Predicting Spam Emails Through Data Mining

Tyler Conn, Jeff Dye, Rahul Isaac - CSC 272 - April 26th, 2015

Introduction

Email has become a ubiquitous technology since its inception in the 90's as "electronic mail" - where checking one's email inbox is perhaps increasingly more commonplace than checking a physical inbox. However, with its prevalence as a means of communication, the potential for misuse is also high - mass, unsolicited, fraudulent, malicious, and occasionally explicit emails are spread through the Internet daily. Requiring nothing more than a list of emails, spammers can send content en masse with little to no overhead. These emails are potentially more than just space in an inbox, they can also be a vector for attack, where unsuspecting users can be phished for information or open a malicious attachment.

Email providers have since developed systems to combat the flood of spam emails. More often than not, spam emails can be fairly easily identified from genuine emails by the human eye (not considering the unsuspecting victims), but these systems look to detect spam before it ever reaches the user's consideration. It was our curiosity about these systems that led us to our topic in the first place: what are the attributes/characteristics that can be used to classify a spam email from a genuine email?

Data Description

(https://archive.ics.uci.edu/ml/datasets/Spambase)
Collected by Hewlett-Packards Labs in California , our dataset utilizes numeric frequencies to classify a given email as spam or not. It contains 4,601 unique instances and 58 attributes (57 continuous numeric and 1 nominal class label). 1,813 of the instances are spam (39.4%). Given the nature of the data, each instance contains numerous missing values as every email will not contain every attribute we are searching for. These missing values are substituted with with a 0 in the dataset. A handful of example attributes are included below to exhibit the various attributes we are analyzing (Table1). word_freq_WORD is given as the frequency of that word compared to the total words of the email (100 * number of times the WORD appears in the email/ total number of words in the email). word_freq_CHAR is achieved using the same algorithm while looking at total characters in the email. capital_run_length_average is the average length of uninterrupted capital letters, capital_run_length_longest is the longest length of continuous capital letters and capital_run_length_total represents the

total number of capital letters in the email. The last attribute in the dataset is the only nominal attribute that denotes whether the email is spam (1) or not (0).

Attribute	**Type**	**Description**
word_freq_your	Numeric	Frequency of a given word compared to the total words in an email.
char_freq_*	Numeric	Frequency of a given character compared to the total characters in an email.
capital_run_length_average	Numeric	Average length of continuous capital letters.
capital_run_length_longest	Numeric	Longest length of continuous capital letters.
capital_run_length_total	Numeric	Total number of capital letters in an email.
spam	Nominal	Whether the email is classified as spam or not.

Table 1

Data Preparation

The data preparation involved several steps. We started with our data in a raw text format, with the column information in a separate text file. We modified the names of the attributes to match a column format for a csv file, and combined this header with the data file, to create a Weka supported .csv. We let Weka handle conversion to the .arff format. We modified the “Spam” attribute to output “Yes” or “No” instead of 1 or 0, to make it less confusing. After this, we split the data into out training and testing sets. After some quick trials suggested an 80/20 split of training/testing was a good balance. To do this, we randomized (with seed 42, for reproducibility) the order of instances in the dataset to mix up pre-existing ordering. We separated the first 80% and last 20% into different files and verified that there was an appropriate mix of spam and not spam emails in both datasets.

Since our data was numeric, we could not run classification algorithms like Prism on it, so a discretized data set was the next natural step. We created both supervised and unsupervised discretization datasets to complement the numerical one. We also experimented with removing attributes that we thought would be unproductive, based on our own knowledge of spam emails. For example, the frequency of the number “85”, one of the attributes in the dataset, did not match our experience of spam emails. A final thought was grouping attributes into different categories: word frequencies, letter frequencies, numeric frequencies.

Apart from our main dataset, which was shadowed to only contain word frequencies, we'd wanted to also experiment with StringToWordVector in Weka. We found a set of emails in plaintext from Enron servers and put together a set of 100 instances from these - 50 spam and 50 not spam, into an .arff file with two attributes: emailString (string, of course) and spam (nominal, {yes,no}). For the email string attribute, an email's contents in plaintext (subject, and body, as they were preprocessed) was first stripped of each newline and quotation mark, then added it to our .arff between two quotes. Spam was merely a yes/no according to how the dataset had them separated, so the string, a comma and the class value were all we needed for each 'row' of our EmailString dataset. Additionally, StringToWordVector will produce an error and quit if the dataset already has an attribute named as one of the words it found, so some slight modifications to the attribute names might be in order to allow the filter to go through.

