horizon dimension is for class while the vertical is for prediction. In this example 606 (True Positive) samples were predicted to be true and they did show up as spam. 519 examples (True Negative) show that they were predicted as False and they did not end up as spam.

Model 6: Logistical Regression (with R/RStudio)

Logistical regression was utilized in R and presented the user with several different data points. Similar to that of the LR model found in RapidMiner, we set out to distinguish between legitimate emails and spam emails. To prepare this model, we first had to import the data with a read.csv function and had to identify our function called "groupProjecttrain" as this was our training set. Then we had our second function with a read.csv called "groupProjectPredict" as this was our prediction set. At this point, we determined the dimension of the files using the dim function and then viewed a summary of the dataset. Furthermore, we installed the correct packages in order to properly model the logistical regression. These libraries included "e1071" and "caret". Then we moved on to setting the seed to "123" and then checked the ratio of the training set. At this point, we were now ready to build our LR model and apply it to the

prediction set. Once we received the results of the LR model and its predictions, we built a confusion matrix in order to determine the accuracy, precision and recall of the dataset. Looking at our attributes, we are starting with Spam(1) and Legit(0). Secondly, looking at the accuracy of the model with a score of 85.8%, the LR model accurately predicts the outcome for a large portion of legit messages. Precision resulted in 94.7% indicating a very high level of confidence in identifying legit messages amongst the predicted spam messages. Recall, also known as sensitivity, is around 78.5%, suggesting that the model captures a large portion of the legit messages. The F1-score, which is a measure of precision and recall, resulted in 85.9%, enforcing the models overall effectiveness. These scores indicate that it is effective in predicting legit messages vs. spam messages.

Model 7: Neural Network (with RapidMiner)



The screenshot shows the 'PerformanceVector (Performance)' window in RapidMiner for the 'ImprovedNeuralNet (Neural Net)' model. The 'Table View' is selected, displaying a confusion matrix. The matrix shows 802 true positives (pred 0, true 0), 70 false positives (pred 0, true 1), 34 false negatives (pred 1, true 0), and 474 true negatives (pred 1, true 1). The overall accuracy is 92.46%, precision is 91.97%, and recall is 95.93%.

	true 0	true 1	class	precision
pred 0	802	70		91.97%
pred 1	34	474		93.31%
class	recall	87.13%		

Figure 11 Details: Confusion Matrix for Neural Network in RapidMiner

Looking at the Neural Net (NN) performance vector, the accuracy result is 92.46%, Precision is 91.97% and recall is 95.93%. In this example 802 (True Positive) samples were predicted to be true and they did show up as spam. 474 examples (True Negative) show that they were predicted as False and did not end up as spam emails.

The above indicates that the Neural Network model has the following values for spam messages:

Accuracy: 92.46%

Recall: 95.93%

Precision: 91.97%

The ratio of correctly predicted observation to the total observation is: 92.46%. So, for spam messages in this model we have a total percentage of 95.93% the correct total of actual positive numbers for the spam messages. And 91.97% for the exactness values of spam messages.

However, as our recall increases, our precision decreases because in addition to increasing the positives, we increase the false positives too. In this NN model we have higher recall which means that an algorithm returns most of the relevant results. So, if we want to decide on which one is better, recall or precision?

Similar to Model 3, the first thing is to remember that it is not possible to amplify both Precision and Recall matrices at the same time, as one comes at the cost of another. For simplicity, there is another metric available called F-score, which is a harmonic mean of precision and recall. For problems when both Precision and Recall are important the equation as below:

$$F1 \text{ Score} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall}) = 2 * (91.97 * 95.93) / (91.97 + 95.93)$$

= 46.95% is the weighted average of precision and recall translating as about 47% takes both false positives and false negatives into account, less than the NB Model 3.

Model 8: Neural Network - R/RStudio

For our neural network model, we used the following packages in R/RStudio to create it - “*dplyr*”, “*nnet*”, and “*NeuralNetTools*”. We performed the same pre-processing step of removing the “*rand_num*” column and setting our random seed to ensure consistent results, as with our previous models created in R. As another reminder, our model was also created with the training portion of our dataset and the model was evaluated with the testing portion of our dataset.

Similar to the decision tree model in R/RStudio, we also created three different iterations to work with based on the “*size*” parameter in our “*nnet*” function. This function is only able to develop neural network models with a single hidden layer, but we were able to specify the amount of nodes the hidden layer uses. In general, we created a model with a higher number of hidden layers and nodes, which can become a more powerful model, but it also imposed the risk of overfitting with our training model. Even though the training model can perform well on its own, it will not be very useful for future predictions or evaluations. We want to create a powerful enough model that will not be prone to overfitting, and will still be useful in future predictions.

With our three models, we ended up creating neural networks with 1 hidden layer containing 1, 4, and 8 hidden nodes, respectively for each of our models. We also specified the “*maxit*” parameter to be 10,000 for each of our models as well so our neural networks can have enough iterations for each model to be optimized as much as possible. A summary of each of our models and their graphs are shown below:

- SpamNN1 ← size = 1, maxit = 10000
- SpamNN2 ← size = 4, maxit = 10000
- SpamNN3 ← size = 8, maxit = 10000

