# York University at TREC 2005: SPAM Track

Wei Cao, Aijun An and Xiangji Huang
Department of Computer Science and Engineering, York University
Toronto, Ontario, M3J 1P3, Canada

**Abstract**

We propose a variant of the k-nearest neighbor classification method, called instance-weighted k-nearest neighbor method, for adaptive spam filtering. The method assigns two weights, *distance weight* and *correctness weight,* to a training instance, and makes use of the two weights when classifying a new email. The correctness weight is also used in the maintenance of the training data to make the training data more adaptive to the changes of spam characteristics. We submitted 4 spam filters to the Spam Track. Two of the filters are purely based on the instance-weighted kNN method. The two other filters combine the kNN method with other spam filtering and classification techniques. We report the official results of our submissions on the Spam Track evaluation data sets.

## 1 Introduction

As spam emails evolve with time, it is important for a spam filter to dynamically adapt itself to the changes of spam emails. We investigate the use of the k-nearest neighbor (kNN) classification method for adaptively filtering spam emails. The kNN method can incrementally learn from data by updating the training data as new spam emails arrive. For the 2005 TREC Spam Track, we submitted four spam filters. Two of the filters are based on a single kNN algorithm. The other two combine kNN with other spam filtering and classification techniques. The four filters are pre-trained with a set of training data. In this paper, we will first describe our method for data preprocessing and then describe the design of the four filters. The results of the filters on the Spam track test data sets are also given and analyzed.

## 2 Data Preprocessing

All the four submitted filters (referred to as *submission 1, submission 2, submission 3* and *submission 4*, respectively) involve a pre-train step. In this step, a filter is trained with a set of training data that include some public data obtained from SpamAssassin [4] and some private emails of our own. The training set consists of 2214 emails, of which 1659 are spam and 555 are ham. We chose to use a small training set with our kNN methods due to the time limit imposed by the Spam Track on classifying a new email.

Data preprocessing converts an email into a token stream. It processes each email in a training corpus and converts it into an attribute vector. Before classifying a new email in a test set, it also converts the new email into an attribute vector. In our spam filters, data preprocessing involves the following steps: *MIME parsing, tokenization, stop word removal, word stemming,* and *instance representation.*

### 2.1 MIME parsing and unpacking

The objective of this step is to parse and decode a MIME (Multipurpose Internet Mail Extensions) formatted email, and then retrieve the useful text part for tokenization. A MIME email may include binary data, such as encoded text parts, audio attachments, video attachments, pictures and PDF files. In the implementation of our MIME parser, we remove all the attachments except the encoded text parts that are decoded into readable text.

### 2.2 Tokenization

After MIME parsing and unpacking, the retrieved text part of an email is further converted into a set of tokens. A regular expression is used to segment a piece of text to tokens. The regular expression specifies the characters that can be included in a valid token. The other characters are used as delimiters between tokens. Different regular expressions may be used for different submissions. For example, alphabetical letters, 0-9, ‘$’, ‘@’, ‘-“, ‘_’, ‘!’, ‘.’ ‘\’, and ‘*’ are considered as valid characters in a token for submission 1; but only alphabetical letters, ‘$’ and ‘@’ are considered valid in submissions 3 and 4.

### 2.3 Stop word removal

All of our submitted spam filters remove stop words. But different filters may use different lists of stop words.

Submission 4 adopts a standard stop word list from Weka [5,6]. Submission 3 adds to this list 8 words that appear frequently in the header of emails, such as "message-id" and "reply-to". Submission 1 adds more of such words to the stop word list. Submission 2 uses what is used in SpamAssassin [4].

## 2.4 Stemming

We use Porter's stemming algorithm [2,3] to stem the words obtained from the previous steps.