Data Analysis

General Models

To start off our analysis of the data, we decided to run the standard set of classifying models on our preprocessed training set. In order to obtain the model with the highest accuracy for classification, we ran ZeroR, OneR, NaiveBayesSimple, NaiveBayes, J48, and IBk (nearest neighbor). The ZeroR model took none of the attributes into account and simply made a prediction based on which class attribute had the higher frequency. OneR used the data to create rules based on one attribute that had the highest accuracy of classification. NaiveBayes models numeric attributes by a normal distribution and creates an estimate for classification. The J48 model uses a similar technique to OneR to create a decision tree that attempts to accurately classify a new instance. Lastly, IBk looks at an instances nearest (k) neighbors and selects an appropriate placement for new instances using cross-validation. For all six models we also ran them with ten fold cross-validation, against the training data, and with the 20% test data. We also analyzed the confusion matrices from each model. It was important to note which model produced the smallest number of false positives as getting a real email sent to spam would be the worst case scenario.

Discretized Models

While creating an unsupervised discretization was straightforward, we had trouble trying to use supervised discretization. After running the discretization filter on the data, and performing the 80/20 training/testing split, we found that all the algorithms performed far too well, with IBk nearing 100% accuracy. We were justifiably skeptical of such good results and soon realized that because supervised discretization by definition took into account the class attribute, the training and testing sets were not actually independent of each other. At first we were unsure how to solve this issue: trying to first split up the data and then run the discretization filter resulted in incompatible datasets. Eventually we used meta classifier FilteredClassifier, on algorithms like Prism that needed discrete values, as well as on J48 and IBk as an alternate approach.

Cost Matrices
While users prefer spam to not appear in their inbox (a false negative), they would certainly not want personal emails to be classified as spam (a false positive), and marked for automatic deletion. That is, there is a significant cost difference to the user between a false positive and false negative, which we would like reflected in our model. To achieve this, we make use of cost matrices, using the meta classifier, CostSensitiveClassifier. A cost matrix is simply a matrix whose entries represent the cost of an item being in that position in the confusion matrix. The main diagonal is filled with 0s as there is no negative cost for a correct classification. The other two positions refer to the cost of a false positive versus a false negative. Since classifying a personal email as spam is very costly, we experimented with a higher cost of a false positive.

0	1
5	0

This is an example of a cost matrix that states that a false positive is 5 times worse than a false negative.

Linear Regression
The class attribute, if the email is spam or not, is strictly speaking a nominal attribute. However, it can be treated as numerical, with 0 representing “Not Spam” and 1 representing “Spam”. Furthermore, since all other attributes are numerical, this is a natural place for a regression analysis. Using the LinearRegression classifier gives a line of best fit for the data. In other words, it outputs a linear equation that will give us the predicted value for the spam attribute, given all other attributes.

The regression formula gives a number, usually between -1 and 2. We specify a cutoff value, say .50, below which the email is classified as not spam and above which it is classified as spam. This approach has several advantages over other classification approaches. There is a natural method to reduce false positives, without using cost matrices - we can set the cutoff value to a higher number. Furthermore, the output value contains more information. Rather than a simple yes or no, the output value is essentially a probability. If the output is 1.3, a false positive is less likely than if the output is 0.6. Finally, the formula itself provides information: if the regression formula did not use some attributes, it would suggest that those attributes were not highly predictive.

