Spam Classification

Curtis Belmonte
curtislb@princeton.edu

Dorothy Chen
dschen@princeton.edu

Abstract

Identifying spam, or unsolicited email, is a problem of central relevance and importance due to the popularity of email as a means of electronic communication. In this assignment, we discuss various machine learning methods and use them to create spam classifiers based on a bag-of-words representation. We then analyze their performance within and across data sets to determine which classification models are the most well-suited to this problem.

1 Introduction and background

Email is a service used daily by an overwhelming number of people. But in addition to ordinary, benevolent users, advertisers and scammers take advantage of the popularity of email by sending out a huge volume of unsolicited emails, often referred to as spam. Spam filters aim to alleviate this problem by identifying these spam emails and sequestering them in a separate folder, to prevent them from clogging up users' inboxes.

2 Description of data and data processing

Our training data set consists of 22,500 spam and 22,500 non-spam emails from the trec07p data set. [6] We used the email processing script provided to define a vocabulary and create a bag-of-words representation for each email. The resulting vocabulary contained 9,579 words. Each of the classifiers was trained using these bag-of-words representations as features for the training data. The testing data set consists of 2,500 spam and 2,500 non-spam emails from the same corpus, processed in a similar manner to create bag-of-words representations.

In addition, we tested our models on 20,000 emails from the Enron data set [3] in order to infer how well our methods would generalize across data sets.

3 Methods

We used the following methods, as implemented in the Python scikit-learn package. [1]

3.1 Simple methods

1. Naive Bayes classifier, using multinomial implemenation
2. Decision tree, using Gini impurity scores and no max tree depth
3. Logistic regression, with l_2 penalty

3.2 Complex methods (ensemble learning)

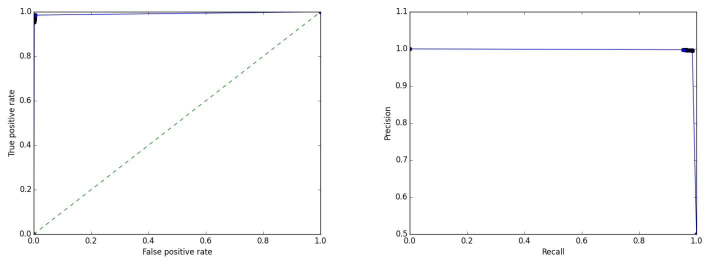
1. Random forests, with 10 trees and Gini impurity scores
2. AdaBoost, with 50 decision stumps as weak learners

4 Analysis of results

4.1 Evaluation metrics

For each classification method, we used stratified 5-fold cross-validation on the training set of emails. We used the same number of folds across all methods in order to facilitate comparison.

Preliminary comparison of results was conducted by generating the ROC curve for each classifier on the trec07p data set. The ROC curve for our multinomial naive Bayes classifier can be seen in figure 1(a) and indicates a high true positive and low false positive rate. The precision-recall curve can be seen in figure 1(b) and also shows that this classifier has extremely good performance on the data set. Curves for our other methods were extremely similar, and are not included in this report. The full set of ROC and precision-recall curves can be found in the "writeup/images" folder of the public GitHub repository for this project. [2] In an effort to view the data more clearly, we plotted the data on log scales. Specifically, we plotted the x-axis on a $\log_{10}$ scale in order to examine lower false positive rates. However, this did not significantly improve the quality of the graphs.



(a) ROC curve (b) Precision-recall curve

Figure 1: ROC and precision-recall curves for Multinomial Naive Bayes

We conducted further analysis using the accuracy (fraction classified correctly), precision, recall, and F_1 score metrics. Figure 2 shows these values for all classifiers on the provided training and test sets of the trec07p data set, and averages from running 5-fold cross-validation on the training set.

There are a few differences between performance on the entire corpus versus doing cross-validation. The most easily explained of these is the time: because cross-validation trains on only 4/5 of the available training data for each fold, it takes less time than training on all the data.