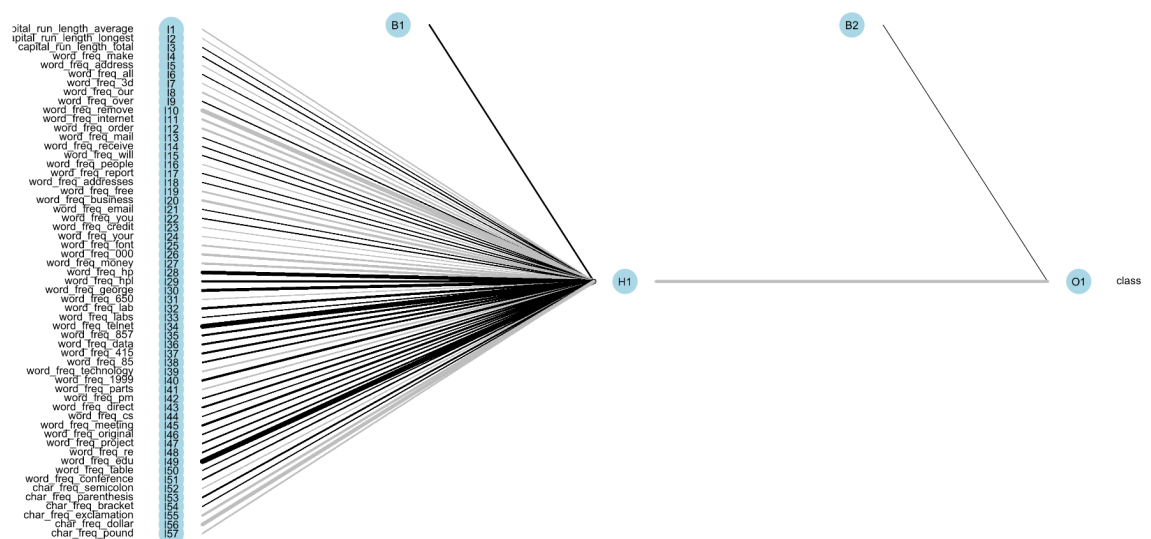


Figure 12 Details: SpamNN1 Neural Network Graph in R/RStudio

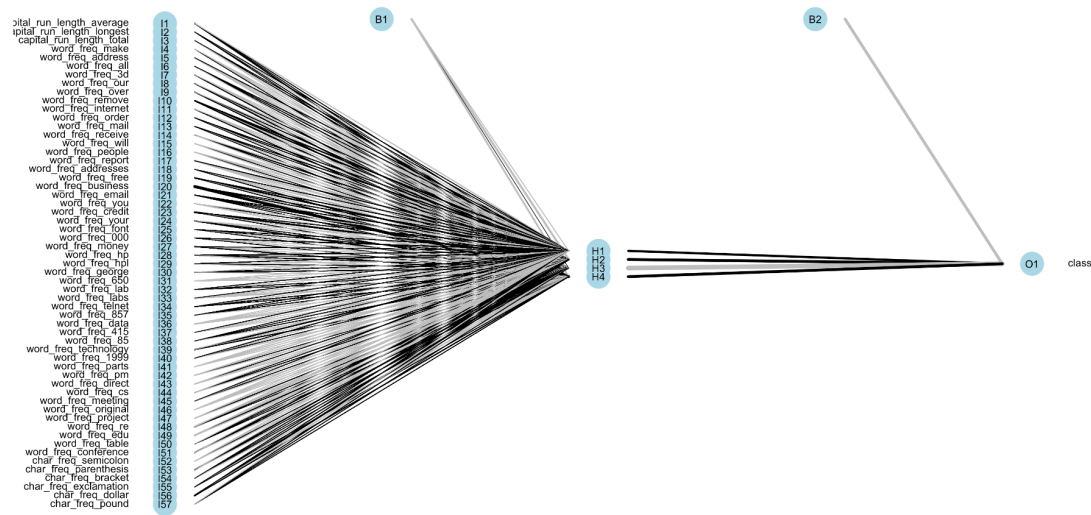


Figure 13 Details: SpamNN2 Neural Network Graph in R/RStudio

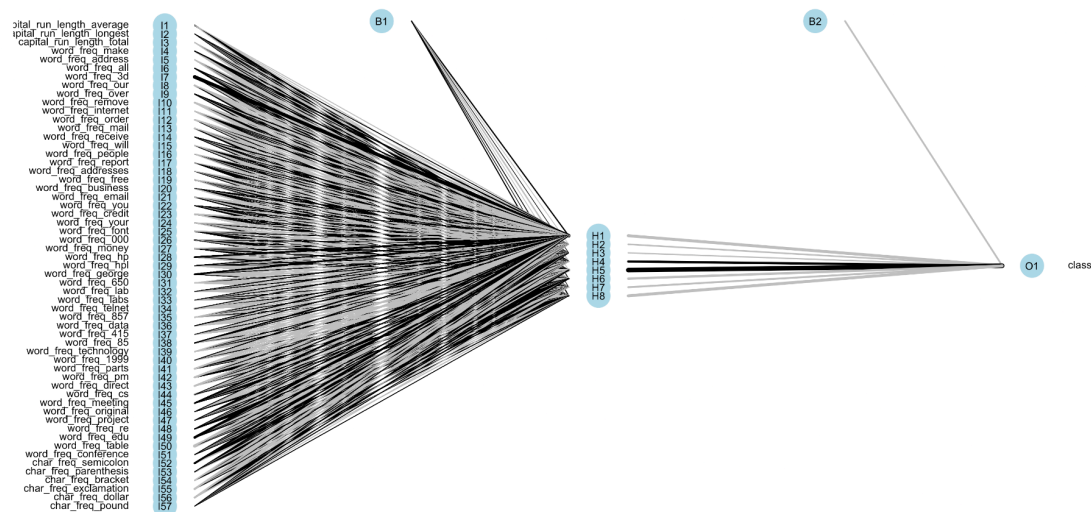


Figure 14 Details: SpamNN3 Neural Network Graph in R/RStudio

With neural network graphs, we can't use them to draw very useful conclusions about how the model operates, since neural networks are seen as a “*black box*” model where most of the model can not be visualized or interpreted very easily because of how complex the model

was. We were able to see how it performed though and it will be covered in the Model Evaluation section.

Model Evaluation

After working with each of our models, we evaluated and compared our models based on their overall accuracy, precision (to see if our predicted spam emails were actually spam), and recall (to see how complete our predictions were). From our perspective, we would rather mislabel a legitimate email as spam over mislabeling a spam email as legitimate, since clicking and interacting with a spam email as if it was a legitimate one could have potentially harmful consequences (especially if it also happened to be a phishing email), such as stolen/leaked data, an invasion of privacy, or in some cases identity theft or financial theft. Also, if a legitimate email is accidentally classified as spam, a user could easily look in their spam inbox to retrieve the email and train the model for future emails.

The summary table below showcases the performance metrics for each of our eight models and how they compare with each other. For models that included multiple variants, such as the decision tree, the variant that had the best performance metrics compared to the others within the model was selected to be in the summary table.

<u>Model Name</u>	<u>Accuracy</u>	<u>Precision</u>	<u>Recall</u>
Model 1: Decision Tree (RM)	86.59%	83.59%	96.89%
Model 2: Decision Tree (R)	80.12%	88.10%	69.30%
Model 3: Naive Bayes (RM)	80.14%	96.68%	96.32%
Model 4: Naive Bayes (R)	69.95%	97.02%	56.74%
Model 5: Logistic Regression (RM)	92.39%	92.55%	88.24%
Model 6: Logistic Regression (R)	87.98%	94.67%	78.50%
Model 7: Neural Network (RM)	92.46%	93.31%	87.13%
Model 8: Neural Network (R)	93.90%	92.94%	91.58%

The highlighted cells in each of the accuracy, precision and recall columns indicate the highest

value found in each of those columns. As we can see from this summary table, we can find a few key takeaways:

- Our Neural Network model in R/RStudio (Model 8) was the model that generated the highest overall accuracy at 93.90%.
- Our Naive Bayes model in R/RStudio (Model 4) was the model that generated the highest spam email precision at 97.02%.
- Our Decision Tree model in RapidMiner (Model 1) was the model that generated the highest spam email recall at 96.89%.

