1-20-2003

Scavenger: A Junk Mail Classification Program

Rohan V. Malkhare
University of South Florida

Scavenger: A Junk Mail Classification Program

by

Rohan V. Malkhare

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Date of Approval:
January 20, 2003

Keywords: spam filter, machine learning, feature extraction

Acknowledgements

Many thanks to Dr. Eugene Fink for suggesting me this problem, for providing me with data for measurements and for guiding me through the research. Special thanks to Dr. William Yerazunis for providing extremely valuable technical feedback.

Table of Contents

List of Tables

List of Figures

Scavenger: A Junk Mail Classification Program

Rohan V. Malkhare

ABSTRACT

The problem of junk mail, also called spam, has reached epic proportions and various efforts are underway to fight spam. Junk mail classification using machine learning techniques is a key method to fight spam. We have devised a machine learning algorithm where features are created from individual sentences in the subject and body of a message by forming all possible word-pairings from a sentence. Weights are assigned to the features based on the strength of their predictive capabilities for spam/legitimate determination. The predictive capabilities are estimated by the frequency of occurrence of the feature in spam/legitimate collections as well as by application of heuristic rules. During classification, total spam and legitimate evidence in the message is obtained by summing up the weights of extracted features of each class and the message is classified into whichever class accumulates the greater sum.

We compared the algorithm against the popular naïve-bayes algorithm (in [8]) and found it’s performance exceeded that of naïve-bayes algorithm both in terms of catching spam and for reducing false positives.

Chapter 1 – Introduction

Junk mail, commonly known as spam, has become more than just a daily nuisance for email users; it has become a national issue with newspapers regularly reporting on the latest efforts to fight spam. Legislation and technology are the two main tools being used to fight spam.

Content-based filtering is a key technological method to fight spam and numerous learning techniques have been developed to implement content-based filtering, with the naïve-bayes method ([8]) being the most popular. Most learning algorithms for spam classification include three main steps:

1. A mechanism for extracting features from messages.
2. A mechanisn for assigning weights to the extracted features.
3. A mechanism for combining weights of extracted features to determine whether the mail is spam.

Learning techniques use the word (or chains of words as in [9]) to implement step 1, probability of features in spam/legitimate collections to implement step 2 and the naïve-bayes theorem or some variant of the bayes rule to implement step 3.

Although this approach creates an accurate system for catching spam, the continuing problem of false positives prompts us to have a fresh look at each of the steps.

We have devised a machine learning algorithm that implements a unique mechanism for each of the three steps. The algorithm (named ‘scavenger’) implements the three steps as:

1. Features are created from individual sentences in the subject and body of a message by forming all possible word-pairings from a sentence.
2. Weights are assigned to the features based on the strength of their predictive capabilities for spam/legitimate determination. This predictive strength is based on the frequency of occurrence of the feature in spam/legitimate collections as well as on heuristic rules.
3. During classification, total spam and legitimate evidence in the message is obtained by summing up the weights of extracted features of each class. The message is classified into whichever class accumulates the greater sum.

The bayes rule and it’s variants have become the most popular method for implementing steps 2 and 3 in spam classification algorithms. We compared the algorithm against the naïve-bayes algorithm given in [8] and found it’s performance exceeded that of the naïve-bayes algorithm both in terms of catching spam and for reducing false positives.

We have implemented the algorithm as a filter running on a Windows PC. The filter operates for individual email accounts of IMAP mail servers.

Chapter 2 – Related Work

We shall restrict our focus to examining machine learning and text classification techniques as applied to the problem of classifying junk mail.

Cohen [1] devised a Rule-learning system called RIPPER which automatically generated "keyword-spotting" rules of the form

$$cs328 \leftarrow \text{“utexas”} \in from \wedge \text{“utexas”} \; \varepsilon \; to$$

The rule states that a message belongs to the folder *cs328* if the word "utexas" appears in both the *from* and *to* headers. The advantage of this system was that the rules were in a human-readable format and they could be manually extended. However, Provost [2] showed that classification accuracy of is very low as compared to statistical classification algorithms.

Pantel and Lin [3] employed the naïve-bayes algorithm to classify messages as spam or legitimate. Words were used as features of the messages and frequency counts of words in spam and legitimate collections were used to generate probability of a word being in spam and probability of a word being in a legitimate message. For classification, words were extracted from a message and the naïve-bayes method was used to calculate the probabilities of the message being spam and legitimate.

Sahami et al [4] also used the naïve-bayes method but differed from Pantel and Lin in their use of the mutual information measure as a feature selector to select words with the strongest resolving power between spam and legitimate messages as well as their use of domain-specific features of spam like specific phrases, overemphasized punctuation etc. as attributes of a message.

Katirai [5] compared the genetic programming with naïve-bayes for spam classification and discovered that while precision of classification was comparable to naïve-bayes, recall for genetic programming was poor.

Carreras and Marquez [6] used the AdaBoost algorithm for filtering spam and found that for a small corpus of emails, the performance of the AdaBoost algorithm is comparable to naïve-bayes.

Paul Graham [8] described a simple implementation of the naïve-bayes algorithm where probability of an email being a spam given a word occurrence is pre-calculated for each word on a training set of spam and legitimate messages. The incoming message is parsed and sorted out into fifteen words (or tokens) having the strongest probability that an email containing that word is either spam or legitimate. These probabilities are then combined using naïve-bayes method to give the probability of the email being spam.

Yerazunis [9] has created a powerful feature extraction technique where the incoming text is decomposed into short phrases of one to five words each while still maintaining the order of the words. For each feature, frequency of occurrence of that feature in both the spam and legitimate categories is counted and probabilities of spam and legitimate messages containing that feature are assigned as weights of the feature. The Bayesian chain rule is used to compute the probability of the mail being spam. Although the algorithm gives a highly accurate spam filter, it is computationally too expensive ([9]) for widespread implementation.

Chapter 3 – The Algorithm

3.1 Overview

Feature extraction is common to both the training and classification portions of the algorithm. Assigning weights to extracted features is specific to training and combining the weights of extracted features is specific to classification. Training consists of feature extraction followed by weight assignment and classification consists of feature extraction followed by combining the weights of extracted features.

3.2 Feature Extraction

For extraction of features, we use the ‘sentence’ of a message as the semantic unit and decompose individual sentences of messages into all possible word-pairings. We define a sentence as a series of words in a message delimited by either a ‘.’, ‘?’, ‘!’ , ‘;’ , ‘<’ or a ‘>’. If number of words becomes greater than a constant K, then we treat the group of K words as a sentence. We have set the value of K to 20. Commonly occuring words are skipped.

Consider the following sentence occuring in a legitimate message:

“There is a problem in the tables that are copied in the database.”