StringtoWordVector

We wanted to explore a feature in Weka that we hadn't used in class quite yet - StringToWordVector, in order to add another perspective for analysis. One of the unsupervised filters in the preprocess tab, StringToWordVector is designed for datasets like our EmailStrings. This filter takes a String attribute and then converts it into a set of new attributes based on the occurrence of each word. By default, StringToWordVector will produce a binary output - did this instance contain this word or not? However, setting the 'outputWordCounts' flag produces attributes that are based on the frequency of each word in a given instance. From there, you can further specify a cap on the number of words you'd like it to consider, and a minimum frequency as a cutoff, allowing the user to narrow down resulting attributes and eliminate potentially unwanted word attributes. This filter is designed to split one string attribute into a new set of attributes based on the words found, creating an entirely new perspective on the dataset. From there, one is free to run any of the algorithms on it, we'd specifically chosen J48 to focus on with EmailStrings with 10-fold cross validation to account for the size of our dataset. In our eyes, J48 was the most robust choice in algorithm, both for the sake of the decision trees that it produces and also the ranking of attributes that could be implied from the branches of the tree itself - on both the occurrence and frequency datasets.

Results

General Models

Our analysis began with the suite of general models - thankfully the holdout split of our training and test data had a fairly similar distribution of spam versus not spam instances, shown by our ZeroR results. From there, we tried OneR to see which attributes stood out as the most identifying of a given instance, we saw that char_freq_! was chosen - logically, we can see that emails that are overly exclamatory can be caught as spam. We then removed that attribute and re-ran OneR to get an idea of which other attributes were effective: Char_freq_$, in terms of how many times an email references dollar amounts ($1,000,000USD), Capital_run_length_average, in terms of the length of capital letter runs in an email, and lastly Word_freq_your, where a certain threshold of direct references to something of yours/your can be seen as a targeted spam email.The results from NaiveBayesSimple and NaiveBayes were fairly predictable. We believe that as the algorithm was attempting to obtain the mean and standard deviation, our numbers were most often 0. This is likely the reasoning behind its lower accuracies in comparison to the other models. NaiveBayesSimple in particular, couldn't even run with 10 fold cross validation due the standard deviation of multiple attributes resulting in 0. In these tests, IBk was the second most predictive model behind J48. Its confusion matrix (Table 4) is nearly as accurate as J48 for minimizing false positives and its accuracies respectable. When run with the training set, we even reached our highest accuracy (99.92%) although it is not particularly useful as it is simply grouping the instances against itself.

In terms of pure accuracy, one can see how J48 stands out among the other models. Looking a bit deeper into the resulting model, we were very pleased with the confusion matrix it generated compared to the other basic models (Table 3) where it, by virtue of having a higher accuracy, was able to keep false positives to a minimum. Additionally, looking at the decision tree generated with J48, we noticed some similarities in terms of the attributes chosen for high nodes in the tree with OneR, char_freq_!, word_freq_your, and even the capital_run_length attributes. Notable distinct attributes in the tree were word_freq_000, where large numbers or sums of money appearing in an email could indicate spam, word_freq_money/word_freq_free, in which gratuitous references to money or things for free can definitely appear in spam emails.

Accuracies (%)

Classification Algorithm	**Training(Cross Validation)**	**Training(Use Training Set)**	**Test Data (20%)**
ZeroR	60.9239	60.9239	59.2834
OneR	77.9891	81.4946	79.2617
NaiveBayesSimple	Error	81.67	82.0457
NaiveBayes	79.4837	79.2391	79.5874
J48	92.07	97.17	94.25

IBk	K=1: 90.27
	K=3: 89.59
	K=5: 89.67
		K=1: 99.92
		K=3: 94.46
		K=5: 93.02
			K=1: 91.10
			K=3: 89.14
			K=5: 89.47

Table 2

To discover which model produced the least amount of false positives we observed each of their confusion matrices. Our top two models for the least false positives are listed below. After running the J48 model, 144 of the notSpam instances were classified as Spam and with IBk, 181 of the notSpam were incorrectly classified as Spam. This puts a significant preference on J48 as it put less real emails in the spam folder.

J48:

Classified as -->	**Spam**	**NotSpam**
Spam	347	28
NotSpam	25	521

Table 3

IBk:

Classified as -->	**Spam**	**NotSpam**
Spam	322	53
NotSpam	29	517

Table 4

Discretized Results

The unsupervised discretization results were very poor, so we quickly discarded the idea of using this further. The supervised discretization was more successful, with Prism giving us a respectable 90% accuracy.

