Spam Classification

Curtis Belmonte
curtislb@princeton.edu

Dorothy Chen
dschen@princeton.edu

Abstract

Spam, or unsolicited email, is becoming an increasingly large problem as email becomes more and more popular. In this assignment, we discuss various machine learning methods and use them to create spam classifiers. We then analyze the results and effectiveness of these methods.

1 Introduction and background

Email is something used daily by a large number of people. Advertisers and other people trying to sell things have taken advantage of this fact by sending out unsolicited emails, which is referred to as spam. In response to this, spam filters were created to identify these emails and to keep them from clogging up inboxes.

2 Description of data and data processing

The training data set consists of 22,500 spam emails and 22,500 non-spam emails from the trec07p data set. We used the provided script to define a vocabulary create a bag-of-words representation for each email. The resulting vocabulary contained 9579 words. The classifiers are built using these bag-of-words representations as features for the training data. The testing data set consists of 2,500 spam emails and 2,500 non-spam emails from the same corpus, and they are processed in a similar manner to also create bag-of-words representations.

3 Methods

We used the methods implemented in the scikit-learn package. [1]

3.1 Naive Bayes, using multinomial implementation

As a baseline, we trained a Naive Bayes multinomial model on all of the training data using the default parameters provided by scikit-learn. The default parameters included Laplace smoothing, learning class prior probabilities, and no previously provided class priors.

Using this classifier, we predicted labels for the testing data and got an accuracy rate of 98.74%.

3.2 Decision tree

Training this classifier on all available training data and testing on the provided testing set resulted in an accuracy rate of of 99.5%. We also used the default parameters, which meant that splits were determined using Gini impurity, the best split at each node was chosen, and the depth of the tree was not constrained. However, nodes were only split if it contained at least two samples; a leaf could only exist if it had at least one sample.

Because of the large number of features and because the depth of the tree was not constrained, this method was significantly slower than the other two.

3.3 Random forests with 10 trees

We once again used the default parameters. This includes building 10 random trees using Gini impurity Using this method, we got 99.58% accuracy.

This method was much faster than the decision tree

4 Results

5 Analysis of results

6 Conclusion and possible extensions

7 Acknowledgments

8 References

[1] http://scikit-learn.org/stable/