For the most part, when comparing our models against each other, we saw that most of our metrics were very close to each other, scoring anywhere above 80%-90% for every evaluated performance metric. Since our team was originally working with one large dataset from the UC Irvine Machine Learning Repository, we use the holdout method to split our dataset and reserve 30% of the original dataset (approximately 690 records) for testing, predictive analysis, and model evaluation.

However, one of the important takeaways from this entire report is that based on our findings, we were able to determine that our Naive Bayes model in R/RStudio (Model 4) is the best model for our given circumstances of contributing to the development of a spam filter. Even though it had the lowest percentages for overall accuracy and recall by an extremely wide margin (69.95% and 56.74% respectively), we believed that having the highest precision percentage at 97.02% is a more important measure to go off of.

This is because precision is a more important measure when the cost or stakes of creating a false positive is greater than the cost or stakes of creating a false negative. As previously mentioned in this section, if our spam filter sent a false positive (ie. - a spam email to a user's main inbox) and the user clicked on it, the consequences could range from annoying to devastating.

On one end, at its best-case scenario, the user would click on this spam email and train the model to accept more emails of this type as legitimate, giving the user the inconvenience of seeing more spam and junk email to delete or mark as spam again. In more worse scenarios, these spam emails could retrieve information about a particular user and either leak it publicly if it is personal information or sell this information to companies with malicious or immoral intent, such as targeted advertising or phishing. If these spam emails were also a phish that the user fell for since it would appear as a legitimate email, these senders could obtain usernames and passwords, financial information, or other sensitive personal information to use for blackmail, harassment, or identity theft.

On the opposite end of the spectrum, if a legitimate email were to be misplaced, most email providers, such as Google, Yahoo, or ProtonMail, have separate inboxes for spam or junk email. If the user was expecting a legitimate email that the user could not find, oftentimes, they would look in their spam inbox to retrieve the email and mark the sender as legitimate for future emails. As a worst-case scenario for this side, the user could also reach out to the sender through other means, like through phone or a website to inform the sender that their email did not get sent to the recipient so it can be re-sent or troubleshooted. The consequences of the former compared to the latter end are more severe, so this is why precision is a more valued measurement over anything else. As another example use-case, consider a research study where

you, as a scientist, are wanting to create the best model to detect a rare and fatal type of cancer among 100 patients. The consequences of incorrectly identifying a patient as “cancer-free” even though they actually do have cancer would be extremely devastating as opposed to identifying a false positive, since false negative patients would not start the cancer treatment process.

Discussion and Conclusion

After going through the entire process of data selection, data understanding, pre-processing, data mining with several different data mining models, making data predictions, and model evaluation, we now come to point our data mining cycle where we deploy our model and monitor its results to keep building upon the model and refining it as we continue to get more data. Even though most of these detail are still abstract and not planned to a certain degree of detail, we can still visualize and speculate how our findings can help people across the internet stay away from spam emails and phishing attempts.

Our hope is to collaborate with different email providers, whether that be with larger companies like Google, Yahoo, or Microsoft, or smaller and more niche companies, like ProtonMail, to implement our model into their existing spam filters to see how well it performs or how we can continue to maintain or refine the model with the help of these companies. We could also consider working with different scales of businesses, such as larger enterprise organizations with their own domains, or smaller companies that use personal emails for business to assist with the development of custom spam filters for their businesses.

This project could benefit all email users, but it really just depends on who we work with and our target audience, but the flexibility of working with different providers or users gives us several opportunities to work with. The challenge of reducing spam or phishing attempts is ongoing, and the problem can never truly be resolved, but with the power of data mining, spam filters have been greatly enhanced through several years worth of data for training and testing. Anytime a spam email gets filtered out, their chances of it posing as a legitimate email that users could fall for is greatly reduced, and their data and privacy can stay a bit more secure because of that.

A few big limitations we have to realize with our models though are with the lack of data and the age of it. With our dataset, even though we have over 4,500 records to work with, larger email providers often have millions or billions of emails to work with, all with more metadata and information to extrapolate and work with, which makes our dataset pale in comparison to what companies like Google or Microsoft are able to work with. Also, the dataset was fully finalized and uploaded to the UC Irvine Machine Learning Repository in the early 2000's, so a lot of trends have changed, started, or stopped since then in the past 20 years. It's more important for our data to be as new as possible since trends in technology in the present-day increase at an exponential rate (partially because of factors like Moore's Law). This might make our data irrelevant from some perspectives, but for the case of this report, the data that we have is still a great starting point for the development of our data mining models throughout the entire data mining process, and this can be backed up by the fact that this dataset is still cited in several peer-reviewed articles, with several of the most recent articles and reports being published as recent as 2019.

The evolution of email has become exponential in our world today and is vital to communication with each other. However, email being an essential form of communication also

leaves the door open for bad actors to take part in devious attack methods. “Furthermore, the legal remedies that can be taken against spammers are limited: it is not difficult to avoid leaving a trace, and spammers easily operate outside the jurisdiction of those countries with anti-spam legislation” (Carpentir & Hunt, p.). This comes in the form of spam emails, which contain phishing attacks to obtain personal information including, but not limited to; SSN, birthdates, bank accounts, etc. As earlier stated, an average of over three billion spam emails are sent daily, therefore, it is safe to assume most people experience spam emails multiple times daily. This project aimed to use various data mining methods to determine different types of spam emails using specific language within the email message. The different methods used include Decision Trees, Neural Networks, Naive Bayes, and others. We used two different software programs including RapidMiner and RStudio to provide a more in-depth look into the data and to also provide a different perspective for each model.