We form pairs by combining 1st word with 2nd, 1st word with 3rd…….1st word with the last word, 2nd word with the 3rd and so on. We also create pairs by reversing the order of words i.e. create a pair where the 2nd word comes before 1st, 3rd word comes before 1st and so on. Without considering the commonly occurring words, “problem tables”, “tables problem”, “problem database”, “database problem” etc. are the pairs that would be formed in the sentence above. Thus, if a sentence contains *n* words, then number of pairs would be *n.(n-1)*.

Instead of a sentence, we could use the complete message and create all possible pairs of words from the message or use the sliding window scheme as detailed in Bill Yerazunis CRM114 algorithm ([9]). However, the number of features generated out of such a scheme increases the computational complexity of the algorithm and makes it infeasible to create a usable filter. For example, the CRM114 algorithm ([9]) would create *2^(n-1)* features from a sentence as compared to *n.(n-1)* features created by the ‘scavenger’ algorithm.

We have compiled a list of commonly occurring words and we skip a word if it is a part of this list. These words are:

Hi hello dear regards thanks thank of into they she it been he in the how where microsoft us than like ascii us-ascii urn schemas vml office word xmlns smarttags http content path return hr no yes meta equiv border marginwidth marginheight leftmargin topmargin text when which what from as a an out you I am are is was by to br rowspan colspan on at for *to be our and but this that these many more all font face arial times verdana helvetica span there not can could would will if has have why who had with your or any my we so nbsp date content-type http-equiv width height from to reply-to subject fw fwd re mon monday tue tuesday wed wednesday thu thursday fri Friday sat Saturday Sunday sun jan feb mar apr may jun jul aug sep oct nov dec format flowed message charset td tr table href valign top bottom align title body head cellspacing cellpadding img src alt target class right left center div us-ascii http www return-path x-keywords content-disposition received message-id html font content-id let make put seem take be do have say will about among at between by down from in on over through to under up with as for of till than all any every no other some such that this I he you who and because but or if through while how when where why here again ever far near now out still there then well almost even much not only quite so very please yes*

Detailed steps for feature extraction are as follows:

1. The subject and body is extracted from the message. Only MIME parts having content-type as ‘text’ or ‘message’ are used for extraction. This ignores attachments.
2. Characters a-z, A-Z, 0-9, single quote and ‘$’ are used for formation of a word whereas all other characters are treated as word separators. Uppercase letters are converted to lower-case.
3. Sentences are created from the body and all possible pairs of words are created to form the features. The entire text of the subject line is treated as one sentence. . For HTML, the series of words within a tag (between ‘<’ and‘>’) is treated as a

sentence. A URL (web address) in a message is treated as a separate sentence. Commonly occurring words are skipped as are words which are composed of all digits. They are, however, retained while parsing the subject.

4. Extracted features are stored in a hash table in memory.

3.3 Assignment of Weights

Weights represent predictive strength of a feature. Weights are assigned to features depending on whether the feature is categorized as a ‘strong’ evidence or a ‘weak’ evidence.

The categorization of features into ‘strong’ and ‘weak’ pieces of evidence depends on the frequency of occurrence of the feature in spam/legitimate collections, the exclusivity of occurrence and on heuristic rules like the distance between words of the word pairing, length of each word in the word pairing and whether the word-pairing is from a subject or the body.

In Chapter 4, we experimentally determine the best choice of weights representing ‘strong’ and ‘weak’ evidences. Let us initially assume 0.9 as the weight representing ‘strong’ evidence and 0.1 as the weight representing ‘weak’ evidence.

Following criteria are used to determine ‘strong’ and ‘weak’ pieces of evidence:

1. Phrases occurring exclusively in one collection are treated as 'strong' evidence. For example, if the phrase ’medical director’ occurs even once in the legitimate collection but not at all in the spam collection, then it is assigned a legitimate weight of 0.9. For exclusively-occurring phrases in the spam collection, the count must be at least 3 to be considered as ‘strong’ evidence. A count below 3 is considered ‘weak’ evidence.
2. Experimentally, we observed that word pairs created from non-consecutive words have a lower predictive strength than word pairs created from consecutive words. Thus, a word-pair created from non-consecutive words is treated as a ‘strong’ evidence only if it occurs with sufficient frequency. We form a rule where a word-pair created from non-consecutive words is treated as a ‘strong’ evidence only if the percentage of frequency of it’s occurrence is at least 10% of the highest occurring frequency of word pairs.
3. Experimentally, we observed that if length of both the words in the pair is greater than 5, the pair is most likely a good predictor of the class. Such pairs are treated as ‘strong’ evidence.
4. All word pairs created from the subject of messages are treated as ‘strong’ evidence.

For better clarity, the assignment of weights to features is described by the pseudo-code below:

maxoccurspam = highest occurring count of an exclusively occurring feature in spam collection.

maxoccurlegit = highest occurring count of an exclusively occurring feature in legit collection.

```
For each feature in hash table
{
  if((feature.legitimateCount = 0)AND(feature.spamcount <> 0)) // exclusive in spam
 {
    feature.legitimateWeight = 0;

    if((strlen(feature.firstword) > 5) AND (strlen(feature.secondword) > 5))
      feature.spamWeight = 0.9; // strong evidence
    elseif ( isSubject(feature)) // feature occurs in subject of a message
      feature.spamWeight=0.9; // strong evidence
    elseif (feature.spamCount > = 3)
    {
      if(isConsecutive(feature)) // feature is a consecutive-word pairing
        feature.spamWeight = 0.9; //strong evidence
      elseif( feature.spamcount > 0.1 * maxoccurspam) //greater than 10% of max
        feature.spamWeight = 0.9; // strong evidence
      else
        feature.spamWeight = 0.1; // weak evidence
    }
    else
      feature.spamWeight = 0.1; // weak evidence
  }

  if((feature.legitimateCount <> 0)AND(feature.spamcount = 0)) // exclusive in legit
 {
    feature.spamWeight = 0;

    if((strlen(feature.firstword) > 5) AND (strlen(feature.secondword) > 5))
      feature.legitimateWeight = 0.9; // strong evidence
    elseif ( isSubject(feature)) // feature occurs in subject of a message
      feature. legitimateWeight=0.9; // strong evidence
    elseif (feature. legitimateCount > = 1)
    {
      if(isConsecutive(feature)) // feature is a consecutive-word pairing
        feature. legitimateWeight = 0.9; //strong evidence
      elseif( feature. legitimatecount > 0.1 * maxoccurlegit) //greater than 10% of max
        feature. legitimateWeight = 0.9; // strong evidence
      else
        feature. legitimateWeight = 0.1; // weak evidence
    }
    else
      feature. legitimateWeight = 0.1; // weak evidence
 }
}
```

3.4 Combining Weights

During classification, the total legitimate and spam evidence in the message is obtained by extracting features from the message, matching them with features collected during the training phase and summing up the weights of the matching features.

