Topic : Spam Filter
Developer : Shweta Garg

Problem Statement: Design a spam filter to distinguish between spam and ham emails effectively.

Approach: The problem is to identify whether a given mail is spam or non-spam (ham). To manually classify e-mails, we have trained ourselves to pick certain words to mark it as useful or abuse. This is because in spam mails generally some words occur more than usual no of times. For ex: Free, 1billion$, price used to occur in lots of spam mails. Apart from this, sender data, information present in headers, etc. helps to classify the mails.

The problem can easily be translated to a binary classification problem of machine learning. From perspective of spam filtering, spam mails are considered as +ve instances and ham mails as -ve instances. There can be multiple approaches to spam filtering like supervised, unsupervised or some hybrid approach. Here, I have used supervised learning approach of spam filtering. To classify the mails, features are the tokens/words present in the mail. To find tokens I have used the following approach. First, I divide the corpus into two parts as "ham" and "spam". I scan the full text, including headers, of each message of both parts. Then I perform some preprocessing steps, like lower casing, removing htms tags and attributes, replacing all url addresses with "httpaddr", replacing all email addresses with "emailaddr", replacing $ with "dollar", replacing numbers like 10 with "number", replacing multiple underscore with single underscore, removing = occuring because of word-wrap, etc. I remove all the stopwords occuring in the text with the help of a downloaded list of stopwords. I tokenize the text and get rid of any non-alphanumeric characters, punctuation tokens, numbers and single letters. I perform stemming of the words to get final tokens from the scanned text. Finally I prepare a pair of each token and count of the token in the mail. But all these preprocessing steps has their pros and cons in respect of false positives and false negatives which I will dicuss in the next section.

To build the model from extracted features, I have used Naive Bayes approach. Naive Bayes classifier assumes that the value of a particular feature is independent to the presence or absence of any other feature. Typically it is based on Bayes formula[3]:

$$p(C|F_1, F_2...F_n) \quad = \frac{p(C).p(F_1, F_2...F_n|C)}{p(F_1, F_2...F_n)} \tag{1}$$

where,

C: is the class, i.e, spam or ham
F_i: are different tokens extracted after preprocessing of mails, used as features
$p(C)$: is the overall probability that any given message belongs to class C
$p(F_1, F_2...F_n|C)$: is the probability that features $F_1, F_2...F_n$ are present, given the mail belongs to C
$p(F_1, F_2...F_n)$: is the overall probability that any given message contains features $F_1, F_2...F_n$
$p(C|F_1, F_2...F_n)$: is the probability of mail with features $F_1, F_2...F_n$ belongs to class C

Denominator is effectively constant as it does not depend on the class. Also, because of the naive independence assumption, equation1 effectively becomes as:

$$p(C|F_1, F_2...F_n) \quad = p(C) \prod_{1 \le i \le n} p(F_i|C) \tag{2}$$

With assumptions on distributions of features and types of attributes (boolean or frequency-valued), there are different types of Naive Bayes (NB) classifiers, like, Bernoulli NB with boolean attributes, multinomial NB with boolean attributes, multinomial NB with frequeny-valued attributes, etc. I have used python nltk NaiveBayes classifier which use multinomial NB with boolean attributes.

Since the learning process of Naive Bayes is very fast, this system is easily scalable to large systems.

Library Used: Used nltk tokenizer[5] to tokenise preprocessed mails, nltk PorterStemmer[5] to do stemming of tokens and nltk Naive Bayes classifier[5] to build the model.

Interesting Findings: While building the filter, I found some interesting things. These are related to different preprocessing steps which I discussed in previous section.

- Header contains a lot of important information data for spam filter. Using header data in features set improved precision by 8.4%.
- NOT lowering the case of email-content, improved precision by 1.5% .
- NOT Stemming words to their root improved precision by 1.2 %.
- Removing htms tags and attributes, improves precision on cost of recall with slight improvement in F1.
- Removing stopwords from email-content does not have any significant impact on results.

Results: I have used five fold cross validation and took average of all result metrics.

Table 1: Precision, Recall, F1 and Accuracy

Cross-Fold	Precision	Recall	F1	Spam-Accuracy	Ham-Accuracy
1	95.45	99.27	97.33	99.27	97.79
2	94.10	98.55	96.27	98.55	97.11
3	96.13	99.27	97.67	99.27	98.13
4	94.77	98.91	96.80	98.91	97.45
5	96.40	97.45	96.93	97.45	98.30
AVG	95.37	98.70	97.00	97.76	98.69

Possible Improvements:

- Currently I have used multinomial Naive Bayes with boolean attributes as nltk does not support other types of NB. But other flavours of NB can be tried using scikit or other package.
- I tried DecisionTreeClassifier, but it was taking a lot of time to get trained. Some further debugging or tuning can be done to make it work.
- Currently I am using all the tokens as feaures to classify. Rather only some significant features can be used to train the model.[2]
- In spam filter false positives are more risky as compared to false negatives. So biasing against false negative can be tried.[2]
- Preprocessing step can definitely be improved by looking each and every false positive instance and getting insites from there.
- Try to find probability of different class with the formula given at [4].

Bibliography

[1] A Plan for SPAM : http://www.paulgraham.com/spam.html

[2] Graham's web page for better Bayesian filtering ://www.paulgraham.com/naivebayes.html

[3] Naive Bayes spam filtering : http://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering#cite_note-12

[4] Graham's web page for the probability formula used in his spam algorithm : http://www.paulgraham.com/naivebayes.html

[5] Natural Language Processing with Python : http://www.nltk.org/book/