## 2.5 Feature Selection

In our spam filters, each email in the training and test sets is represented by a feature vector $\langle a_1, a_2, \ldots, a_m \rangle$, where $a_i$ for $i \in [1,m]$ corresponds to a selected feature. Using selected features to describe an instance reduces the dimensionality of feature vectors. There are several methods for selecting features, such as document frequency (DF), information gain (IG), mutual information (MI), $\chi^2$-test (CHI), and term strength (TS). In [7], it is reported that IG and CHI are the most effective methods, and DF performs similarly. We use IG to select features based on the set of tokenized training data. Information gain measures the amount of information obtained for classification by knowing the presence or absence of a term in an email. The information gain of term $t$ is defined as:

$$G(t) = -\sum_{i=0}^{1} P(c_i)\log P(c_i) + P(t)\sum_{i=0}^{1} P(c_i \mid t)\log P(c_i \mid t) + P(\bar{t})\sum_{i=0}^{1} P(c_i \mid \bar{t})\log P(c_i \mid \bar{t})$$

where $c_0$ and $c_1$ denote the classes of emails, i.e., ham and spam. The information gain of each unique token in the training corpus is computed. The tokens are then sorted in decreasing order of their information gain. Only the first $m$ unique tokens are selected as features to represent emails. The number $m$ is pre-determined in the pre-train stage. The value of $m$ is set to 400 for submissions 1 and 4, 500 for submission 2 and 200 for submission 3.

## 2.6 Instance Representation

Given a set of selected features, $\langle a_1, a_2, \ldots, a_n \rangle$, an email can be represented by a vector of feature values $\langle v_1, v_2, \ldots, v_n \rangle$, where $v_i$ is the value for feature $a_i$. The value is determined by using term frequency of the feature in the email, which is the number of times the feature (which is a token) occurs in the email.

## 3 Instance-Weighted kNN

All of our submitted spam filters use a variant of the kNN algorithm, which we refer to as *instance-weighted kNN*. Our submissions 1 and 3 use only the instance-weighted kNN method. Submission 2 combines the instance-weighted kNN with SpamAssassin. Submission 4 is a hybrid spam filter combining instance-weighted kNN, naïve Bayesian and random forest algorithms [1].

## 3.1 Distance Function

We use a weighted Euclidean distance function to calculate the distance between a training instance and the instance to be classified as follows:

$$d(X,Y) = \sqrt{\sum_{i=1}^{m} w_i \times (x_i - y_i)^2}$$

where $X$ and $Y$ are two vectors representing two instances, $m$ is the number of selected feature, $w_i$ is the weight of $i$th feature, and $x_i$ and $y_i$ are the values of the $i$th feature in $X$ and $Y$ respectively. The weight of a feature is the normalized information gain of the feature.

## 3.2 Instance Weights

Two weights are associated with each instance in the training set, *distance weight* and *correctness weight*. They measure the goodness of an instance in classifying a new instance and are used in the classification function of the kNN method. The correctness weight is also used in the training set maintenance.

### 3.2.1 Distance Weights

The distance weight of a training instance is related to the distance between the training instance and the instance to be classified. It measures the similarity between the training instance and the instance to be classified. The closer the training instance is to the instance to be classified, the more influence the training instance should have on classifying the new instance. The distance weight is used when classifying a new instance with k nearest neighbors, which will be described in Section 3.3. We define the distance-weight of a training instance to be the inverse of the distance of the training instance to the new instance.

### 3.2.2 Correctness Weights

The correctness weight of a training instance reflects the historical influence of the training instance on classification of new instances. The more times a training instance gives a correct contribution to historical classifications, the higher correctness weight it obtains. The correctness weight of a training instance is initially set to 1 when the instance is added to the training set, and is updated when the instance is used in classifying a new instance. Also, the correctness weights of the training instances are preserved when re-training is performed.