Total spam evidence = sum of spam feature weights.

Total legitimate evidence = sum of legitimate feature weights.

If total spam evidence >= M total legitimate evidence,*

then the mail is classified as a spam mail, else it is a legitimate mail.

We should note here that repeats of extracted features are ignored.

The constant M can have any value between 0.5 and 2.5 and can be used as a trade-off between the number of false positives and number of uncaught spam mails for individual email accounts.

The algorithm can be easily extended to include headers of messages. For headers, it is useful for the feature extraction technique to extract just words as features. Algorithm can also be extended to include exclusively occuring non-textual features like specific phrases, over-emphasized punctuation etc. These features should also be treated as 'strong' evidence.

Chapter 4 – Measurements

4.1 Methodology

We use the parameters of precision and recall for measurement of the accuracy of the algorithm. Spam precision is the percentage of messages in the test data classified as spam that actually are spam. Spam recall is the proportion of messages in the test data correctly classified as spam to the total number of spam messages in the test data. Spam precision gives us the accuracy of the filter with respect to false positives and spam recall demonstrates the capacity of the filter to catch spam.

For our measurements, we downloaded a corpus of 5538 unique spam mails from the *http://www.spamarchive.org* website and used 960 legitimate mails from Eugene Fink's mailbox. We performed experiments using K-fold cross validation for two values of K: K=5 and K=2 . We used smaller values of K so that the error estimate is pessimistically biased i.e. the algorithm faces a harder accuracy test than 10-fold cross validation.

For K= 5, the corpus of spam and legitimate messages is divided into five equal-sized sets and four are used for training and the last one is used for testing. Five such runs are carried out each time using one different testing set and four remaining sets for training. Accuracy values for these five runs are averaged out to get the final values.

For K= 2, the corpus of spam and legitimate messages is divided into two equal-sized sets and one set is used for training and the other set is used for testing. The testing and training sets are then swapped and another run is carried out. Accuracy values for these two runs are averaged out to get the final values.

4.2 Comparison with Naïve-Bayes

Naïve-Bayes is currently the most popular algorithm for junk mail classification. For benchmarking the ‘scavenger’ algorithm against naïve-bayes, we implemented the algorithm in Paul Graham’s popular ‘A Plan for Spam’ ([8]) and tested it using K-fold cross validation for the same data set. We implemented the algorithm for two separate methods of feature extraction. In one method, we used words+phrases as features. In the other method, mechanism of feature extraction was same as the ‘scavenger’ algorithm. For both the methods, weight assignment and combining of weights was implemented as given in Graham’s algorithm. In implementing the two variations of Graham’s algorithm, we did not use message headers during feature extraction so that the data from which features are extracted remains the same for ‘scavenger’ as well as for naïve-bayes.

Value of M, the thresold parameter, was kept at 1. 0.9 was assigned as the weight for ‘strong’ evidence and 0.1 was assigned as the weight for ‘weak’ evidence.

Table 1 gives the results of our experiments.

Table 1: Comparison with Naïve-Bayes

ALGORITHM	K=5		K=2	
	SPAM PRECISION (AVERAGE)	SPAM RECALL (AVERAGE)	SPAM PRECISION (AVERAGE)	SPAM RECALL (AVERAGE)
Scavenger (M=1)	100%	99.85%	99.92%	99.72%
Naïve-bayes (words+phrases)	100%	98.87%	99.80%	97.03%
Naïve-bayes (with scavenger feature extraction method)	100%	99.15%	99.65%	98.68%

As we see in the table, ‘scavenger’ gives a better performance over both the versions of naïve-bayes in terms of both spam precision and spam recall, especially for K=2.

On examination of the logs of extracted features for naïve-bayes, we find that most of the features selected during classification have probability values equal or close to 0.01 or 0.99. In the absence of any heuristics-based discrimination, 0.01 or 0.99 values get assigned to unimportant pieces of evidence. Thus, it becomes a simple race between which class (spam or legitimate) has more number of matching features and the class with more number of matching features ‘wins’.

On the other hand, the ‘scavenger’ algorithm has:

1. A powerful feature extraction technique that is able to collect and use a huge amount of *quantity* of evidence.
2. Usage of heuristic rules in addition to the frequency of occurrence of features to determine the *quality* of evidence. This serves to clearly discriminate between important and unimporant pieces of evidence.

We can infer that while using a powerful feature extraction technique as in ‘scavenger’ or as in Yerazunis’s CRM114 ([9]), the heuristics based discrimination between important and unimportant pieces of evidence would lend more accuracy to classification.

4.3 Experiments on M, the Thresold Parameter

We performed another set of experiments for plotting the ROC curve by varying the value of M from 0.25 to 2.5. Weights for ‘strong’ evidence and ‘weak’ evidence were kept at 0.9 and 0.1. We used K-fold cross validation for each value of M, keeping the value K at 2. Table 2 shows the results for this experiment. Figure 3.1 shows the ROC curve obtained by plotting the average percentage values of missed spam against the average percentage values of false positives for the various values of M.

The curve shows that as we increase the value of M, the effect on missed spam is very small but the effect on false positives is quite significant. For M=2, M=2.25 and M=2.5, we got zero false positives with a very high value of recall. Thus, values of M between 2 and 2.5 should be considered as 'safe' values for the algorithm. In section 4.4, we shall investigate a mechanism to automatically set the value of M during the training phase.

Table 2: Accuracy for Various Values of M

M	Missed Spam (%)	False Positives (%)	Spam Recall (%)	Spam Precision (%)
0.25	0	13.23	100	98.35
0.5	0.07	6.17	99.93	99.22
0.75	0.11	2.05	99.89	99.74
1	0.28	0.58	99.72	99.92
1.25	0.3	0.58	99.70	99.92
1.5	0.35	0.58	99.66	99.92
1.75	0.41	0.29	99.59	99.96
2	0.49	0	99.51	100
2.25	0.6	0	99.4	100
2.5	0.74	0	99.36	100

