ifile: An Application of Machine Learning to E-Mail Filtering*

Jason D. M. Rennie
Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA 02139
jrennie@ai.mit.edu

ABSTRACT

The rise of the World Wide Web and the ever-increasing amounts of machine-readable text has caused text classification to become a important aspect of machine learning. One specific application that has the potential to affect almost every user of the Internet is e-mail filtering. The WorldTalk Corporation estimates that over 60 million business people use e-mail [6]. Many more use e-mail purely on a personal basis and the pool of e-mail users is growing daily. And yet, automated techniques for learning to filter e-mail have yet to significantly affect the e-mail market.

Here, I attack problems that plague practical e-mail filtering and suggest solutions that will bring us closer to the acceptance of using automated classification techniques to filter personal e-mail. I also present a filtering system, **ifile**, that is both effective and efficient, and which has been adapted to a popular e-mail client. Results are presented from a number of experiments and show that a system such as **ifile** could become a useful and valuable part of any e-mail client.

1. INTRODUCTION

E-mail clients generally allow users to organize their mail into folders. Netscape Messenger, Pine, Microsoft Outlook, Eudora and EXMH are all examples of this fact. Folders allow the user to organize her mail by meaningful topic. This facilitates more efficient searching when the user is looking for a previously sent or received e-mail. As it becomes easier to store and manipulate documents electronically, the e-mail folder system may become a store for a wide array of documents. Being able to efficiently maintain and search such a collection is an important aspect of any document management system.

Mail folders can also serve the purpose of a prioritization system. Some e-mail is very important and needs to be dealt with when it arrives. Other mail is less important and can be scanned in batches. This is one of the principles that Helfman and Isbell strove to build into Ishmail [7]. Users are able to program simple rules for filtering e-mail into different mailboxes. Ishmail alerts the user when a message is filtered into a high-priority folder or when a large number of messages have accumulated in a lower-priority folder. Ishmail uses hand-constructed filtering rules and would greatly benefit from the complexity reduction that an automated filtering system would allow.

Many mail clients have similar, yet less sophisticated, filtering subsystems that allow the user to filter and prioritize mail. However, constructing and maintaining rules for filtering is a burdensome task. Users sometimes define folders according to message content rather than by pattern matching, whereas mail clients generally require that filters be based on pattern matching. Maintaining a set of filtering rules can also be a difficult task for a user with a large number of folders. Any changes to the folder organization will require significant restructuring of the filtering rules.

These scenarios provide ample ground for automated mail filtering to positively affect the experience of the average e-mail user. Classification techniques exist which can easily automate the filtering task and such classifiers can be learned within the context of most existing mail clients, so the user must endure no additional burdens to make use of an automated mail filter. The main barrier to seeing automated mail filtering becoming commonplace is the resolution of issues regarding the implementation of mail filters and their integration into e-mail clients. Some of the important issues regarding mail filters include speed efficiency, database size and the collection of supervised training data. Time-consuming training or classification can degrade the interface. A large database to store the classification model may limit the user base. Also, a mail filter is only likely to be used if no additional effort is required to reap the benefits. In the following sections, I discuss these issues in detail, describe a mail filter, **ifile**, that I have written for the EXMH mail client and give promising results from experiments that have been run on e-mail collections of 4 different **ifile** users.

2. MAIL FILTERING

The term "mail filtering" is used in different contexts and requires some discussion before we can be clear of the meaning here. Not long after e-mail came into existence did people

*__ifile__ is available at http://www.ai.mit.edu/~jrennie/ifile.

KDD-2000 Text Mining Workshop Boston, MA USA

seek ways to organize and archive their mail. Thus begat mail filters, or, sets of rules that users put together to file incoming e-mail into different mailboxes or folders. I'll call this personal mail filtering because it pertains directly to a single person's organizational preferences. As e-mail use has grown, some regularity has come about the sort of e-mail that appears in users' mail boxes. In particular, unsolicited e-mail, such as "make money fast" schemes, chain letters and porn advertisements, is becoming all too common. Filtering out such unwanted trash is known as junk mail filtering. Drucker et. al. [5] and Sahami et. al. [14] have each examined this problem in detail. Finally, there is also significant interest in filtering e-mail feeds or varying sorts of electronic documents, such as netnews or Reuters articles [10]. For the purposes of this paper, I'll be primarily concerned with personal e-mail filtering. The following few sections discuss issues pertinent to the general problem of mail filtering and specific to the domain of personal mail filtering.

