# Introduction

As electronic mail (email) became a more commonly used form of communication, advertisers and companies took advantage of the essentially free method of communication and created emails the recipients did not want or care for (spam). While email users used to have to manually delete spam emails that made it to the inbox, users tagging or labeling emails as spam can be used to create a training set for a machine learning algorithm to classify incoming emails as either “spam” or “ham” before the message reaches the recipient.

Using a dataset of emails from the Enron public corpus, I will attempt to train a classifier to predict whether an email is “spam” or “ham”. Multiple experiments will be conducted to see which features are more valuable to the classifier; additionally, two classifiers will be trained and tested using a term frequency-inverse document frequency matrix, one manually created using the functions of the NLTK package, and another created with the Sci-Kit Learn vectorizer and an SVM classifier.

# Data

The data provided consisted of 3,672 “ham” emails, and 1,500 “spam” emails. To balance the classifier, I only used 1,500 if the real emails, and all of the spam emails. A glimpse of the first couple of emails shows that emails generally start with a subject line with the first word being “Subject”. After the first word of the email, all other words appear to be lowercased.

# Pre-Processing

To simplify the task of processing the data into a manner readable by the classifier, the program read the first 1,500 spam emails and placed them in an empty list, and read the first 1,500 “ham emails and placed them in and empty list. After all of the emails were read, each email was tokenized, and the tokens were placed in a tuple with the category of the email, either “spam” or “ham”. The tuples were then placed in another empty list, and shuffled so the training of the classifier would not be biased.

# Experiments

To find out which features work the best when classifying the emails, four different classifiers were trained and evaluated for their average accuracy, recall, precision, and F1 Score on a 10-fold cross validation with both “spam” as the positive class and “ham” as the positive class. The first three models were trained with various features extracted from the text, such as presence of top words and parts of speech tags. These classifiers were trained using the Naïve Bayes classifier in the NLTK package.

The fourth classifier trained was created using the Sci-Kit Learn (sklearn) package from a term frequency-inverse term frequency (TF-IDF) matrix. The TF-IDF matrix finds the term frequency of a given term within a document, and multiplies if by the log of the inverse term appearance throughout all the documents. An SVM classifier from the sklearn package will then use the TF-IDF as the features for the emails to predict whether an email is spam or ham.

## Baseline Model

The first classifier created was considered the baseline classifier. The features used to classify the emails were taken from the most common 1,000 tokens in all the emails. A feature was either true of false for the email based on whether a token from an email was one of the most common 1,000 tokens in all emails. For example, the most frequently occurring token in all emails was the symbol “-“; if the symbol “-“ was in the given document, the feature was give the value of true, if the symbol was not present in the email, then the feature was given the value of false.

After the features were created, ten-fold cross validation was used to retrieve the average accuracy and the precision, recall and F1 score for each class. This baseline model achieved an average accuracy 0.950 on the ten-fold cross validation. The precision, recall and F1 score for each of the classes is provided in Table 1.

Table Metrics for Baseline Classifier

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1 Score** |
| Ham | 0.902 | 0.999 | 0.948 |
| Spam | 0.999 | 0.911 | 0.952 |

The classifier is 95% accurate, which is a pretty high accuracy for a baseline model. One item to note from the performance metrics is the high recall in the ham class. This means that the classifier catches 99.9% of all actual emails and labels them as ham. This is a desirable feature in the for the email classifier as a user would likely rather have to delete a couple of extra spam emails from the inbox, than totally miss some actual emails because they never made it to the inbox for the recipient to read.

## Stopwords and Punctuation Removal

The next experiment conducted was to train a classifier using tokens that had stopwords, and non-alphabetical tokens removed. The stopwords were attained from the NLTK stopword corpus. A list was created that went through each token in every email, and if the token not alphabetical or if the token was a stopword, the token was not placed in the list. With the list of all non-stopword and only alphabetical tokens, a word frequency dictionary was created, and the top 1,000 words were chosen as the word features.