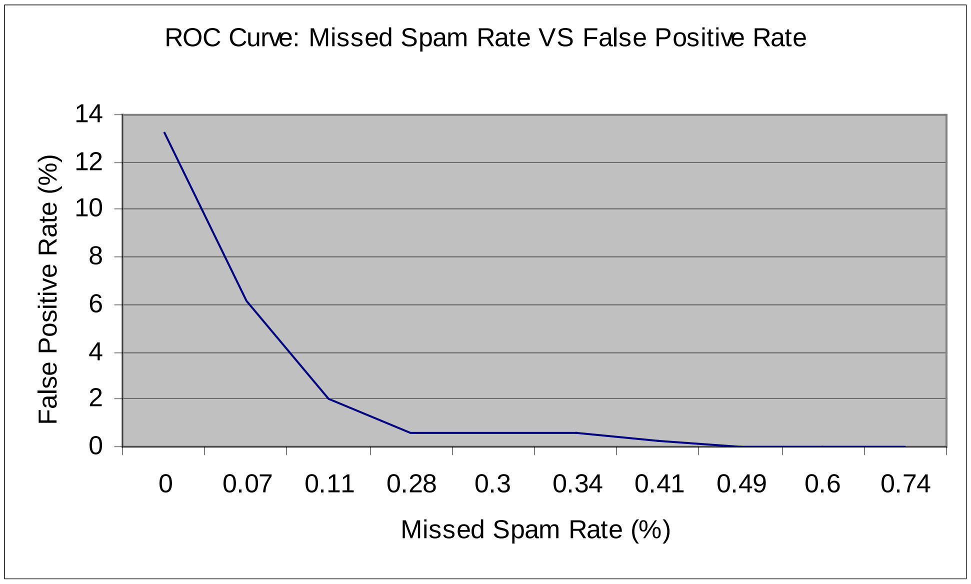


Figure 1: ROC Curve

4.4 Choosing the best set of weights

To experimentally determine the best set of weights to represent ‘strong’ and ‘weak’ evidences, we perform experiments where value of ‘strong’ evidence is kept at 0.9 and value of ‘weak’ evidence is varied from 0.1 to 0.9 in steps of 0.1. Value of M is kept at 2.

Table 3 shows the accuracy measurements for the selected weight combinations of strong and weak evidences.

Table 3: Accuracy for Different Sets of Weights

Weights for strong evidence /weak evidence	Missed Spam (%)	False Positives (%)	Spam Recall (%)	Spam Precision (%)
0.9/0.1	0.49	0	99.51	100
0.9/0.2	0.46	0	99.54	100
0.9/0.3	0.41	0	99.59	100
0.9/0.4	0.37	0	99.63	100
0.9/0.5	0.36	0	99.64	100
0.9/0.6	0.36	0	99.64	100
0.9/0.7	0.30	0.88	99.70	99.88
0.9/0.8	0.30	0.88	99.70	99.88
0.9/0.9	0.30	0.88	99.70	99.88

Weight Combination of 0.9/0.6 gave zero false positives and a missed spam rate of 0.36. Although weight combinations 0.9/0.7, 0.9/0.8 amd 0.9/0.9 gave a lower missed spam rate, rate of false positives was non-zero in these cases.

4.5 The TUNE Strategy

Bill Yerazunis has written about the increased accuracy obtained by using the 'Train until no errors' (TUNE) strategy ([18]). We attempted to apply the TUNE strategy to automatically set the thresold parameter M for the 'scavenger' algorithm. However, while applying the TUNE strategy, time required for training becomes very large since training iterations have to be repeated over all the messages in the training set until we get accuracy rate to the desired level or until a fixed number of iterations are over because of the uncertainty of convergence.

As we saw in section 4.3, increasing the value of M has a very small impact on the spam filtering rate but a significant impact on the rate of false positives. We use this experimental evidence to implement an alternative method for TUNE. The Training process is modified as follows:

1. In the first pass, extract features and store them in a hash table in memory as described in section 3.2. Assign weights 0.9 and 0.6 to features for 'strong' and 'weak' evidences. M is initially kept at 2.
2. In the next pass, validate only the set of legitimate messages in the training set. Increase M by an increment of 0.1 if there are any false positives.
3. In subsequent passes, validate only the false positives from the previous passes. Since we are increasing M on each iteration, it is not necessary to validate all previous correctly classified legitimate messages again.

4. Terminate when all legitimate messages are classified correctly or when the value of M reaches 2.5, whichever is earlier.

By using this variation of TUNE strategy, we 1> ensure that the training process converges quickly and 2> give priority to reducing the risk for false positives.

Chapter 5 – Implementation

We have created a windows PC-based spam filter that operates for individual email accounts on IMAP mail servers. The filter classifier runs as a windows service on a Windows NT/2000/XP PC.

The filter has three main modules:

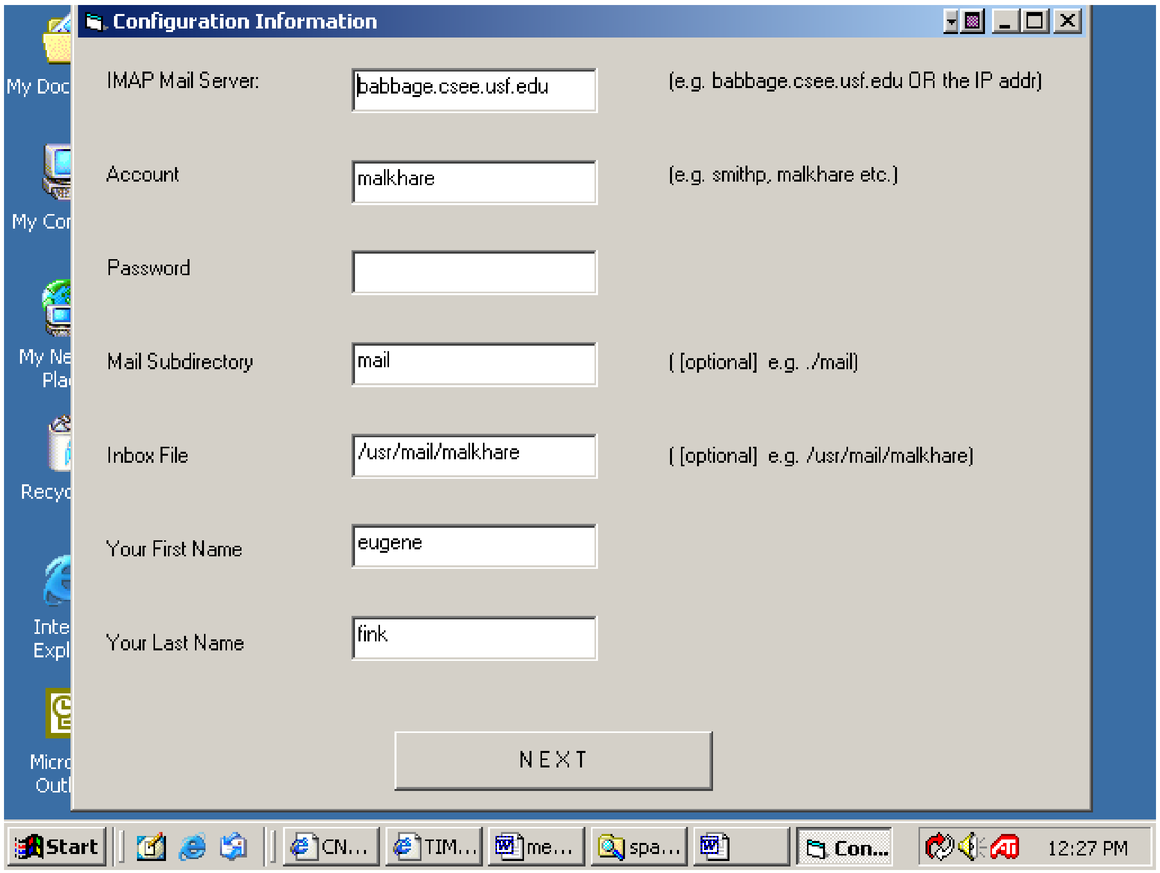
1. The configuration program: This module collects configuration data like the IP address of the mail server, account name, legitimate folders, Junk mail folders etc. and stores it in a text file in the installation directory on the PC.
2. The training program: This program is used for training the filter on the messages in the legitimate and spam folders selected by the user during configuration.
3. The classification service: This service is the actual classifier that runs as a windows service.