2.1 Classification Efficiency

Text classification is a problem that has been given much attention recently [2] [15]. Joachims [9] gives evidence that a recent development in classification, Support Vector Machines, can readily be applied to text to achieve error rates significantly below those achievable using more traditional techniques such as kNN, C4.5, Naive Bayes and Rocchio. Support Vector Machines and other recent developments in classification, such as Maximum Entropy discrimination [8], provide significant improvements in accuracy, but at the cost of simplicity and time efficiency. As Joachims mentions, SVMs and C4.5 are costly to train and kNN requires significant time to classify a single data point. There also have yet to arise good methods for iteratively training SVMs and decision trees. Efficient training and classification are important concerns since any lags can significantly deter the interface and usability of an e-mail client.

I am hardly the first to bring up the importance of efficiency in the context of mail filtering. Cohen discusses it at some length in [4], noting that RIPPER, his rule-learning system, when adapted to text may not be efficient enough for the sort of on-line classification that mail filtering requires. Of particular concern is be RIPPER's ability to learn rules iteratively; Cohen mentions the need to post-process a set of constructed rules. Such a process is effective when training can be done in large batches, however, personal e-mail filtering is an iterative process where the classifier must be constantly kept up-to-date and training and classification are highly intertwined.

One classification algorithm that provides efficient training, quick classification and easy extensibility to iterative learning is Naive Bayes. The details of Naive Bayes are given in section 2.4. Briefly, adding a document to a trained model requires the recording of word occurrence statistics for that document; no rules need to be learned and no weights need to be optimized. Classifying a document involves calculating a log sum for each class where the size of the sum equals to the number of words in the document to be classified. Each term in the sum is proportional to the frequency with which the corresponding word has occurred in the training data. Training consists of updating word counts; classification consists of a normalized sum of the counts corresponding to the words in the document in question. Hence, training and classification are both simple and efficient and can be integrated into a mail client without degrading the interface. The real-world test for this is **ifile**, a mail filtering system that uses Naive Bayes to do classification. **ifile** has been seamlessly integrated into the EXMH mail client and is currently being used by a number of people without complaints of it damaging the usability of the mail client.

2.2 Supervision

Every user has a unique collection of e-mail and organizes their e-mail in ways that are different from all other users. Granted, there are likely to be similarities between users. Some people keep a special "junk" folder to store unsolicited e-mail; researchers are likely to have folders for talk and conference announcements. One might be tempted to leverage other personal mail filters (e.g. in a collaborative filtering system or to improve prior estimates on word counts), but I find that highly effective filtering can be achieved simply by making use of the information made available through the user interface of the mail client.

Cohen alludes to the supervision that users implicitly give while using their mail client [4]. In interviewing users of Ishmail, he found that many users manually classified e-mail messages that could not be classified through simple pattern-matching rules. This manual classification is a valuable source of training data for a mail filter. Even more valuable is the is the collection of e-mails that are filtered correctly. For, if a filtering system is being used to organize a user's e-mail collection, the user will want to correct any mistakes made by the filtering system. Hence, every e-mail can act as supervised data for the classification model. The label assigned by the filter is assumed to be correct unless the user indicates that it is incorrect by moving the message to another folder. Here, the model must be updated accordingly. The update required for Naive Bayes is simple as it must simply shift word counts from one folder to another.

This learning architecture is implemented in **ifile**. Every newly filtered message is immediately added to the classification model using the label assigned by the filter. When the user moves misclassified mail into the appropriate folder, the filter updates its model. In this way, every piece of filtered mail is a training example for **ifile**. This large number of supervised training examples results in **ifile** quickly building an accurate model of the user's filtering preferences. As a quick validation experiment, I destroyed my **ifile** database and let it rebuild only on newly filtered messages. After filtering 36 messages across 10 different folders, I found that 13 messages where filtered incorrectly. 10 of those were the first for a particular folder and could not have been filtered correctly. Of the remaining, 23 were assigned to the correct folder and 3 were not, an accuracy rate of 88%. Hence, **ifile** is able to quickly learn filtering preferences simply by making full use of the available training data. Using this approach, filtering preferences can be learned both quickly and accurately.

2.3 Features

A classification model acts as a function, f, mapping from features, $\mathcal{F}$, to classes, $\mathcal{C}$. A mail filter is a special classifier where $\mathcal{F}$ is characteristics of e-mail messages and $\mathcal{C}$ is the user's mail folders. As do many others concerned with text classification, I view each e-mail message as a bag of (independent) words. Hence, the classification function, f, maps an unordered set of words to a folder name and takes no account for possible interdependencies between words. One might consider using a different set of features for classification, but bag of words is ample for most tasks and allows for efficient and relatively simple implementation.