The same function that created the features for the baseline experiment was used to create the features for the new experiment without stopwords or punctuation by passing each email through the function, along with the newly created token list. Once again a dictionary was returned with a key for each of the most common 1,000 alphabetical, non-stopwords and whether it was in the email or not.

Once again, a Naïve Bayes classifier was trained with the new feature sets and the average accuracy and average precision, recall, and f1 score was calculated for each class on a 10-fold cross validation. The average 10-fold accuracy of the model was 0.957, which only increased the accuracy by 0.3%. The average precision, recall, and F1 score for both ham and spam are provided in Table 2.

Table Metrics for Classifier without Stopwords or Punctuation

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1 Score** |
| Ham | 0.927 | 0.986 | 0.956 |
| Spam | 0.988 | 0.932 | 0.959 |

One of the main differences between the baseline classifier and the classifier without stopwords or punctuation is the increase in ham precision at the cost of ham recall. While the accuracy of the classifier did increase slightly, the fairly significant decrease in ham recall would not be worth the 0.3% increase in the accuracy of the classifier.

## Part of Speech Tagging

The next classifier trained using the NLTK package was a classifier that had the stopwords and punctuation removed, but also provided a count of words that were tagged as a noun, verb, adjective, or adverb. To add the part of speech counts to the email features, a new function was created. Most of the old function was preserved, but there was an additional step in which the document had each of its words tagged (even repeated words) and each of those tags were passed through. Counts for each part of speech were initialized to be equal to zero, and every time a part of speech were encountered on the for loop, the given part of speech’s count increased by one.

Once again, a 10-fold cross validation was conducted and the average accuracy, and the average precision, recall, and F1 score were calculated for each class. The average accuracy for the new classifier with part of speech counts was 0.949. The average precision, recall, and F1 score for both ham and spam classes are provided in Table 3.

Table Metrics for Classifier with Stopwords and Punctuation and POS Counts

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1 Score** |
| Ham | 0.925 | 0.971 | 0.947 |
| Spam | 0.973 | 0.929 | 0.950 |

The addition of the part of speech tags to the classifier actually made the classifier less accurate than the baseline model and had a worse recall for the ham class than the baseline model. Adding the parts of speech counts to the feature set was not helpful in classifying the emails.

## TF-IDF Matrix and Naïve Bayes Classifier

The last classifier trained with the NLTK package was a TF-IDF feature set with a Naïve Bayes Classifier. To make the TF-IDF feature set, I created a frequency distribution of tokens, not including stopwords or punctuation, and then found the most common 1,000 terms. After the dictionary was created, I took all the terms found to be used most often, and placed them in a list. Now that a list of words of interest was created, I created an empty dictionary, and looped through each of the emails. In each email, I created a list of unique tokens in the email and looped through the words of interest. I then looked to see if that word of interest was in the email; if it was, the word in the document counting dictionary increased by one, and if it wasn’t, the email count remained the same. After all the emails were looped through, each item in the document counter, and the document count value was divided by 3,000 (the number of emails inspected).

Now that the document frequency dictionary was created, I created a function that returns a TF-IDF observation for a given email in the email document collection. The function takes two parameters, the tokenized document and a document frequency dictionary. First, a word count dictionary was created with the tokenized email, and then the counts were divided by the number of tokens in the email. Then, an empty feature dictionary was created, and each word and document frequency value in the document frequency dictionary was looped through. For each word in the document frequency dictionary, a key was created in the dictionary that combined “Tfidf\_” + the word. The word was then used to find the term frequency value in the term frequency dictionary, and if it was not present, a zero was returned, and this value was multiplied by the log of the inverse of the document frequency. After all words of interest were looped through, the tfidf dictionary was returned.

After the dictionary was created, I created a list that contained a tuple of a tfidf dictionary and the ham or spam classification for each of the emails. After the list was created, a Naïve Bayes classifier was created and trained with a 10-fold cross validation. The average 10-fold cross validation accuracy was calculated to be 0.854. The average 10-fold precision, recall, and F1 score was also calculated and determined and is provided in Table 4.