References

- Awad, W.A., & ELseuofi, S.M. (February 2011). "Machine Learning Methods for Spam EMail Classification." *International Journal of Computer Science & Information Technology*: 3(1). 173-184. 10.5121/ijcsit.2011.3112.
https://www.researchgate.net/publication/50211017_Machine_Learning_Methods_for_Spam_email_Classification.
- Carpinter, J. & Hunt, R. (2006). "Tightening the net: A review of current and next generation spam filtering tools." *Computers & Security*, 25:8, 566-578.
<https://doi.org/10.1016/j.cose.2006.06.001>.
<https://www.sciencedirect.com/science/article/pii/S0167404806000939>.
- Clark, J. & Koprinska, I., & Poon, J. (2003). "A neural network based approach to automated email classification." *IEEE Xplore*, 702-705. 10.1109/WI.2003.1241300.
<https://ieeexplore.ieee.org/document/1241300>.
- Damar, M. (2023). "email Marketing in all its Dimensions: Literature, Server Selection, Auxiliary Tools, Sample Coding and Design." *The Essentials of Today's Marketing-2*. 183-210. İstanbul: Efe Akademi. ORCID: 0000-0002-3985-3073.
<https://avesis.deu.edu.tr/yayin/42339536-e215-489a-8922-7323d1ad546c/email-marketing-in-all-its-dimensions-literature-server-selection-auxiliary-tools-sample-coding-and-design>.
- Griffiths, C. (02 October 2023). "The Latest Phishing Statistics." *AAG IT Services*.
<https://aag-it.com/the-latest-phishing-statistics/>.
- Gomes, L.H., Cazita, C., Almeida, J.M., Almeida, V., & Meira, W. (August, 2007). "Workload models of spam and legitimate emails." *Performance Evaluation*: 64(7-8), 690-714.
<https://doi.org/10.1016/j.peva.2006.11.001>.
<https://www.sciencedirect.com/science/article/pii/S0166531606001167?via%3Dihub>.
- Guzella, T. S. and Caminhas, W. M. (2009). "A review of machine learning approaches to Spam filtering." *Expert Systems with Applications*: 36. 10206–10222.
<https://www.sciencedirect.com/science/article/pii/S095741740900181X>.
- Hopkins, M., Reeber, E., Forman, G., & Suermondt, J. (1999). "Spambase." UCI Machine Learning Repository. <https://doi.org/10.24432/C53G6X>.
- Kindness, J. (26 February 2023). "Social Media vs. Email Marketing." *AgencyAnalytics*.
<https://agencyanalytics.com/blog/social-media-vs-email-marketing>.
- Pérez-Díaz, N., Ruano-Ordás, D., Fdez-Riverola, F., & Méndez, J. R. (2016). "Boosting Accuracy of Classical Machine Learning Antispam Classifiers in Real Scenarios by Applying Rough Set Theory." *Scientific Programming: 2016*. Article ID 5945192.
<https://doi.org/10.1155/2016/5945192>.
- Petrosyan, A. (8 March 2023) "Monthly share of spam in the total email traffic worldwide from January 2014 to December 2022." Statista.
<https://www.statista.com/statistics/420391/spam-email-traffic-share/>.
- Rainie, L., & Fallows, D. (March, 2004). "The can-spam act has not helped most email users so far." *Pew and Internet American Life Project*.
https://www.pewresearch.org/internet/wp-content/uploads/sites/9/media/Files/Reports/2004/PIP_Data_Memo_on_Spam.pdf.pdf.
- Steinbrinck, K. (28 January, 2021). "History of Email." *Email on Acid*.

- <https://www.emailonacid.com/blog/article/email-marketing/history-of-email/>.
- Yu, B., & Ben Xu, Z. (2008). "A comparative study for content-based dynamic spam classification using four machine learning algorithms." *Knowledge-Based Systems*, 21(4), 355–362.
- <https://www.sciencedirect.com/science/article/pii/S0950705108000026?via%3Dihub>.

Appendix**Model 1: Decision Tree (RM)**

1. RM, Decision Tree training dataset top 3 attributes

✓ capital_run_length_average	Real	0	Min 1	Max 667	Average 6.617
✓ capital_run_length_longest	Integer	0	Min 1	Max 1333	Average 51.676
✓ capital_run_length_total	Integer	0	Min 2	Max 3498	Average 272.491

2. RM, Decision Tree testing dataset top 3 attributes

✓ capital_run_length_average	Real	0	Min 1	Max 1102.500	Average 4.940
✓ capital_run_length_longest	Integer	0	Min 1	Max 9989	Average 52.260
✓ capital_run_length_total	Integer	0	Min 1	Max 15841	Average 285.191

3. RM, Decision Tree training dataset, 4th highest average

✓ word_freq_you	Real	0	Min 0	Max 9.090	Average 1.661
-----------------	------	---	----------	--------------	------------------

4. RM, Decision Tree testing dataset, 4th highest average

✓ word_freq_you	Real	0	Min 0	Max 18.750	Average 1.662
-----------------	------	---	----------	---------------	------------------

Tree

```

word_freq_remove > 0.010
| word_freq_george > 0.500: 0 {1=0, 0=6}
| word_freq_george ≤ 0.500
| | word_freq_lab > 0.075: 0 {1=0, 0=3}
| | word_freq_lab ≤ 0.075
| | | word_freq_hp > 3.425: 0 {1=0, 0=2}
| | | word_freq_hp ≤ 3.425
| | | | word_freq_labs > 0.500: 0 {1=0, 0=2}
| | | | word_freq_labs ≤ 0.500
| | | | | word_freq_85 > 0.885: 0 {1=0, 0=2}
| | | | | word_freq_85 ≤ 0.885
| | | | | | word_freq_edu > 0.175
| | | | | | word_freq_mail > 0.650: 1 {1=3, 0=0}
| | | | | | word_freq_mail ≤ 0.650: 0 {1=0, 0=6}
| | | | | | word_freq_edu ≤ 0.175
| | | | | | word_freq_hp > 1.650: 0 {1=1, 0=2}

```



```

| | | | | | | word_freq_hp ≤ 1.650: 1 {1=760, 0=20}
word_freq_remove ≤ 0.010
| char_freq_$ > 0.087
| | word_freq_hp > 0.390: 0 {1=1, 0=37}
| | word_freq_hp ≤ 0.390
| | | word_freq_edu > 0.505: 0 {1=0, 0=12}
| | | word_freq_edu ≤ 0.505
| | | capital_run_length_longest > 4.500
| | | | word_freq_meeting > 0.450: 0 {1=0, 0=3}
| | | | word_freq_meeting ≤ 0.450: 1 {1=477, 0=25}
| | | capital_run_length_longest ≤ 4.500: 0 {1=0, 0=9}
| char_freq_$ ≤ 0.087
| | word_freq_credit > 2.910: 1 {1=10, 0=0}
| | word_freq_credit ≤ 2.910
| | | char_freq_! > 0.775
| | | | word_freq_edu > 0.140: 0 {1=0, 0=11}
| | | | word_freq_edu ≤ 0.140
| | | | word_freq_george > 0.500: 0 {1=0, 0=7}
| | | | word_freq_george ≤ 0.500
| | | | word_freq_meeting > 0.875: 0 {1=0, 0=4}
| | | | word_freq_meeting ≤ 0.875
| | | | | char_freq_! > 8.088: 0 {1=0, 0=4}
| | | | | char_freq_! ≤ 8.088
| | | | | char_freq_( > 0.754: 0 {1=0, 0=3}
| | | | | char_freq_( ≤ 0.754: 1 {1=149, 0=21}
| | | char_freq_! ≤ 0.775
| | | | word_freq_3d > 3: 1 {1=4, 0=0}
| | | | word_freq_3d ≤ 3
| | | | capital_run_length_total > 7495: 1 {1=3, 0=0}
| | | | capital_run_length_total ≤ 7495
| | | | word_freq_font > 12.010: 1 {1=3, 0=0}
| | | | word_freq_font ≤ 12.010
| | | | word_freq_650 > 7.105: 1 {1=2, 0=0}
| | | | word_freq_650 ≤ 7.105
| | | | capital_run_length_longest > 353: 1 {1=9, 0=1}
| | | | capital_run_length_longest ≤ 353: 0 {1=391, 0=2608}