To connect to the mail server and to perform operations on mailboxes and messages, we have used email COM objects from quiksoft (http://www.quiksoft.com). The primary purpose of using these objects is the convenience in distinguishing between read and unread messages, parsing MIME parts of the messages as well the ease of reading headers, subject and the body of each message separately. Moreover, the objects can convert from different encodings into the standard 7-bit ascii thus presenting the textual content in a uniform manner.

5.1 Configuration Program

Spamconfig.exe in the installation/spamconfig directory is the configuration program. Figure 5.1 shows the main screen, called 'Configuration screen' of the program.

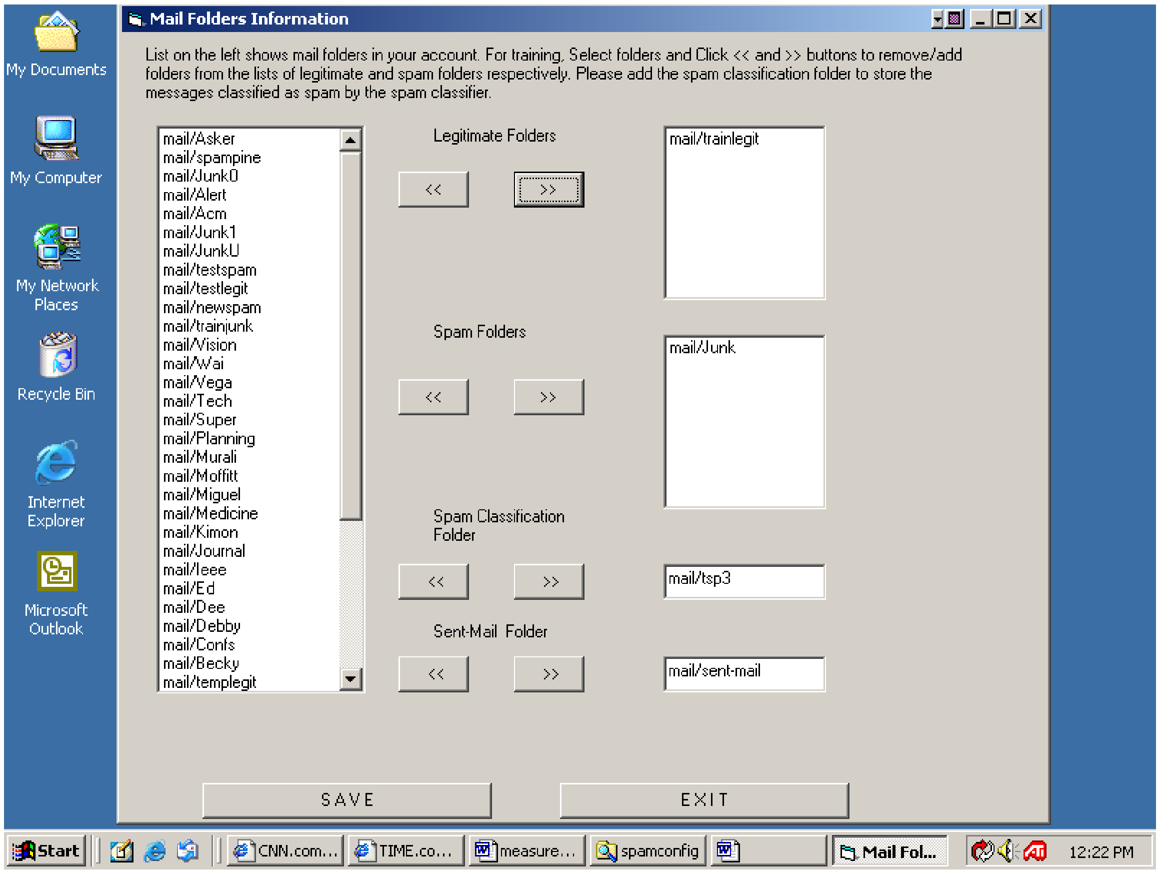
IP address of the mail server, account name, password, mail subdirectory, location, name of the Inbox file and the name of the user are collected and stored in a text file in the installation directory.



IMAP Mail Server:	babbage.csee.usf.edu	(e.g. babbage.csee.usf.edu OR the IP addr)
Account	malkhare	(e.g. smithp, malkhare etc.)
Password		
Mail Subdirectory	mail	([optional] e.g. ./mail)
Inbox File	/usr/mail/malkhare	([optional] e.g. /usr/mail/malkhare)
Your First Name	eugene	
Your Last Name	fink	

Figure 2: Main Screen for Configuration

On clicking the NEXT button, screen shown in figure 5.2 is displayed.



List on the left shows mail folders in your account. For training, Select folders and Click << and >> buttons to remove/add folders from the lists of legitimate and spam folders respectively. Please add the spam classification folder to store the messages classified as spam by the spam classifier.

mail/Asker
mail/spampine
mail/Junk0
mail/Alert
mail/Acm
mail/Junk1
mail/JunkU
mail/testspam
mail/testlegit
mail/newspam
mail/trainjunk
mail/Vision
mail/Wai
mail/Vega
mail/Tech
mail/Super
mail/Planning
mail/Murali
mail/Moffitt
mail/Miguel
mail/Medicine
mail/Kimon
mail/Journal
mail/Ieee
mail/Ed
mail/Dee
mail/Debby
mail/Confs
mail/Becky
mail/templegit

Legitimate Folders	mail/trainlegit
Spam Folders	mail/Junk
Spam Classification Folder	mail/tsp3
Sent-Mail Folder	mail/sent-mail