Prism with Supervised Discretization:

Classified as -->	**Spam**	**NotSpam**
Spam	347	18
NotSpam	48	484

Table 5

Cost Sensitive Results

We experimented on several cost matrices, and settled on a 5:1 cost of misclassification of a false positive versus a false negative. While this did not greatly improve the performance of either IBk or Prism, we had excellent results with J48.

J48:

Classified as -->	**Spam**	**NotSpam**
Spam	297	78
NotSpam	8	538

Table 6

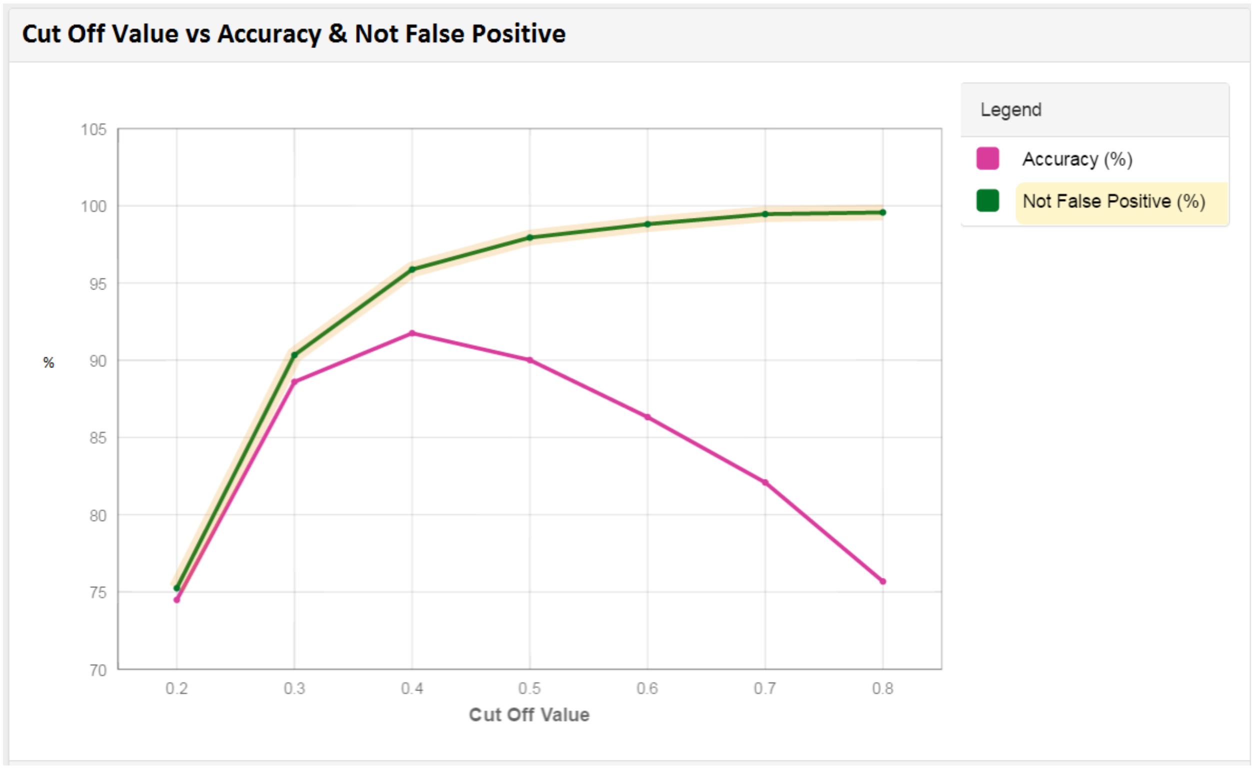
This gave us a 91% overall accuracy, and less than 1% false positives, the best results of any algorithm we tried.

Linear Regression

The regression formula did not use nine of the attributes in the formula, suggesting that these were not highly predictive of if the email was spam or not. These were word_freq_receive, word_freq_people, word_freq_report, word_freq_650, word_freq_lab, word_freq_857, word_freq_cs, char_freq_[, capital_run_length_longest. Most of these lined up with our intuition of spam emails. For actual classification, we tried cutoff values between 0.2 and 0.8. The key attribute we looked for in addition to accuracy, was the percentage not falsely classified as spam. We found that .5 was a suitable cutoff, with a 90% accuracy and only about 2% of emails falsely classified as spam.