```

PerformanceVector

PerformanceVector:

accuracy: 86.59%

ConfusionMatrix:

```

True:  1      0
1:    385    26
0:    159   810

```

precision: 83.59% (positive class: 0)

ConfusionMatrix:

```
True:  1      0
1:    385    26
0:    159    810
```

recall: 96.89% (positive class: 0)

ConfusionMatrix:

```
True:  1      0
1:    385    26
0:    159    810
```

AUC (optimistic): 0.982 (positive class: 0)

AUC: 0.840 (positive class: 0)

AUC (pessimistic): 0.698 (positive class: 0)

Model 2: Decision Tree (R)

1. Full R Code used:

```
# Team C: DMM Project (Code by Rex A. Herndon)

#
=====
====
# ===== Decision Tree
=====
#
=====
=====

# defining and choosing our training/testing datasets
SpamTrain <- read.csv(file.choose(), header=T, sep = ",")
SpamTest <- read.csv(file.choose(), header=T, sep = ",")

# doing initial checks to verify if data imported correctly
names(SpamTrain)
dim(SpamTrain)
summary(SpamTrain)
head(SpamTrain)

names(SpamTest)
dim(SpamTest)
summary(SpamTest)
head(SpamTest)

# removing rand_num column since it is not needed for analysis
SpamTrain[59] <- NULL
SpamTest[59] <- NULL

# checking to verify if col was successfully removed
names(SpamTrain)
```

```

names(SpamTest)

# attaching training dataset for ease of access
# no longer needed on my machine, but leaving in case of future use
# attach(SpamTrain)

# installing/invoking "party" library for DT modeling
# commenting out "install.packages("party")" since i already have this
# installed on my machine, but could be used in the future if using package
# for the first time
# install.packages("party")
library("party")

# building DT model using ctree function using "class" as the target
# attribute and everything else as the predictor attributes
# creating three different models based on minsplitted to see which one
# performs the best

SpamTree1 <- ctree(class ~., data=SpamTrain,
                   controls = ctree_control(
                     mincriterion = 0.99, minsplitted = 1000
                   ))
SpamTree2 <- ctree(class ~., data=SpamTrain,
                   controls = ctree_control(
                     mincriterion = 0.99, minsplitted = 1500
                   ))
SpamTree3 <- ctree(class ~., data=SpamTrain,
                   controls = ctree_control(
                     mincriterion = 0.99, minsplitted = 2000
                   ))

# check output
SpamTree1
SpamTree2
SpamTree3

# generate a decision tree graph
plot(SpamTree1)
plot(SpamTree2)
plot(SpamTree3)

# ===== DT Model Predictions and Evaluation
=====

# ----- Creating Prediction Dataset
-----

# apply DT model to our test dataset to make predictions
SpamPredictions1 <- predict(SpamTree1, SpamTest)
SpamPredictions2 <- predict(SpamTree2, SpamTest)
SpamPredictions3 <- predict(SpamTree3, SpamTest)

# round predicted probabilities to the 3rd decimal place

```

```

SpamPredictions1 <- round(SpamPredictions1, 3)
SpamPredictions2 <- round(SpamPredictions2, 3)
SpamPredictions3 <- round(SpamPredictions3, 3)

# round them to 0 and 1 values for easier predictions
DT_SpamPredictions1 <- ifelse(SpamPredictions1 >= 0.5 & SpamPredictions1
<= 1, 1, 0)
DT_SpamPredictions2 <- ifelse(SpamPredictions2 >= 0.5 & SpamPredictions2
<= 1, 1, 0)
DT_SpamPredictions3 <- ifelse(SpamPredictions3 >= 0.5 & SpamPredictions3
<= 1, 1, 0)

# view cleaned/processed prediction values
DT_SpamPredictions1
DT_SpamPredictions2
DT_SpamPredictions3

# ----- Confusion Matrix / Model Evaluation
-----

# generating confusion matrix from processed predictions
DT1_conf_matrix <- table(DT_SpamPredictions1, SpamTest$class)
DT2_conf_matrix <- table(DT_SpamPredictions2, SpamTest$class)
DT3_conf_matrix <- table(DT_SpamPredictions3, SpamTest$class)

# viewing confusion matrix
DT1_conf_matrix
DT2_conf_matrix
DT3_conf_matrix

# calculating accuracy metrics
DT1_accuracy <- sum(diag(DT1_conf_matrix)) / sum(DT1_conf_matrix)
DT2_accuracy <- sum(diag(DT2_conf_matrix)) / sum(DT2_conf_matrix)
DT3_accuracy <- sum(diag(DT3_conf_matrix)) / sum(DT3_conf_matrix)
DT1_accuracy <- round(DT1_accuracy * 100, 3)
DT2_accuracy <- round(DT2_accuracy * 100, 3)
DT3_accuracy <- round(DT3_accuracy * 100, 3)

# calculating precision metrics
DT1_precision <- DT1_conf_matrix[2, 2] / sum(DT1_conf_matrix[, 2])
DT2_precision <- DT2_conf_matrix[2, 2] / sum(DT2_conf_matrix[, 2])
DT3_precision <- DT3_conf_matrix[2, 2] / sum(DT3_conf_matrix[, 2])
DT1_precision <- round(DT1_precision * 100, 3)
DT2_precision <- round(DT2_precision * 100, 3)
DT3_precision <- round(DT3_precision * 100, 3)

# calculating recall metrics
DT1_recall <- DT1_conf_matrix[2, 2] / sum(DT1_conf_matrix[2, ])
DT2_recall <- DT2_conf_matrix[2, 2] / sum(DT2_conf_matrix[2, ])
DT3_recall <- DT3_conf_matrix[2, 2] / sum(DT3_conf_matrix[2, ])
DT1_recall <- round(DT1_recall * 100, 3)
DT2_recall <- round(DT2_recall * 100, 3)
DT3_recall <- round(DT3_recall * 100, 3)