Table Metrics for Naïve Bayes Trained on TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1 Score** |
| Ham | 0.848 | 0.858 | 0.852 |
| Spam | 0.860 | 0.851 | 0.854 |

The metrics shown in Table 4, and the overall accuracy show that the TF-IDF matrix performs significantly worse in a Naïve Bayes classifier than the other models created. All of the metrics were in the 80% range, whereas all of the metrics in the previous three models scored greater than 90%.

## TF-IDF Matrix and SVM Classifier

The last experiment conducted was with the sklearn package and a TF-IDF matrix. To create the TF-IDF matrix, I had to reprocess the text from the email files. First two lists were created for spam emails and ham emails and each email was placed in the appropriate list, unprocessed, along with the class of the email. For this classification task, spam emails were given a class of 1 and ham emails were given a class of 0. After the two kinds of emails were processed, they were combined into one single list, and then randomly shuffled.

After the shuffling, two more lists were created, one with the text of the emails and a second with the class of the emails. This way the features and the labels were in the same index in their corresponding list. Then a TfidfVectorizer object was created and took in the parameters of tokenizer, stopwords, lowercase, maximum document frequency, and max features.

The tokenizer chosen was the nltk.word\_tokenize tokenizer to stay somewhat consistent with the previous experiments. The same stopwords in the NLTK stopwords corpus was also used as the stopwords, all words were picked to be lowercase, and the vectorizer was only to choose the 1,000 most frequently occurring words. Additionally, the maximum document frequency was chosen to be 0.9, meaning that a token could not occur in more than 90% of the documents to be included in the TF-IDF matrix. With these parameters, the TF-IDF was fit and transformed with all the documents collected. Additionally, all the labels were converted into a NumPy array.

After the features were converted into the TF-IDF matrix and the labels were converted into the NumPy array, I split the data into a training set and a test set with 20% of the data acting as the hold out test set. Using the training set, I created a grid search to find the best parameters to for an SVM model to classify the emails and determined that has a cost value of 5 and uses the linear kernel. The accuracy on the holdout set was calculated to be 0.973.

Using the cross validation function in the sklearn package, I was able to calculate that the average 10-fold cross validation accuracy for this model is 0.972. I was not able to calculate the average precision, recall, and f1 score in a 10-fold cross validation for both class labels, however I calculated the precision, recall, and f1 score for each class using the trained model for both of the classes. The precision, recall and f1 score were calculated on predicting the entire dataset and is provided in Table 5.

Table Metrics for SVM Trained on TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1 Score** |
| Ham | 0.997 | 0.986 | 0.991 |
| Spam | 0.986 | 0.997 | 0.991 |

The metrics provided in Table 5 show that the SVM model with the TF-IDF matrix provides the most balanced model for each class. However, this model has a recall for ham emails that performs worse than the baseline model; as stated for the task of classifying emails, we may want to prioritize the recall of ham emails rather than overall accuracy.

# Conclusions

When it comes to classifying emails between spam and ham, two classifiers stand out among the four tested. The SVM classifier with the TF-IDF matrix provides the most accurate option with an average 10-fold cross validation accuracy of 0.972. This accuracy was 1.5% higher than the next best classifier. However, the baseline model trained with the Naïve Bayes algorithm in the NLTK package was able to score a recall of 0.999 on the ham dataset. If we were ok with a 2% drop in overall accuracy in exchange for accurately classifying (and not putting in the spam box) nearly all of the real emails, then the baseline model would be the best classifier.

If I were to have to choose which classifier to implement in order to best sort out a person’s email, I would choose the baseline model with an accuracy of 0.950. While this was not the most accurate model, the recall on ham emails performed the best (0.999), meaning nearly 100% of all real emails were captured as emails. I find this feature of the model to be more useful than overall accuracy because there is a much greater harm caused by putting a real email in the spam box, than there would be from placing a spam email in a user’s inbox. The choice is pretty easy when deciding between a user miss an important actual email and a user having to delete a spam email from an inbox.