Of course, even with a feature set of independent, unordered words, there are many features to consider. An indexing of the 20,000 articles of the "20 Newsgroups"[1] data set using the default settings of McCallum's **rainbow** software [13] compiles over 100,000 unique tokens. More features mean a larger database/classification model and slower training and classification. One way to make filtering more efficient is to reduce the number of features considered for classification.

Yang and Pedersen review a variety of feature selection techniques for text classification and find that good techniques rate highly features that occur frequently [19]. This observation lends itself to simple feature selection scheme that works well within our iterative training framework. Naive Bayes requires the tracking of word frequency statistics. Selecting words by document frequency requires that we also track, for each word, the number of documents filtered while statistics are kept for that word. In other words, we have to know how long each word has been in our database. I call this statistic the word's *age*. *Age* is defined as the number of e-mail messages which have been added to the model since frequency statistics have been kept for the word. Old, infrequent words are to be dropped while young words and old, frequent words should be kept. One way to quantify this is to say that words which occur fewer than $\log_2(age) - 1$ times should be discarded from the model. For example, if "baseball" occurred in the 1st document and occurred 5 or fewer times in the next 63 documents, the word and its corresponding statistics would be eliminated from the model's database. This feature selection cutoff is used in **ifile** and is found to significantly improve efficiency without noticeably affecting classification performance.

2.4 Naive Bayes

"Naive Bayes" is a simple statistical classification model often utilized in the problem of text categorization. McCallum and Nigam give a good discussion of this model [12] and the two event models that are most frequently used. For this paper, I use the multinomial event model, where a document is assumed to be generated by number of rolls of a weighted die, one roll for each word in the document. There is a unique die for each class and each face of each die corresponds to a different word. The likelihood of a face coming up for a particular die is exactly $\theta_{w_t|c_j} = P(w_t|c_j;\theta)$, or the probability of word w_t given class c_j. These values are estimated from the data as follows

$$P(w_t|c_j;\hat{\theta}) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} N_{it} P(c_j|d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{is} P(c_j|d_i)} \tag{1}$$

where $|V|$ is the size of the vocabulary, N_{it} is the number of times word t occurs in documents of class i and $\hat{\theta}$ is used to denote parameters learned from the empirical data. The probability of a document having been generated from this model is

$$P(d_i|\theta) = \sum_{j=1}^{|\mathcal{C}|} P(c_j|\theta) P(d_i|c_j;\theta) \tag{2}$$

$$P(d_i|c_j;\theta) = P(|d_i|)|d_i|! \prod_{t=1}^{|V|} \frac{P(w_t|c_j;\theta)}{N_{it}!}. \tag{3}$$

Using Bayes rule, our maximum likelihood classification rule is

$$\arg\max_{c_j} P(c_j|d_i;\hat{\theta}) \text{ where } P(c_j|d_i;\hat{\theta}) = \frac{P(c_j|\hat{\theta}) P(d_i|c_j;\hat{\theta})}{P(d_i|\hat{\theta})}. \tag{4}$$

Since we are only concerned with relative values, $P(c_j|d_i;\hat{\theta})$ is calculated as a sum of logs in order to avoid round off error. Since log is a strictly increasing function, classification is not affected by this transformation. Note also that we can disregard the $P(d_i|\hat{\theta})$, $P(|d_i|)$ and $|d_i|!$ terms for the purposes of classification since they are constant across different values of c_j.

3. IFILE

ifile began in the summer of 1996 as a fun application of the Naive Bayes classification algorithm of which I had recently learned. Over the years it has generated a significant user base and with the help of a number of individuals[2] has developed into a mature package. At one point in time, over 40 people were signed up to receive announcements regarding **ifile** releases. For many people it supplements or replaces the functionality provided by other manually-specified mail filters. **ifile** exists as a core C executable, two Perl wrapper scripts and Tcl code to interface with the EXMH mail client. The core executable could be adapted to allow automated filtering using any mail client. The executable itself stores and maintains the classification model and produces class labels for new e-mail. The two wrapper scripts are command-line interfaces for filtering incoming mail and updating the model when the user moves mail between folders. The wrappers are specific to the MH mail system on top of which EXMH runs. Finally, the Tcl code provides hooks into the user interface so that **ifile** can be used with almost no user effort.

ifile takes into account all of the issues discussed in section 2. It uses an efficient Naive Bayes implementation which can both build a classification model and filter a new e-mail message very quickly. For example, it builds a model of my 7000+ e-mail messages (stored on a local disk) in 27 seconds,

[1]The 20 Newsgroups data set can be obtained from http://www.ai.mit.edu/people/~jrennie/20_newsgroups.