The correctness-weight of an instance is updated as follows. When the instance is used as one of the k-nearest neighbors in classifying a new instance and the classification is correct, its correctness weight is updated according to whether it has made a correct contribution in the classification. If it belongs to the same class as the new instance (i.e., it has made the correct contribution), its correctness-weight is incremented by a constant; otherwise, it correctness weight is decremented by a constant. When a misclassification occurs with the training instance as one of the k nearest neighbors, the update of its correctness weight is reversed, that is, the weight is decremented when the two instances belong to the same class and incremented otherwise.

The correctness weight monitors the behavior of a training instance in classification. It is used for both training set maintenance and new instance classification. If the correctness weight of a training instance is lower than a threshold, the instance will be removed during the re-training stage.

## 3.3 Classification Function

To classify a new instance, we first find the set *K* of its *k* nearest neighbors with the distance function described in Section 3.1. Then we compute classification scores for the two classes, spam and ham. The classifications score for spam is defined as

$$CS(spam) = \sum_{i \in spam \cap K} CW(i) \times DW(i)$$

where *i* is a spam email in set *K*, and *CW*(*i*) and *DW*(*i*) are the correctness weight and distance weight of *i*, respectively. Similarly, the classification score for ham is defined as

$$CS(ham) = \sum_{i \in ham \cap K} CW(i) \times DW(i)$$

where *i* is a ham email in set *K*. The new instance is classified into the class with the higher classification score.

## 3.4 Incremental Training and Re-training

Whenever a new email is classified, the correctness weights of the k nearest neighbors are updated and the new email with its correct class label is added into the training set to include as much information about a personalized email stream as possible.

### 3.4.1 Maintenance of Training Set

To obtain a reasonable speed for classification with kNN, the size of the training data is maintained at a fixed range. A window size specified at the initialization phase defines the range. When the size of the training data exceeds the window size, the training instances with a correctness weight less than a threshold are removed. If no instance has a correctness weight less than the threshold, the oldest instances are removed to guarantee the training data set falls within the window size.

### 3.4.2 Re-training

For our kNN method, re-training is to re-select features based on the updated training data set and then represent each instance using the newly selected features. This is to reflect the evolvement of spam and ham characteristics so that the training data will be more adaptive to the test email stream than to the original training data collected by us. Note that the correctness weight of each training instance is kept during the re-training process so that classification history of each case can still be used later. For kNN, there is no need to re-train a model after re-selecting the features. But for other learning methods used in our submission 4, such as naïve Bayesian, a new model is trained after re-selecting the features. Since re-training is expensive, it cannot be performed very frequently due to the limit on classification time. In our submissions, the re-training frequency is once every 800 classifications for submission 1 and once every 1000 classifications for submissions 2, 3 and 4.

## 4 Integrating Instance-Weighted kNN with SpamAssassin

SpamAssassin is a mature and widely-deployed open source spam filter [4]. It uses a variety of mechanisms including header and text analysis, heuristic rules, Bayesian filtering, DNS blocklists, and collaborative filtering databases to filter spams. In our submission 2, we first employ SpamAssassin to produce a result and then use the instance-weighted kNN method to generate another result. The two results are combined together to make a final decision. In this integration, the kNN method makes use of the MIME parsing and tokenization results from SpamAssasin instead of using our own MIME parsing and tokenization methods.

Based on our preliminary experimental results, the instance-weighted kNN has better false negatives than SpamAssassin, but worse false positives. Therefore, we combine the result of SpamAssassin and the instance-weighted kNN as follows. If the classification result from kNN is spam, the score outputted from SpamAssassin is incremented by 1. If the incremented score is greater than 5, the instance is classified into spam; otherwise, the result is ham. If the result from kNN is ham, the result from SpamAssassin is used as the final result.

## 5 Integrating kNN, Naïve Bayesian and Random Forest Algorithms

Our submission 4 is a hybrid spam filter that combines decisions from the instance-weighted kNN, naïve Bayesian classifier and the random forest classifier [1]. The implementation of naïve Bayesian and random forest classifiers is taken from Weka [5,6], an open source data mining software package. When combing the results from individual classifiers, each classifier is equally weighted and the majority class label is returned as the final result.