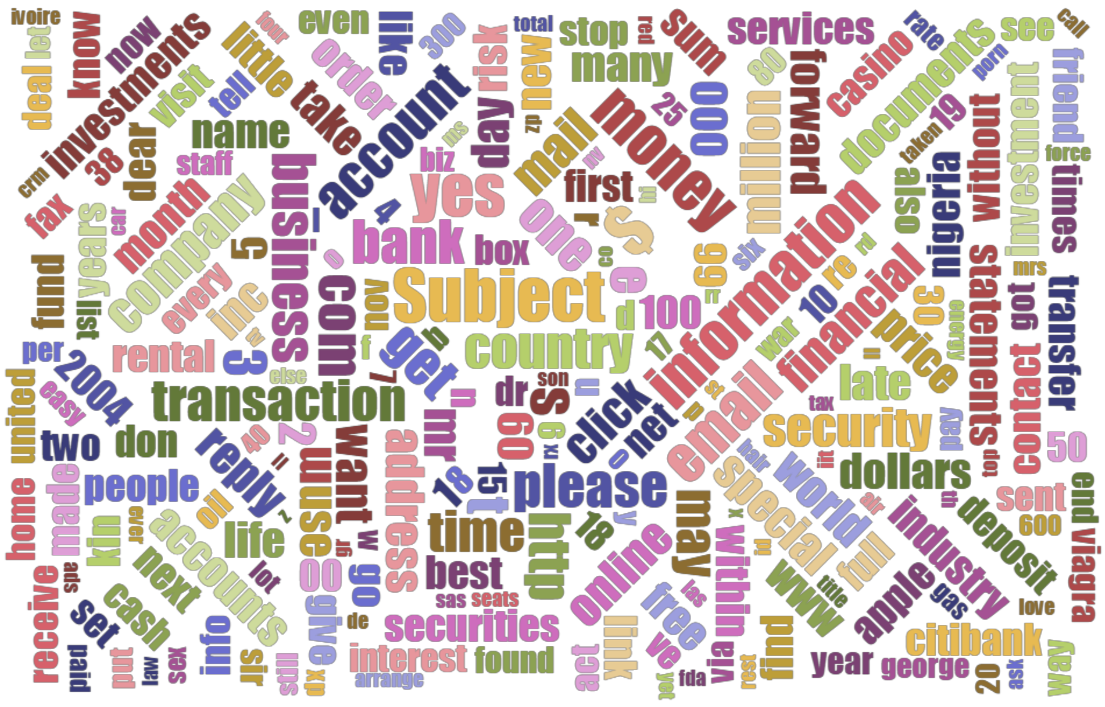
Linear Regression Classification with cutoff at 0.5