[2]Those contributing code to **ifile** include Andrew McCallum, Diego Zamboni, Harry G. McGavran Jr., Carl Staelin, Valdis Kletnieks, Dave Robinson and Chris Browne.

User	Number of E-mail Messages	Number of Folders
1	2715	27
2	373	16
3	655	13
Rennie	4447	33

Table 1: Information about the e-mail corpora on which classification experiments were conducted.

an average of 259 messages per second[3]. Creating a targzip ball of the same 7000+ messages requires 17 seconds. MailCat/SwiftFile, a filtering system constructed by Segal and Kephart, requires four minutes to compile a database of 1000 messages, an average of 4 messages per second[4] [16].

ifile selects features for classification according to the $\log_2(age) - 1$ formula mentioned in section 2.3. The classification model built on my e-mail corpus of 7000+ messages across 49 folders requires only 447,090 bytes of disk space (a very reasonable size in an age where 10 gig hard drives are common). Loading the model and making a classification decision on a single 2500 byte e-mail message takes a approximately 0.2 seconds. Performing the updates necessary to move a message takes only slightly longer, approximately 0.4 seconds. Here the database needs to be read, modified and written back to disk. These are similar to the 0.3 second classification times reported by Segal and Kephart [16]. Sub-0.1 second times could easily be achieved with **ifile** via a client-server architecture where the database is stored in memory rather than on disk. These numbers also compare favorably compared to the numbers that Card [3] gives for performing primitive UI operations, such as pressing a particular key (0.5 seconds) or moving the mouse to a target location on the screen (1.1 seconds). Since any filtering action will be complete by the time the user finishes her next primitive UI operation, the user will notice little, if any, UI degradation while using **ifile**. Yet, she will gain the benefits of an automated filtering system. Hence, **ifile** can be considered "fast enough" because it does not limit the speed at which a user can perform a sequence of actions.

The efficiency of **ifile**, combined with its classification performance (described in the next section) have resulted in many users finding **ifile** to be a usable and welcome addition to their attempts to prioritize and organize their e-mail collections.

4. EXPERIMENTS

As with any supervised classification problem, training examples are an essential part of the construction of a model. Mail filters are no different. An interesting aspect of mail filtering is that labeled data is ubiquitous, yet difficult to obtain for experimental purposes due to privacy concerns. As has recently been discussed on the ddlbeta mailing list[5], one can construct data sets that mimic the structure and properties of personal e-mail (such as collections of newsgroups or mailing list messages), but no reasonable alternative is guaranteed to be a good benchmark for testing.

[3]Performance tests were run on a PIII 500Mhz PC with a Quantum Atlas IV SCSI 9.1 gig hard disk.
[4]Performance test were run on a PII 400 Mhz PC.
[5]Contact David Lewis <lewis@research.att.com> for information concerning ddlbeta.

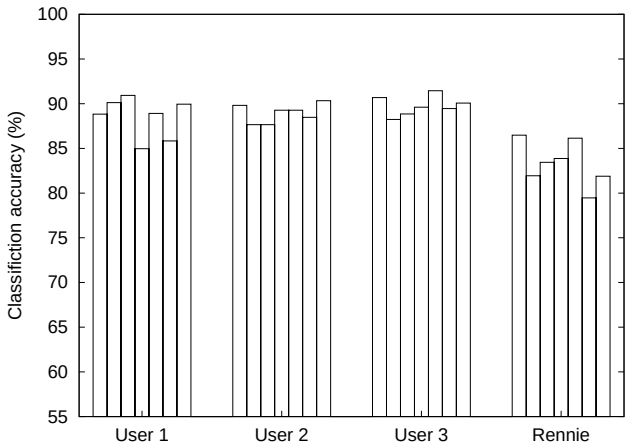

Figure 1: Classification accuracies for seven different experiments on four different users. Experiments are ordered from left to right within each user block. Note that no settings produce the best results across all users.

As far as I know, there are no freely available data sets for mail filtering. E-mail is considered to be private; few people are willing to let strangers view and manipulate their e-mail corpus. Hence, it can be difficult to obtain realistic experimental results for the mail filtering task. One benefit of **ifile** being a real, usable, freely available system is that every **ifile** user has the framework necessary for performing mail filtering experiments. During the course of **ifile**'s development. I asked for volunteers who would be willing to have such experiments performed on their mail collection; four users (including the author) volunteered. The results that I received on different mail collections are quite interesting.