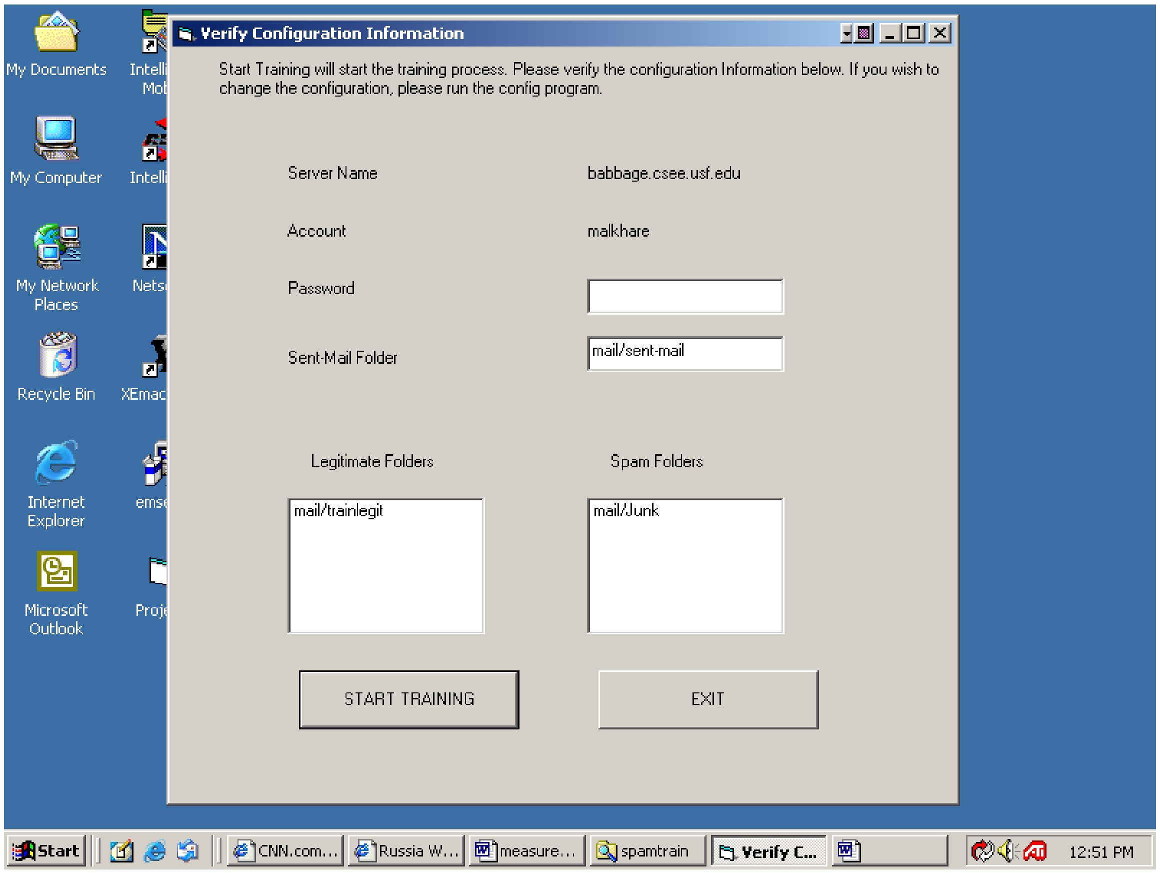
Figure 3: Selection of Folders for Training

All folders in the user’s mailbox account are listed in the list on the left. The user can select legitimate folders and spam folders to be used for training. Spam classification folder is the folder where spam mail would be stored after classification. The Sent-Mail folder is used as a legitimate folder for training. On clicking ‘Save’, the information is saved in text files in the installation directory on the PC.

5.2 Training Program

Spamtrain.exe is used to train the filter on the legitimate and the spam folders selected during configuration.

On running spamtrain.exe, the screen shown in Figure 5.3 is displayed. Folders selected for training are shown for confirmation in the 'legitimate folders' and 'spam folders' lists. On entering the password and clicking 'Start Training', the training process is started.



Verify Configuration Information

Start Training will start the training process. Please verify the configuration Information below. If you wish to change the configuration, please run the config program.

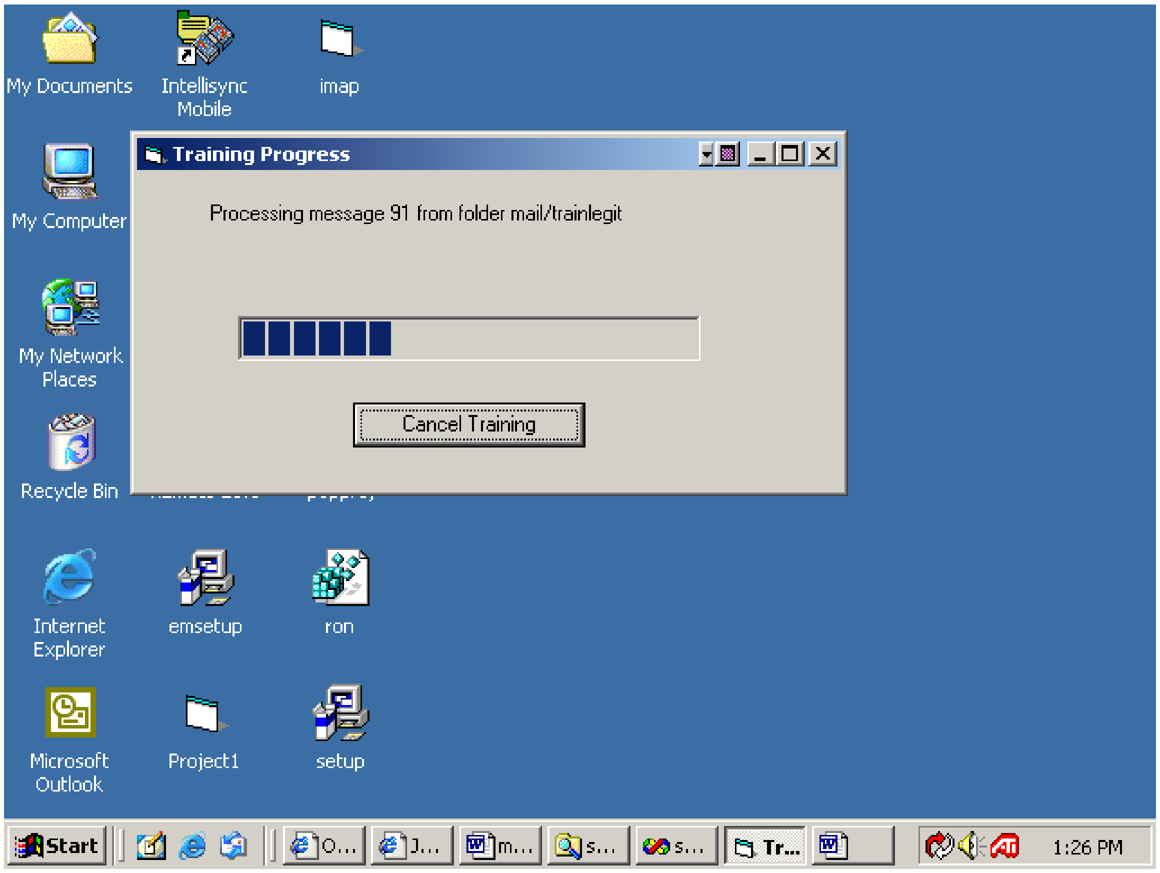
Server Name	babbage.csee.usf.edu
Account	malkhare
Password	
Sent-Mail Folder	mail/sent-mail