## 6 Results and Analysis

The submissions were evaluated on four data sets: full, mrx, sb, tm. The first one is a public data set and the last three are private data sets. The sizes of the data sets are shown in Table 1. The results of our submissions in terms of the ham misclassification rate (ham%) and the spam misclassification rate (spam%) are shown in Table 2.

| Data set | Size |
|---|---|
| full | 92,189 messages |
| mrx | 48,000 private messages |
| sb | 7,000 private messages |
| tm | 170,000 private messages |

**Table 1. Test Data Sets**

| submissions / datasets | yorSPAM1 | | yorSPAM2 | | yorSPAM3 | | yorSPAM4 | |
|---|---|---|---|---|---|---|---|---|
| | Ham% | Spam% | Ham% | Spam% | Ham% | Spam% | Ham% | Spam% |
| full | 2.44 | 2.43 | 0.92 | 1.74 | 1.29 | 1.2 | 2.99 | 1.36 |
| mrx | 4.96 | 0.55 | 0.34 | 1.03 | 4.41 | 0.65 | 5.20 | 0.45 |
| sb | 1.19 | 13.81 | 0.14 | 23.64 | 1.36 | 15.32 | 0.77 | 91.35 |
| tm | 0.82 | 8.28 | 0.25 | 14.90 | 0.92 | 8.11 | 0.63 | 9.04 |
| *average* | *2.3525* | *6.2675* | *0.4125* | *10.3275* | *1.995* | *6.32* | *2.3975* | *25.55* |

**Table 2. Results of our Submissions on the Test Data Sets**

Based on Table 2, the best ham% is achieved by submission 2, which is the combination of instance-weighted kNN and SpamAssassin, and the second best one is submission 3, an instance-weighted kNN. The best spam% is achieved by submission 1, an instance-weighted kNN. The second best one in terms of spam% is submission 3.

Based on the ROC curves (not shown in this paper due to the space limitation), submission 2 achieves the best results on all the three private data sets. For the public data set (*full*), submission 2 achieves the best results in the low and medium ham misclassification rate (ham%) regions. In the high region of ham%, submission 4 is the best, which is the hybrid model combining kNN, naïve Bayesian and random forest. For the private data sets, the ROC curves for submissions 1 and 3 are very close. But for the public data set (*full*), submission 3 is much better than submission 1. Both submissions 1 and 3 use a single instance-weighted kNN algorithm. They differ in tokenization, re-training frequency, and the number of selected features, which indicates that data preprocessing techniques could greatly affect the performance of spam filters on some datasets, but not always. The performance of submission 4 in terms of ROC curves is better than submissions 1 and 3 on two private datasets, but worse than them on the other private set. For the public data set, submission 4 has similar performance with submission 3 and better than submission 1.

## 7 Conclusions

Based on the results, we conclude that integrating a variety of spam filtering techniques (such as in our submission 2) generally achieve better results than using a single method. Also, data preprocessing techniques, such as tokenization techniques, are also important to the performance of spam filtering.

## References

[1] Breiman, L., Random forests, Machine Learning, Vol.45, No.1, 5 – 32, 2001.
[2] Porter, M.F., An algorithm for suffix stripping, Program, 14(3), 130-137, 1980.
[3] Porter, M.F., The Porter Stemming Algorithm, http://www.tartarus.org/~martin/PorterStemmer/.
[4] SpamAssassin, http://spamassassin.apache.org/.
[5] Witten, I.H. and Frank, E. Data Mining: Practical machine learning tools and techniques, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.
[6] Weka 3: Data Mining Software in Java, http://www.cs.waikato.ac.nz/ml/weka/.
[7] Yang, Y., Pedersen, J,O.: A Comparative Study on Feature Selection in Text Categorization. Proceedings of ICML-97 14th Int Conf on Machine Learning. Nashville, US, 412-420, 1997.