```

```
# final results

cat("DT Model 1
    \nAccuracy: ", DT1_accuracy,
    "\nPrecision: ", DT1_precision,
    "\nRecall: ", DT1_recall)
cat("=====")
cat("DT Model 2
    \nAccuracy: ", DT2_accuracy,
    "\nPrecision: ", DT2_precision,
    "\nRecall: ", DT2_recall)
cat("=====")
cat("DT Model 3
    \nAccuracy: ", DT3_accuracy,
    "\nPrecision: ", DT3_precision,
    "\nRecall: ", DT3_recall)

# Model 1 appears to generate the highest recall and accuracy, but model
# 3 generates the highest precision.
# it's more important to have a higher precision, since the the
# consequence of a false positive could lead to unwanted consequences, like
# data loss or identity theft
# with lower recall, having false negatives is more acceptable because
# users can retrieve legitimate emails from their spam folder if needed
# for our circumstances, model 3 gives us the best result
```

2. Text Outputs of Decision Tree Models:

```
> SpamTree1
```

```
Conditional inference tree with 19 terminal nodes
```

```
Response: class
Inputs: capital_run_length_average, capital_run_length_longest,
capital_run_length_total, word_freq_make, word_freq_address,
word_freq_all, word_freq_3d, word_freq_our, word_freq_over,
word_freq_remove, word_freq_internet, word_freq_order, word_freq_mail,
word_freq_receive, word_freq_will, word_freq_people, word_freq_report,
word_freq_addresses, word_freq_free, word_freq_business, word_freq_email,
word_freq_you, word_freq_credit, word_freq_your, word_freq_font,
word_freq_000, word_freq_money, word_freq_hp, word_freq_hpl,
word_freq_george, word_freq_650, word_freq_lab, word_freq_labs,
word_freq_telnets, word_freq_857, word_freq_data, word_freq_415,
word_freq_85, word_freq_technology, word_freq_1999, word_freq_parts,
word_freq_pm, word_freq_direct, word_freq_cs, word_freq_meeting,
word_freq_original, word_freq_project, word_freq_re, word_freq_edu,
word_freq_table, word_freq_conference, char_freq_semicolon,
char_freq_parenthesis, char_freq_bracket, char_freq_exclamation,
char_freq_dollar, char_freq_pound
Number of observations: 3912
```

- 1) word_freq_your <= 0.6; criterion = 1, statistic = 617.847
- 2) word_freq_000 <= 0.25; criterion = 1, statistic = 194.937

```

3) word_freq_remove <= 0.04; criterion = 1, statistic = 173.384
4) char_freq_dollar <= 0.18; criterion = 1, statistic = 98.149
5) word_freq_free <= 0.11; criterion = 1, statistic = 67.048
6) capital_run_length_longest <= 108; criterion = 1, statistic
= 81.956
7) word_freq_money <= 0.46; criterion = 1, statistic = 73.658
8) capital_run_length_average <= 4.608; criterion = 1,
statistic = 44.75
9) char_freq_exclamation <= 0.769; criterion = 1,
statistic = 37.301
10) word_freq_credit <= 0; criterion = 1, statistic =
21.552
11) capital_run_length_average <= 2.362; criterion =
0.999, statistic = 19.69
12) word_freq_mail <= 0.33; criterion = 0.999,
statistic = 17.799
13)* weights = 1109
12) word_freq_mail > 0.33
14)* weights = 83
11) capital_run_length_average > 2.362
15)* weights = 443
10) word_freq_credit > 0
16)* weights = 18
9) char_freq_exclamation > 0.769
17)* weights = 61
8) capital_run_length_average > 4.608
18)* weights = 83
7) word_freq_money > 0.46
19)* weights = 21
6) capital_run_length_longest > 108
20)* weights = 29
5) word_freq_free > 0.11
21)* weights = 154
4) char_freq_dollar > 0.18
22)* weights = 66
3) word_freq_remove > 0.04
23)* weights = 117
2) word_freq_000 > 0.25
24)* weights = 92
1) word_freq_your > 0.6
25) word_freq_hp <= 0.11; criterion = 1, statistic = 245.746
26) word_freq_edu <= 0.16; criterion = 1, statistic = 167.136
27) word_freq_george <= 0.2; criterion = 1, statistic = 205.763
28) word_freq_re <= 1.44; criterion = 1, statistic = 67.414
29) word_freq_hpl <= 0.1; criterion = 1, statistic = 74.179
30) capital_run_length_total <= 67; criterion = 1, statistic
= 51.458
31)* weights = 240
30) capital_run_length_total > 67
32)* weights = 1004
29) word_freq_hpl > 0.1
33)* weights = 14
28) word_freq_re > 1.44
34)* weights = 18

```

```

    27) word_freq_george > 0.2
    35)* weights = 70
    26) word_freq_edu > 0.16
    36)* weights = 83
    25) word_freq_hp > 0.11
    37)* weights = 207

> SpamTree2

Conditional inference tree with 13 terminal nodes

Response: class
Inputs: capital_run_length_average, capital_run_length_longest,
capital_run_length_total, word_freq_make, word_freq_address,
word_freq_all, word_freq_3d, word_freq_our, word_freq_over,
word_freq_remove, word_freq_internet, word_freq_order, word_freq_mail,
word_freq_receive, word_freq_will, word_freq_people, word_freq_report,
word_freq_addresses, word_freq_free, word_freq_business, word_freq_email,
word_freq_you, word_freq_credit, word_freq_your, word_freq_font,
word_freq_000, word_freq_money, word_freq_hp, word_freq_hpl,
word_freq_george, word_freq_650, word_freq_lab, word_freq_labs,
word_freq_telnet, word_freq_857, word_freq_data, word_freq_415,
word_freq_85, word_freq_technology, word_freq_1999, word_freq_parts,
word_freq_pm, word_freq_direct, word_freq_cs, word_freq_meeting,
word_freq_original, word_freq_project, word_freq_re, word_freq_edu,
word_freq_table, word_freq_conference, char_freq_semicolon,
char_freq_parenthesis, char_freq_bracket, char_freq_exclamation,
char_freq_dollar, char_freq_pound
Number of observations: 3912

1) word_freq_your <= 0.6; criterion = 1, statistic = 617.847
  2) word_freq_000 <= 0.25; criterion = 1, statistic = 194.937
    3) word_freq_remove <= 0.04; criterion = 1, statistic = 173.384
      4) char_freq_dollar <= 0.18; criterion = 1, statistic = 98.149
        5) word_freq_free <= 0.11; criterion = 1, statistic = 67.048
          6) capital_run_length_longest <= 108; criterion = 1, statistic
= 81.956
            7) word_freq_money <= 0.46; criterion = 1, statistic = 73.658
              8) capital_run_length_average <= 4.608; criterion = 1,
statistic = 44.75
                9) char_freq_exclamation <= 0.769; criterion = 1,
statistic = 37.301
                  10) word_freq_credit <= 0; criterion = 1, statistic =
21.552
                    11) capital_run_length_average <= 2.362; criterion =
0.999, statistic = 19.69
                      12)* weights = 1192
                        11) capital_run_length_average > 2.362
                          13)* weights = 443
                            10) word_freq_credit > 0
                              14)* weights = 18
                                9) char_freq_exclamation > 0.769
                                  15)* weights = 61