Classifier		Time (s)	Accuracy	Precision	Recall	F_1 score
Naive Bayes	Entire data set	1.51	0.9874	0.9963	0.9784	0.9873
	Cross-validation	1.25	0.9940	0.9984	0.9895	0.9939
Decision Tree	Entire data set	150.0	0.9948	0.9950	0.9936	0.9947
	Cross-validation	106.05	0.9856	0.9956	0.9752	0.9850
Logistic Regr	Entire data set	8.62	0.9974	0.9996	0.9952	0.9974
	Cross-validation	6.08	0.9968	0.9955	0.9981	0.9968
Random Forest	Entire data set	11.46	0.9968	0.9992	0.9944	0.9968
	Cross-validation	8.18	0.9976	0.9969	0.9984	0.9976
AdaBoost	Entire data set	642.94	0.9916	0.9976	0.9856	0.9915
	Cross-validation	482.88	0.9966	0.9970	0.9962	0.9966

Figure 2: Results from using various methods

4.2 Running time

As we can see in figure 2, using Naive Bayes is by far the fastest of the five methods we used. Logistic regression and random forests are also reasonably fast. Decision trees, however, are rather slow. This is likely due to the fact that we did not constrain the height of the tree. AdaBoost is prohibitively slow and does not offer any notable increase in performance with regards to accuracy. It therefore appears to be a suboptimal choice for solving this problem.

4.3 Feature selection results

We tried selection of 5, 10, and 15% of features. Performance among the three was best at the 5% level of selection; we have included these results in figure 3. In comparison to training with the entire set of features, performance was slightly worse in terms of accuracy, precision, recall, and F_1 after performing feature selection. However, run-time also decreased dramatically for most methods.

An exception to running time decreasing drastically is logistic regression. While its running time decreased after feature selection, it did not scale linearly with the number of features. For example, the running time for naive Bayes dropped from 1.51 to 0.08 seconds (about 5% of its original run-time) after feature selection. In contrast, the running time for logistic regression only dropped from 8.62 to 6.01 (69.7% of its original runtime).

Because this decrease in performance is most likely not as significant as the benefit of a much lower run-time in most situations, feature-selected training and testing is probably preferable. In situations that require high accuracy/precision, however, using the full feature set would still be better.

Classifier		Time (s)	Accuracy	Precision	Recall	F_1 score
Naive Bayes	Entire data set	0.08	0.9526	0.9752	0.9288	0.9514
	Cross-validation	0.07	0.9629	0.9636	0.9632	0.9630
Decision Tree	Entire data set	2.96	0.9936	0.9952	0.9920	0.9936
	Cross-validation	2.19	0.9925	0.9956	0.9894	0.9925
Logistic Regr	Entire data set	6.01	0.9938	0.9968	0.9908	0.9938
	Cross-validation	3.13	0.9954	0.9954	0.9954	0.9954
Random Forest	Entire data set	0.82	0.9972	0.9984	0.9960	0.9972
	Cross-validation	0.60	0.9976	0.9972	0.9981	0.9976
AdaBoost	Entire data set	26.26	0.9868	0.9959	0.9776	0.9867
	Cross-validation	17.88	0.9956	0.9965	0.9947	0.9956

Figure 3: Results from using various methods after performing feature selection

4.4 Important features

During feature selection, we selected the "best" 5% of features. Features were ranked by statistical significance, which was found using ANOVA analysis. We also looked at the top 20 features for the trec07p data set according to this analysis. Most of these words appeared to be parts of email headers. Some examples include: "unsubscribe", "list", "bounce", "subject", and "mailto". This suggests that a possible extension is examining these email headers more closely. Two of our features are actually "spam" and "spamassassin", suggesting that these emails may already be labeled as spam or are announcing themselves as "not-spam", which may have impacted performance.

4.5 False negatives versus false positives

For the problem of spam classification, it is generally better to have false negatives than false positives. Whereas a user can ignore a spam email in their inbox, they are unlikely to explore their "spam" folder. Therefore, non-spam emails classified as spam are likely to be missed entirely. In general, the added hassle of having to deal with an occasional spam email outweighs the cost of missing an important email. Thankfully, all of our classifiers yielded far more false negatives than false positives for the trec07p data set.