Legitimate Folders	Spam Folders
mail/trainlegit	mail/Junk

Figure 4: Main Screen for Training

A screen containing a progress bar is displayed. The progress bar indicates the progress of the training (in Figure 5.4). Once feature extraction is over, subsequent passes over the training set validate the legitimate messages for implementing the TUNE strategy.

The training process also collects legitimate and spam email addresses from the training folders. Once training is over, the entire hash table of features is written onto the *occurcount* sub-directory in the installation directory. Legitimate addresses are written onto the *legitaddr* and the spam addresses are written onto the *spamaddr* sub-directories.


Training Progress

Processing message 91 from folder mail/trainlegit

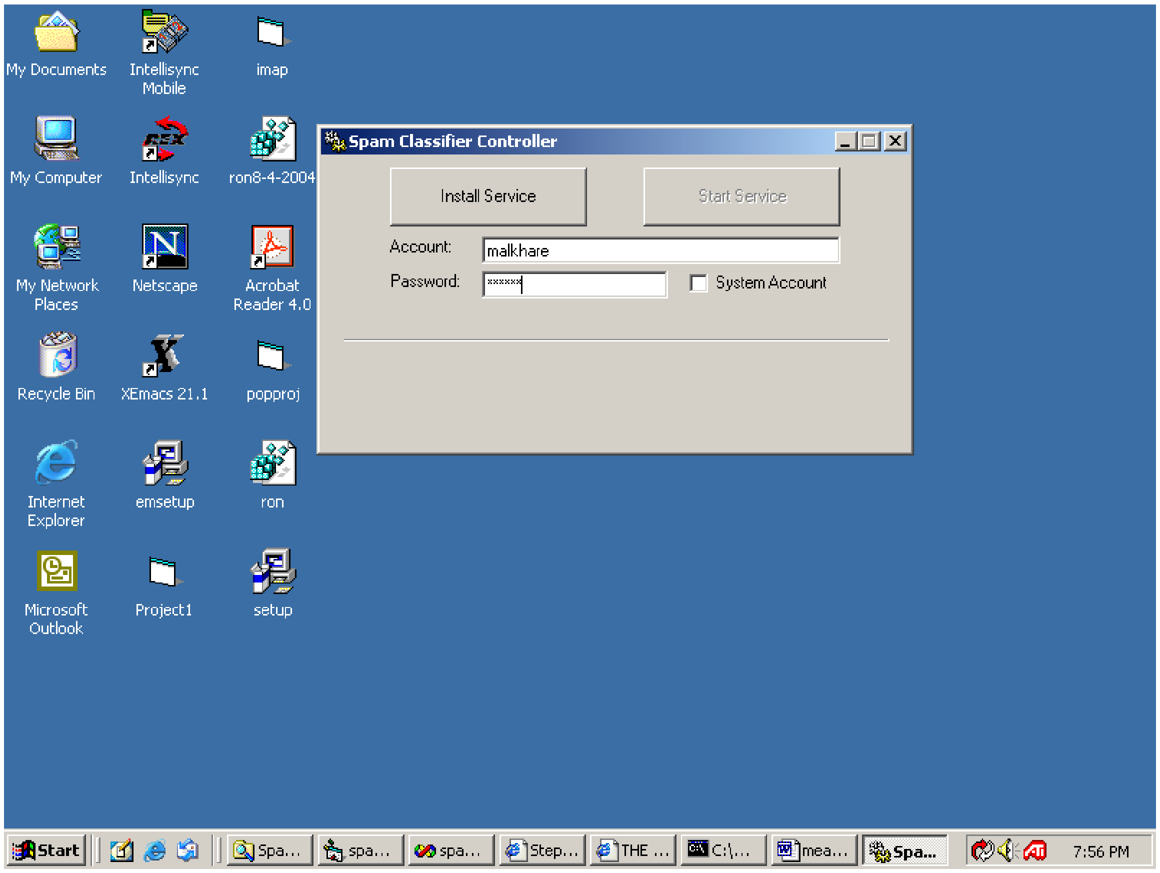
Figure 5: Progress of Training During Feature Extraction

5.4 Classification Service

The classifier runs as a Windows Service which connects to the mail server every 10 minutes, downloads new messages, classifies them and stores messages classified as spam into the spam classification folder.