```

```

      8) capital_run_length_average > 4.608
        16)* weights = 83
      7) word_freq_money > 0.46
        17)* weights = 21
      6) capital_run_length_longest > 108
        18)* weights = 29
      5) word_freq_free > 0.11
        19)* weights = 154
      4) char_freq_dollar > 0.18
        20)* weights = 66
      3) word_freq_remove > 0.04
        21)* weights = 117
      2) word_freq_000 > 0.25
        22)* weights = 92
      1) word_freq_your > 0.6
        23) word_freq_hp <= 0.11; criterion = 1, statistic = 245.746
          24)* weights = 1429
        23) word_freq_hp > 0.11
          25)* weights = 207

```

> SpamTree3

Conditional inference tree with 6 terminal nodes

Response: class

Inputs: capital_run_length_average, capital_run_length_longest, capital_run_length_total, word_freq_make, word_freq_address, word_freq_all, word_freq_3d, word_freq_our, word_freq_over, word_freq_remove, word_freq_internet, word_freq_order, word_freq_mail, word_freq_receive, word_freq_will, word_freq_people, word_freq_report, word_freq_addresses, word_freq_free, word_freq_business, word_freq_email, word_freq_you, word_freq_credit, word_freq_your, word_freq_font, word_freq_000, word_freq_money, word_freq_hp, word_freq_hpl, word_freq_george, word_freq_650, word_freq_lab, word_freq_labs, word_freq_telnet, word_freq_857, word_freq_data, word_freq_415, word_freq_85, word_freq_technology, word_freq_1999, word_freq_parts, word_freq_pm, word_freq_direct, word_freq_cs, word_freq_meeting, word_freq_original, word_freq_project, word_freq_re, word_freq_edu, word_freq_table, word_freq_conference, char_freq_semicolon, char_freq_parenthesis, char_freq_bracket, char_freq_exclamation, char_freq_dollar, char_freq_pound

Number of observations: 3912

```

1) word_freq_your <= 0.6; criterion = 1, statistic = 617.847
  2) word_freq_000 <= 0.25; criterion = 1, statistic = 194.937
    3) word_freq_remove <= 0.04; criterion = 1, statistic = 173.384
      4) char_freq_dollar <= 0.18; criterion = 1, statistic = 98.149
        5) word_freq_free <= 0.11; criterion = 1, statistic = 67.048
          6)* weights = 1847
        5) word_freq_free > 0.11
          7)* weights = 154
      4) char_freq_dollar > 0.18
        8)* weights = 66

```



```

3) word_freq_remove > 0.04
9)* weights = 117
2) word_freq_000 > 0.25
10)* weights = 92
1) word_freq_your > 0.6
11)* weights = 1636

```

Model 4: Naive Bayes (R)

Full R Code used:

```

(coded by Brandon Koch)
#Define and choose training and prediction datasets
Spamtrain <-read.csv(file.choose(),header=T,stringsAsFactors = T)
SpamPredict<-read.csv(file.choose(), header = T, stringsAsFactors = T)
#View column names and dimensions of the training dataset
names(Spamtrain)
dim(Spamtrain)
#view a summary statistics of the training set
summary(Spamtrain)
#install package e1071
install.packages("e1071")
#Invoke the library e1071
library(e1071)
#build a NB model using NB func
SpamtrainNB <-naiveBayes(class ~.,data = Spamtrain)
#View the NB model to the predition dataset
SpamtrainNB
#apply the NB model to the prediction data
Spam5score<-predict(SpamtrainNB, SpamPredict)
#show a summary of the predition model
summary(Spam5score)
#build a confusion matrix.
confu_matrix<-table(Spam5score, SpamPredict$class)
confu_matrix
#Calculate Accuracy
accuracy<-sum(diag(confu_matrix))/sum(confu_matrix)
accuracy
#calculate precision
precision<-confu_matrix[2,2]/sum(confu_matrix[,2])
precision
#calculate recall
recall<-confu_matrix[2,2]/sum(confu_matrix[2,])
recall

```

Model 6: Logistic Regression R

Full R Code used:

```

(coded by Brandon Koch)
#Import data and name functions
groupProjecttrain<-read.csv(file.choose(), header = T, stringsAsFactors = TRUE)
groupProjectPredict<-read.csv(file.choose(), header = T, stringsAsFactors = TRUE)

```

```

#View column names and dimensions of the dataset
names(groupProjecttrain)
dim(groupProjecttrain)
#view a summary of the data set
summary(groupProjecttrain)
#load the required libraries
install.packages("e1071")
install.packages("caret")
#call upon the correct libraries
library("e1071")
library("caret")
#set random seed to make the sampling reproducible
set.seed(123)
smp_size<-floor(.7*nrow(groupProjecttrain))
train_ind<-sample(seq_len(nrow(groupProjecttrain)),size = smp_size)
train<-groupProjecttrain[train_ind,]
test <- groupProjecttrain[-train_ind,]
#check the ratio of the train set
nrow(train)/nrow(groupProjecttrain)
#build the logistical regression model
LRmodel<-glm(class~., family = "binomial", train)
#apply the model to the test set
LRp<-predict(LRmodel, test, type = "response")
#show a summary of the probabilities
summary(LRp)
#Adjust the probabilites
LRpredict<-ifelse(LRp>.2,1,0)
#generate a confusion matrix
conf_matrix<-table(LRpredict,test[["class"]])
conf_matrix
#Calculate accuracy
accuracy<-sum(diag(conf_matrix))/sum(conf_matrix)
accuracy
#calculate precision
precision<-conf_matrix[2,2]/sum(conf_matrix[,2])
precision
#calculate recall
recall<-conf_matrix[2,2]/sum(conf_matrix[2,])
recall

```

Model 8: Neural Network (R)