Classified as -->	**Spam**	**NotSpam**
Spam	302	73
NotSpam	19	527



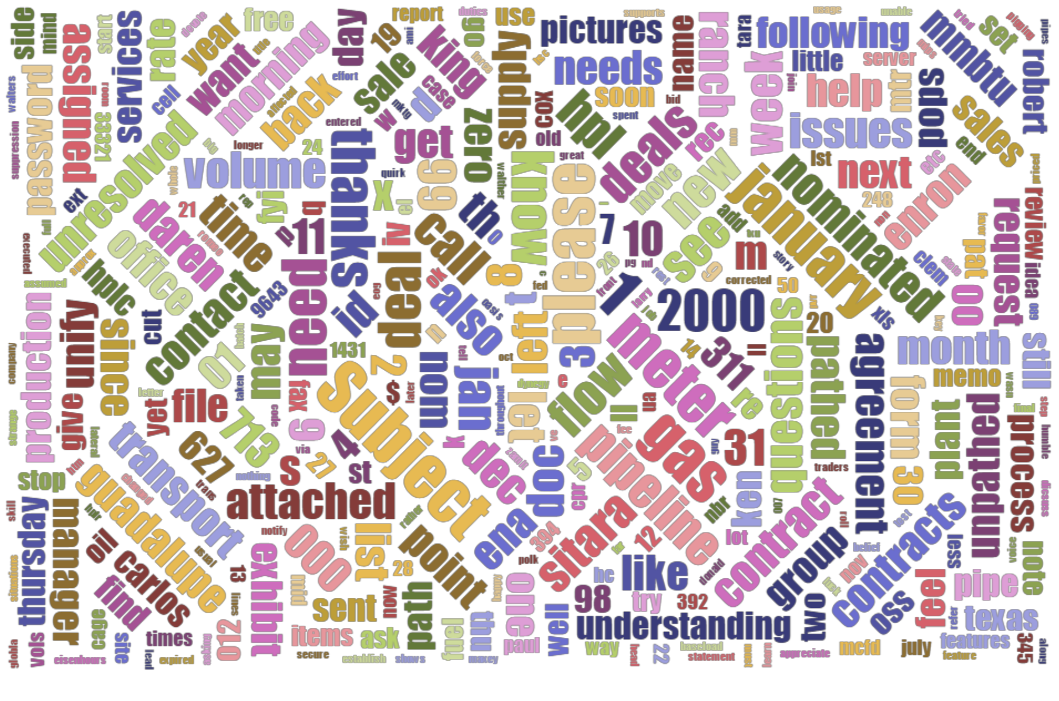
StringtoWordVector
In terms of the results of our post-StringFromWordVector'ed dataset, we found that J48 had similar accuracies in classifying on both the occurrence (77%) and the frequency dataset (74%). With the trees that they produced, money and http were the respective 'root' deciding attributes, both seem to be believable in terms of what would make a spam email stand out: direct references to money or web addresses/links (in excess). In terms of the comparison to be made between the two types of StringToWordVector processes, I would've thought that frequency would be a better indicator than something as binary as whether or not the word occurs in the string, but this could very well just be an outlier case. Additionally, one of the means of visualizations we'd decided upon was a word cloud for the emailStrings dataset, what is a simple yet effective means of visualizing said data. Split between the spam and not spam instances, one can fairly easily see not only the breadth of the terms in play, but also relative frequency - tied to the size of a given term. With this visualization, we found some similarities in the spam terms between those that appeared in our general models and what appeared largest in the word clouds - money, free, 000, your, all in addition to a few classic spam terms - viagra, security, receive, and cash, just to name a few.

Spam Word Cloud



Compared to the the word cloud from the genuine, non-spam emails, we can see where there is some overlap, but for the most part the business terminology (for an energy investment company) is pretty much par for the course: btu/mmbtu as a unit of energy, contract, flow, pipeline, and so forth.

Genuine Word Cloud



Conclusion

Looking at our results as a whole, though we found the most success with our cost-sensitive J48, there are still several limitations and shortcomings in our results that we need to address. Linear Regression too provided a useful tool that gave us a rough probability of an email being spam. All of the various attributes from characters, words, or capital run lengths were of value to modeling our data. Some standout attributes included, word_frequency_money, char_freq_!, and any of the capital_run_length attributes. Our dataset being strictly numerical did not give us certain information that we believe would have been useful. This includes the frequencies of misspelled words, grammatical errors, broken english, or even characters replaced with a non-english ‘equivalent’ (sexual vs SurhĚ'ΦvX∇¡HÜP09Ȁ∈ÿ±Ľ, with decoy characters between each major character). While this dataset was limiting, it also mitigates privacy concerns from the owners of the emails by shadowing the dataset to word/character frequencies, as no actual text is mined by us. Email is, to this day still one of the primary means of communication, but the potential for malicious applications exist. Filters work to protect the end user by learning from both spam and non-spam emails - in effect learning from human behavior with email in general.

Appendix

Dataset: https://archive.ics.uci.edu/ml/datasets/Spambase
Enron Emails: http://www.aueb.gr/users/ion/data/enron-spam/
Word Cloud Generator:
https://www.jasondavies.com/wordcloud/#%2F%2Fwww.jasondavies.com%2Fwordcloud%2Fabout%2F
Background: http://www.quora.com/Why-are-email-scams-written-in-broken-English
Academic Reference: http://research.microsoft.com/pubs/167719/WhyFromNigeria.pdf