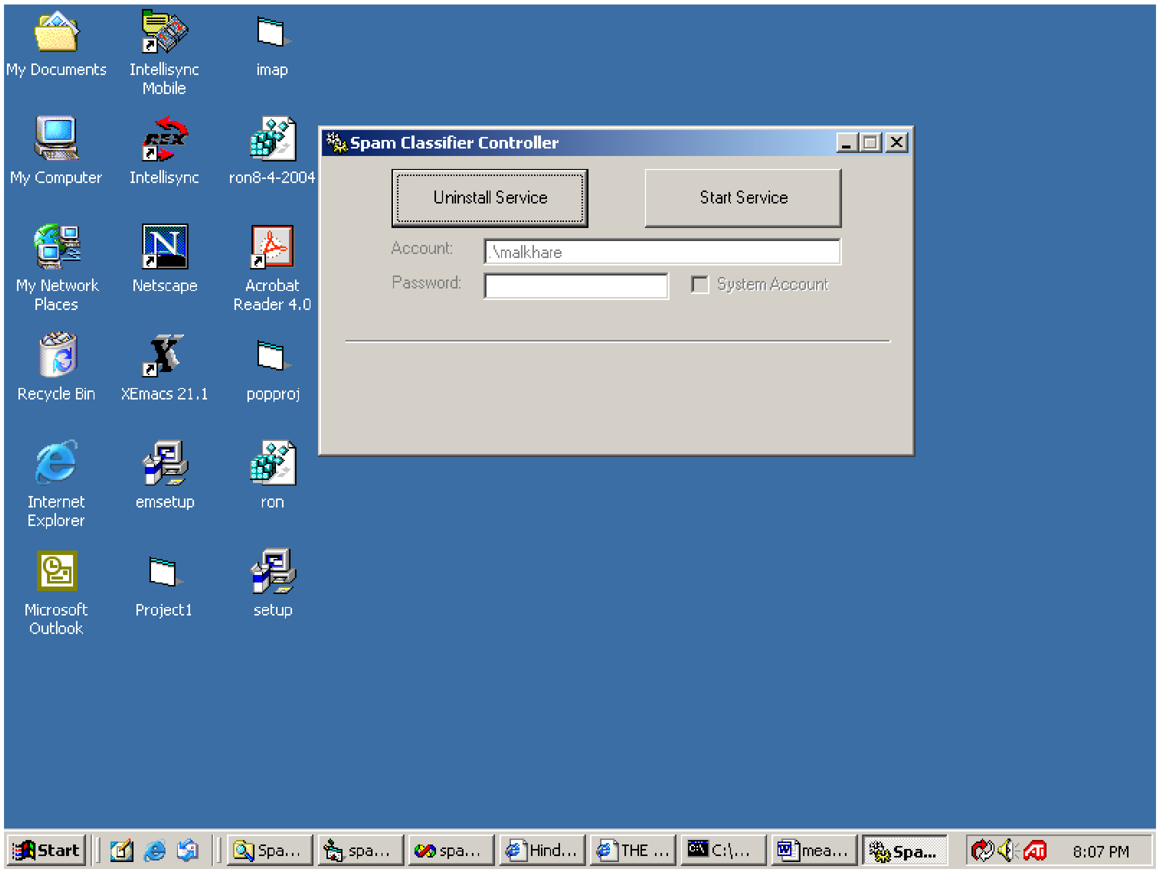
Before usage, the service *sclassify.exe* has to be installed by running the *SControl.exe* program, shown in Figure 5.5. The screen asks for the account name and the password under which the service would run. This account has to have Administrator privileges for the PC.

‘Install Service’ will install and ‘Start Service’ will start the service. Service can also be started through the Services applet in the Control Panel of the PC. Once the service has been started, it can be stopped either through the Services applet or by running the *SControl.exe* program and clicking the ‘Stop Service’ button (Figure 5.6).



Account: malkhare
Password: ******

Figure 6: Installing the Service



Account: .\malkhare

Password:

Figure 7: Uninstalling the Service

During the classification, the address from which the mail has arrived is matched in the addresses stored in the *legitaddr* and the *spamaddr* directories. If there is a match in the *legitaddr* list or in the *spamaddr* list, then the message is classified as legitimate or spam respectively and the classification algorithm is not applied. Classification algorithm is applied only if no match is found in the list of legitimate and spam addresses.

We turned 'off' the address-matching mechanism while testing the classification algorithm. Thus, all measurements in Chapter 4 were done using the content-based classification algorithm.

Chapter 6 – Conclusions and Future Work

We started our research by recognizing the three steps of feature extraction, weight assignment and weight combination used by learning algorithms for spam classification. We created a new learning algorithm which implements a unique mechanism for each of the three steps and improves accuracy for spam classification over the popular naïve-bayes algorithm. Features are created from individual sentences of messages by forming all possible word-pairings from a sentence. Discrete weights are assigned to the features based on the strength of their predictive capabilities for spam/legitimate determination. This predictive strength is based on the frequency of occurrence of the feature in spam/legitimate collections as well as on heuristic rules. During classification, total spam and legitimate evidence in the message is obtained by summing up the weights of extracted features of each class and the message is classified into whichever class accumulates the greater sum.

We defined the concept of 'strong' and 'weak' evidences and listed rules to categorize features into these classes. We empirically determined the best set of weights to assign to features in these two categories. We also defined a thresold parameter to trade-off between the number of false positives and the number of uncaught spam mails. We applied the TUNE (Train until no errors) strategy to auto-set the value of the thresold parameter during training.

The algorithm is implemented on Windows. The implementation uses third-party COM objects to download messages from IMAP mail servers and perform all other operations on messages.

Future improvements to the algorithm include:

1. Use of message headers during the feature extraction step.
2. Use of domain-specific attributes of spam during the weight combination step.
3. Identification of more heuristic rules to separate ‘strong’ and ‘weak’ evidences.

References

[1] William Cohen, 1996. Learning Rules that classify email. In *AAAI Spring Symposium on Machine Learning in Information Access*, *1996*.

[2] Jefferson Provost, 1999. Naïve-Bayes Vs Rule-Learning in Classification of Email.

[3] Patrick Pantel and Dekang Lin, 1998. SpamCop-- A Spam Classification & Organization Program. *Proceedings of AAAI-98 Workshop on Learning for Text Categorization*, *1998*.

[4] Sahami et al, 1998. A Bayesian Approach to Filtering Junk E-Mail. *Proceedings of AAAI*-98 *Workshop on Learning for Text Categorization, 1998.*

[5] Hooman Katirai et al, 1999. Filtering Junk Mail: A Performance comparison between Genetic Programming and naïve-bayes.

[6] Carreras and Marquez, 2000. Boosting trees for Anti-Spam Filtering.

[7] Drucker et al, 1999. Support Vector Machines for spam categorization. *IEEE Transactions on neural networks, 1999.*

[8] Paul Graham, 2002. A Plan for Spam. *MIT Conference on Spam, 2003.*

[9] Bill Yerazunis, 2002. Sparse Binary Polynomial Hashing and CRM114 discriminator. *MIT Conference on Spam,2003.*

[10] Androutsopolos et al. An evaluation of naïve-bayes anti-spam filtering. *Proceedings of the workshop on Machine Learning in the new information age, 11th European conference on machine learning, Barcelona, Spain, pp. 9-17, 2000.*

[11] Apte and Damerau. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems, 12(3):233-251, 1994.*

[12] Cranor and LaMacchia. Spam!. *Communications of ACM, 41(8): 74-83, 1998.*

[13] Duda and Hart. Bayes Decision Theory. *Chapter 2 in Pattern Classification and Scene Analysis, pp. 10-43. John Wiley, 1973.*

[14] Hall R. How to avoid unwanted email. *Communications of ACM, 41(3):88-95, 1998.*

[15] Langley, Wayne and Thompson. An Analysis of Bayesian Classifiers. *Proceedings of 10th National Conference on AI, pp 223-228, San Jose, California, 1992.*

[16] Lewis D. Feature selection and Feature Extraction for Text Categorization. *Proceedings of DARPA workshop on Speech and Natural Language, p 212-217, New York, 1992.*

[17] Lewis D. Training algorithms for linear text classifiers. *Proceedings of 19th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pp 298-306, Konstanz, Germany, 1996.*

[18] William Yerazunis. Beyond the plateau of 99.9% accuracy. *MIT Spam Conference, 2004.*