1. Full R Code

```

# Team C: DMM Project (Code by Rex A. Herndon)

#
=====
# ===== Neural Network
=====
#
=====

```

```

=====

# defining and choosing our training/testing datasets
SpamTrain <- read.csv(file.choose(), header=T, sep = ",")
SpamTest <- read.csv(file.choose(), header=T, sep = ",")

# checking if dataset imported correctly
dim(SpamTrain)
head(SpamTrain)
summary(SpamTrain)

dim(SpamTest)
head(SpamTest)
summary(SpamTest)

# removing rand_num column since it is not needed for analysis
SpamTrain[59] <- NULL
SpamTest[59] <- NULL
names(SpamTrain)
names(SpamTest)

# installing/invoking dplyr, reshape2, NeuralNetTools, and nnet
  libraries
install.packages('dplyr')
install.packages('nnet')
install.packages("NeuralNetTools")
library(dplyr)
library(reshape2)
library(NeuralNetTools)
library(nnet)

# setting seed to 2001 for consistency
set.seed(2001)

# creating neural network model with "class" as the target attribute and
  every other variable as the predictor attributes
# creating three different models specifying different hidden layer
  sizes and maxit operators as well

SpamNN1 <- nnet(class ~., data=SpamTrain, size = 1, maxit=10000)
SpamNN2 <- nnet(class ~., data=SpamTrain, size = 4, maxit=10000)
SpamNN3 <- nnet(class ~., data=SpamTrain, size = 8, maxit=10000)

# plotting resulting NN
plotnet(SpamNN1)
plotnet(SpamNN2)
plotnet(SpamNN3)

# ===== NN Model Predictions and Evaluation
=====

# ----- Creating Prediction Dataset
-----

```

```

# apply NN model to our test dataset to make predictions
SpamNNPredictions1 <- predict(SpamNN1, SpamTest)
SpamNNPredictions2 <- predict(SpamNN2, SpamTest)
SpamNNPredictions3 <- predict(SpamNN3, SpamTest)

# round predicted probabilities to the 3rd decimal place
SpamNNPredictions1 <- round(SpamNNPredictions1, 3)
SpamNNPredictions2 <- round(SpamNNPredictions2, 3)
SpamNNPredictions3 <- round(SpamNNPredictions3, 3)

# view predictions
SpamNNPredictions1
SpamNNPredictions2
SpamNNPredictions3

# round them to 0 and 1 values for easier predictions
NN_SpamPredictions1 <- ifelse(SpamNNPredictions1 >= 0.5 &
  SpamNNPredictions1 <= 1, 1, 0)
NN_SpamPredictions2 <- ifelse(SpamNNPredictions2 >= 0.5 &
  SpamNNPredictions2 <= 1, 1, 0)
NN_SpamPredictions3 <- ifelse(SpamNNPredictions3 >= 0.5 &
  SpamNNPredictions3 <= 1, 1, 0)

# view cleaned/processed prediction values
NN_SpamPredictions1
NN_SpamPredictions2
NN_SpamPredictions3

# ----- Confusion Matrix / Model Evaluation
# -----

# generating confusion matrix from processed predictions
NN1_conf_matrix <- table(NN_SpamPredictions1, SpamTest$class)
NN2_conf_matrix <- table(NN_SpamPredictions2, SpamTest$class)
NN3_conf_matrix <- table(NN_SpamPredictions3, SpamTest$class)

# viewing confusion matrix
NN1_conf_matrix
NN2_conf_matrix
NN3_conf_matrix

# calculating accuracy metrics
NN1_accuracy <- sum(diag(NN1_conf_matrix)) / sum(NN1_conf_matrix)
NN2_accuracy <- sum(diag(NN2_conf_matrix)) / sum(NN2_conf_matrix)
NN3_accuracy <- sum(diag(NN3_conf_matrix)) / sum(NN3_conf_matrix)
NN1_accuracy <- round(NN1_accuracy * 100, 3)
NN2_accuracy <- round(NN2_accuracy * 100, 3)
NN3_accuracy <- round(NN3_accuracy * 100, 3)

# calculating precision metrics
NN1_precision <- NN1_conf_matrix[2, 2] / sum(NN1_conf_matrix[, 2])
NN2_precision <- NN2_conf_matrix[2, 2] / sum(NN2_conf_matrix[, 2])
NN3_precision <- NN3_conf_matrix[2, 2] / sum(NN3_conf_matrix[, 2])
NN1_precision <- round(NN1_precision * 100, 3)

```

```

NN2_precision <- round(NN2_precision * 100, 3)
NN3_precision <- round(NN3_precision * 100, 3)

# calculating recall metrics
NN1_recall <- NN1_conf_matrix[2, 2] / sum(NN1_conf_matrix[2, ])
NN2_recall <- NN2_conf_matrix[2, 2] / sum(NN2_conf_matrix[2, ])
NN3_recall <- NN3_conf_matrix[2, 2] / sum(NN3_conf_matrix[2, ])
NN1_recall <- round(NN1_recall * 100, 3)
NN2_recall <- round(NN2_recall * 100, 3)
NN3_recall <- round(NN3_recall * 100, 3)

# final results

cat("NN Model 1
    \nAccuracy: ", NN1_accuracy,
    "\nPrecision: ", NN1_precision,
    "\nRecall: ", NN1_recall)
cat("=====")
cat("NN Model 2
    \nAccuracy: ", NN2_accuracy,
    "\nPrecision: ", NN2_precision,
    "\nRecall: ", NN2_recall)
cat("=====")
cat("NN Model 3
    \nAccuracy: ", NN3_accuracy,
    "\nPrecision: ", NN3_precision,
    "\nRecall: ", NN3_recall)

# here, model 3 has the highest performance out of all three attributes!

```

Confusion matrix

In binary classification, there are two possible target classes, which are typically labeled as "positive" and "negative" or "1" and "0". In our spam example, the target (positive class) is "spam," and the negative class is "not spam."

When evaluating the accuracy, we looked at correct and wrong predictions disregarding the class label. However, in binary classification, we can be "correct" and "wrong" in two different ways.

Correct predictions include so-called true positives and true negatives. This is how it unpacks for our spam use case example:

- **True positive (TP):** An email that is actually spam and is correctly classified by the model as spam.
- **True negative (TN):** An email that is actually not spam and is correctly classified by the model as not spam.

Model errors include so-called false positives and false negatives. In our example:

- **False Positive (FP):** An email that is actually not spam but is incorrectly classified by the model as spam (a "false alarm").
- **False Negative (FN):** An email that is actually spam but is incorrectly classified by the model as not spam (a "missed spam").

Presentation Link

To supplement this report, an additional presentation was recorded by all project team members, and can be accessed with the following link: <https://youtu.be/29QTNO8LbOE>