CS 6490: Final Paper
A Survey of ML Algorithms for Spam Filtering

Nels Ballard, Ahmet Oguz, Michael Quigley

School of Computing, University of Utah

December 11, 2015

1 Introduction

According to research done by Nash Networks [4] in 2008, 70% of e-mail is estimated to be spam. Furthermore, up to employees can spend up to 16 minutes per week on spam management. While this may seem like a small number, an organization with 50 employees can waste more than $20,000 per year in lost employee time simply due to spam management. Thus, the costs of spam are high for most large companies, so there is a demand for more effective methods of spam filtering. We have attempted to analyze the relative performance of three types of machine learning-based spam filtering techniques: support-vector machines (SVM), decision trees, and AdaBoost.

To accurately assess the performance of these machine learning algorithms, we must also address the challenge of good feature selection. Feature selection is the process of selecting appropriate elements of a data set to measure in order to construct a model. Too many features can result in what is called "overfitting," where a machine learning algorithm will be able to operate perfectly on its training dataset, but its learned knowledge is useless for any other dataset. Too few features results in an algorithm that cannot accurately categorize inputs. For this project, our features consisted of words in various emails. The block diagram in Figure 1 illustrates the process of feature extraction/selection.

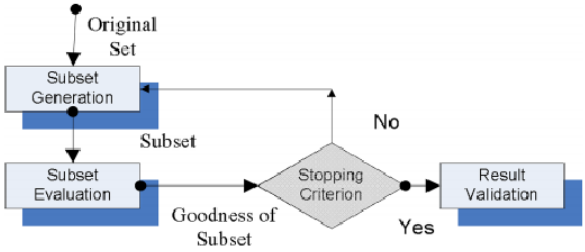

Figure 1: Feature Selection

2 Related Work

In the past, there have been many spam filtering solutions proposed. The support vector machine (SVM) is a common approach. Support Vector Machines were introduced by Drucker, Wu, and Vapnik [5]. The researchers experimented using SVM with decision tree boosters for binary representations, and the results were satisfying. Impressive results are reported using a word set with 3000 characters. After using iteration techniques, researchers found out that increasing the number of iterations increases the accuracy [6]. One of the most famous spam filtering applications, "SpamAssassin", is currently using SVM to identify spam e-mails.

3 Adversary Model

A prototype adversary model can be defined as a person or a group who intend to send spam e mails to a large amount of users. The purpose of the adversary, might be profit based by using commercial and advertisement spamming, getting reputation by his blog/website to be visited or it might be based on malicious purpose as the spam e mail might include malware (malicious software) in e mail content.

The adversary aims to get the users open/click on corresponding links in spam emails. There are some common techniques which is based on creating appealing statements in e-mails. 1-) Appealing Statements : can be given as sending spams that claim the user won a prize, special deals in products or creating a fake "Dating" statement. It is important to note that, the spams might try to trick the users into sharing personal information such as credit card parameters or passwords. The image below represents the most words that are used in spam e mails.

2-) Mail Bombing : The adversary is also able to include various types of malware links which potentially causes malfunctioning in victim's computers. The adversary is able to use specific software called mail bombing or "e-mail bombing", which replicates the same e mail by devastating amounts, which might take a huge portion of user's mailbox.

3-) Mixing characters/numerical :In terms of avoiding the filtering, there are common techniques which aims to trick the filter to classify the spam e mail as "not spam". Some of the methods are, adding alphanumerical strings in order to get the word out of the spam classification, such as"fr3e pr1ze for you" or "you won a free 1pad";however this method has become avoidable with pattern recognition techniques

4-) Taking advantage of global company names: Another methodology is based on tricking the user into clicking on a link, which makes the user thinking to clicking on a "facebook" link. One example can be given as sending an e mail saying that "authenticate your "Facebook" profile by clicking on this link "jk.facebook.gg"

5-) Embedding images on e mail content : By tricking the user, the adversary is able to add the .jpg .png or gif files, which might potentially contains inappropriate content. This method can be avoided by filtering the image extensions in our spam filtering once the e mail is identified as spam.

6-) Faking URL : The adversary might fake the url, which is based on leading an user to a different site than it appears on the e mail. The hyperlink which addresses to the website can be modified as anything; however, the actual link address might be deceiving. In order to avoid this situation, it is advisable to check the hyperlink, such as whether the hyperlink indicated as Amazon website really leads to the Amazon or not. By filtering the URL, the spam filter would be able to solve the problem.

7-) URL Hiding : There are numerous ways to represent a URL for an adversary to embed it into his spam, such as : Numerical IP address encoding: http://3329460759/ Hex IP : http://x6778a26/ Hex URL : http://

However, it is important to note that there is a common attribute of all of the listed hidden URLs, which is http and our filtering mechanism is able to recognize this pattern as a spam category.