4.6 Hard cases to classify

We considered an example "hard" to classify if it was misclassified by more than one classifier. Hard false positives were rare, but appear to mostly be emails sent to mailing lists.

The only cases that were misclassified by all of our classifiers were false negatives. Hard examples included emails whose entire bodies consisted of long strings of letters (such as text representations of images). Because we used a bag-of-words representation, such long strings would be treated as unique and not provide any information about whether an email was spam. Other hard examples were very short emails. These might be misclassified because there is simply less content for classifiers to work with. Finally, one spam email was written entirely in German, which the classifiers did not catch because most, if not all, of the other emails in this data set were written in English.

5 Generalization across datasets

We trained a our naive Bayes and logistic regression classifiers on the trec07p training data and tested them on the Enron data set. The results are summarized in figure 4. The classifiers did much worse than they when tested on the trec07p set. This indicates that the classifiers learned data set-specific features rather than general characteristics indicative of spam.

Feature selection did not help at all and actually worsened performance. While feature selection is intended to increase performance by decreasing overfitting, that did not happen in this case. This might be because the "best" features that were chosen could all have been data set-specific features, which would actually increase overfitting of the models.

Classifier	Feature selection?	Time (s)	Accuracy	Precision	Recall	F_1 score
Naive Bayes	No	1.46	0.6691	0.7610	0.4930	0.5984
	Yes	0.08	0.4876	0.4847	0.3922	0.4336
Logistic Regr	No	8.48	0.4076	0.4426	0.7126	0.5460
	Yes	5.88	0.3147	0.3811	0.5939	0.4643

Figure 4: Results from using classifiers trained on trec07p and testing on the Enron data set

6 Conclusion and possible extensions

For the problem of spam classification, logistic regression seems to be the best choice to use of the five methods we explored. It had extremely high accuracy, precision, and recall and also had reasonable running time. Furthermore, its running time scales sublinearly with respect to the number of features. AdaBoost is not a good choice for this problem due to its large running time and its lack of increased performance relative to other possibilities.

Feature selection presents a tradeoff between correctness and computational time, but correctness suffers by a far smaller margin than computational time is benefited. In most, and especially time-sensitive situations, using feature selection would definitely be preferable.

We feel that using a different model to represent the emails' content would be an interesting and potentially useful extension. While the bag-of-words model was certainly effective, many features were lost; in particular, we feel that things such as word order, capitalization, and punctuation are especially important in language and that their exclusion may have negatively impacted performance. Word order is important because it encodes meaning and context of sentences–single words on their own may be ambiguous. This could be accomplished using a bigram model, as discussed by Cormack et al. [4]. Capitalization also matters, as emails in all capitals, for example, are probably more likely to be spam. Punctuation matters because emails with punctuation are more likely to have proper grammar, and correct grammar intuitively seems indicative of non-spam (or, at least non-computer generated) emails.

One other feature is looking for attachments on files and possibly also looking at their types/file extensions. This would help solve the issue we ran into with image attachments that ended up being spam but were missed by our classifier.

It also may be beneficial to compare the words we get to actual English words. This would help remedy the issue with missing spam emails that just consisted of long strings of random letters or that were written in other languages.

Furthermore, because false positives are worse than false negatives, it may be helpful to penalize false positives more harshly during training to account for this fact. However, some sources, such as Zhang et al., suggest that heavily penalizing false positives may actually reduce performance in some cases. [5]

7 References

[1] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, et al. (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12: 28252830.

[2] https://github.com/curtislb/SpamClassifier

[3] Bryan Klimt and Yiming Yang. Introducing the enron corpus. In CEAS, 2004.

[4] G. V. Cormack, G. Hidalgo, E. P. Snz, and J. Mara. Spam Filtering for Short Messages. In CIKM, 2007.

[5] Le Zhang, Jingbo Zhu, and Tianshun Yao. An evaluation of statistical spam filtering techniques. ACM Transactions on Asian Language Information Processing (TALIP) volume 3 issue 4: 243-269, 2004.

[6] Cormack, G. V. (2007). TREC 2007 spam track overview. In Proc of TREC 2007: The 16th text retrieval conf.