4 Methodology

5 Design And Implementation

The project was implemented entirely in Python and can be found in a public Github [2]. For machine learning, the sklearn library [3] was used. This library supports numerous machine learning algorithms. We chose to use the Enron Email Dataset [1] for both our training and test set. We selected 12,000 spam emails and 12,000 ham emails from this set for training and testing.

The first step to using almost any supervised learning algorithm is feature selection. We decided early on to use the "bag of words" approach to feature selection. In other words, we created a text file of features (words). The occurrence of each feature was counted in each email and recorded into a feature vector for that email. The feature vectors along with their labels (spam or ham) were then all stored in files for later use so that they wouldn't have to be recomputed. Our feature selection was primarily manual in

the sense that we added words into the features list that we though might occur in a spam email. We also made the observation that in some spam emails, the word "free" might be spelled correctly, or it could have one of the following spellings: "fr33", "fre3", or "fr3e". To avoid overfitting (where there are too many features to get good results), we decided to make these different situations all the same feature. We achieved this by allowing the user to specify a regular expression that corresponds to a feature. For the situation listed above, the feature entry looks as follows:

$$fr[3e]\{2\}$$

We have not seen this approach in related work and feel it is useful in avoiding overfitting although we have not performed an extensive analysis to determine if this is the case. Once the features were extracted, we would shuffle them up before passing them to any learning algorithm. We also constructed the data so that 50% was training data and 50% was test data.

Once the features were selected, we decided to run a linear support vector machine (SVM) algorithm on the list of features. To save space, we will spare many of the details except to say that we performed cross validation over the following two parameters in SVM: γ and C. The parameter γ defines the margin which is the distance between the classifying line that SVM uses to separate the data, and the points nearest the line. The parameter C defines penalty for making a mistake during the training process.

Another algorithm we tried was the decision tree. This algorithm represents each feature as a node, and a count of that feature as the edge of a node. The leaves of the tree represent the predictions of the algorithm.

The last algorithm we tried was AdaBoost. In particular, we used an ensemble of decision trees in this algorithm to make predictions. This means that each element in a particular email's feature vector was the prediction of a decision tree. The decision trees were each slightly different so that they could produce a diverse set of results. We ran AdaBoost over fifty iterations to generate our results.

6 Results

The results from SVM cross validation are shown in Table 1. The highest accuracy was obtained when C was 1000 and γ was 0.1.

Table 1: SVM Cross-Validation Accuracies

	C=1	C=10	C=100	C=1000
γ=0.0001	75%	83%	85%	85%
γ=0.001	83%	85%	86%	86%
γ=0.01	85%	86%	86%	87%
γ=0.1	86%	87%	87%	87%

The results from a decision tree classifier with a maximum depth of 10 is shown in Table 2.

Table 2: Decision Tree Prediction Accuracies

Overall	Spam	Ham
85%	94%	76%

The results form the AdaBoost classifier are shown in table 3.

Table 3: Adaboost Prediction Accuracies

Overall	Spam	Ham
89%	92%	86%

7 Conclusion

Using the Enron dataset, we were able to successfully categorize emails into spam and non-spam with a high amount of accuracy. Our regular expression-based feature selection allowed us to avoid overfitting and categorize spam more effectively. From our results, we can conclude that AdaBoost is the most accurate spam filtering algorithm of those we tested.

In the future, we would like to explore the possiblilties of using our spam filtering algorithm on a production email server. The Enron dataset, while large, is quite old, and the nature of spam has changed significantly since 2001 (when Enron went bankrupt). We assume that the features needed to properly categorize modern spam will be different from those needed

on this dataset due to new types of spam emails and increasing email complexity. Our analysis of algorithm performance would be more relevant if it included performance on modern data, a project that we leave to future work.

References

[1] Enron email dataset. `https://www.cs.cmu.edu/~./enron/`. Accessed: 2015-12-9.

[2] Project code. `https://github.com/aoguz/CS6490-FinalProject-SpamFilter`. Accessed: 2015-12-9.

[3] scikit-learn. `http://scikit-learn.org/stable/`. Accessed: 2015-12-9.

[4] Choudhary, P., and Dave, M. Conception and evolution of spamming.

[5] Drucker, H., Wu, D., and Vapnik, V. N. Support vector machines for spam categorization. *Neural Networks, IEEE Transactions on 10*, 5 (1999), 1048–1054.

[6] Sculley, D., and Wachman, G. M. Relaxed online svms for spam filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (2007), ACM, pp. 415–422.