Experiments on each e-mail corpus were performed in a leave-one-out fashion. In other words, each document in a user's corpus was labeled according to a model built on the rest of the e-mail messages. It would be as though each e-mail were a new message to be filtered in the context of the rest. The assigned labels were compared against the true labels to derive an accuracy score. Individual experiments used slightly different settings for tokenizing e-mail messages and selecting features for classification. The settings for the experiments involved using one of three different tokenizers, using a stop list, using stemming, using feature selection as described in section 2.3 and including all e-mail headers (as opposed to only including `From`, `To` and `Subject`). The intention of running a variety of experiments was to determine whether any settings work well across different users or whether it would be useful to tune the settings to each individual user. All of the settings correspond to easily set options of the **ifile** core executable.

Table 1 gives information regarding the e-mail corpora on which experiments were performed. Note the varying sizes of the e-mail collections. The fourth user (Rennie) is the author of this paper. Figure 1 shows the results of the classification experiments. Above each user identifier is a number of bars, one for each experiment. The bars are ordered within each user block, so the far left bar above each user corresponds to the same experimental settings. Classification accuracy is generally quite high, especially considering the simplicity of the underlying classification model. This indicates that a mail filter such as **ifile** may well satisfy the average user's need for filtering accuracy.

From these experiments, it is not clear that any parameter settings provide the optimal classification performance across a wide array of users. For example, the third bar from the left, experiment #3, yields the best results for user #1. This experiment used a lexer that tokenizes strings separated by whitespace; the default lexer tokenizes strings of alphabetic characters. While this lexer gives better results for user #1, it produces relatively mediocre results for the other three users. The far left bar (experiment #1) produces the best average results across the four users (89% accuracy) and corresponds to the default settings of **ifile**. The settings of experiment #1 involve using an alphabetic lexer, using a stop list to remove common words, using feature selection and removing headers other than `From`, `To` and `Subject`; no stemmer is used. Accuracies for experiment #1 lie between 86% and 91%, a range that is likely to be acceptable for a large number of e-mail users. These numbers compare very well against the 52% to 76% "first button" numbers reported by Segal and Kephart in their SwiftFile system [17]. Segal and Kephart consider SwiftFile's performance to be "unacceptable" for automated classification, whereas **ifile**'s classification accuracy is acceptable for automated filtering (proven so by the number of people who use **ifile** for exactly that task).

Since no experimental settings provide the best results across all users, it may be worthwhile to optimize classification settings on a per user basis. This would allow the features used by the classifier to be more closely tuned to how the user makes filtering decisions. In order to reap great benefit from such work, one would need to consider experimental settings other than those used in these experiments; here the performance gain achieved by choosing the optimal settings for each user was minor.

One might be concerned that these experiments do not accurately portray the filtering performance of of a mail filter during normal usage. In real usage, the classification model only has information about e-mail messages that were received before the e-mail in question. In the experiments described in this paper, the model used as training data all e-mails besides the one in question. Hence, it had access to future messages. However, this minor discrepancy is unlikely to affect accuracy scores compared to an entirely realistic experiment. Over the past seven months as I have been using **ifile**, it has been tracking its own filtering performance and shows a close match to the results reported in figure 1. Of the 4772 messages that it has filtered for me, it predicted the correct folder 85.4% of the time. This is nearly as good as the 86.5% accuracy score reported by the leave-one-out experiments. The slight drop may be due to the fact that the model was not able to see future messages and based its classification decisions solely on past messages; it may also be due to the fact that I now have 49 e-mail folders whereas I had only 33 folders when the original experiments were performed. Nevertheless, extrapolating this result to the other data, it should be clear that the accuracies reported in figure 1 are representative of those that one would find with real usage of **ifile**.

5. DISCUSSION

While personal mail filtering generally implies assigning messages a single label, an interesting way to use a text classifier within the context of a mail client is as a filter aid. Instead of using a text classifier to automatically filter messages, one can use a text classifier to suggest folders to which a new message is likely to belong. Many people use filing features for archival purposes, filing messages to folders after replying or having utilized the contained information. However, a person with a large collection of e-mail folders may find the filing process cumbersome because she has to navigate through an interface listing 30 or more folder names. Given that a text classifier can select the appropriate folder more than 85% of the time, it will almost always be able to give the appropriate folder within its top three guesses. Hence, most of the time, the user would only have to select between three options rather than the overwhelming 30+ options if no filtering is performed. Segal and Kephart describe this approach and report on experiments that show how this can make e-mail filing a less painful task [17].

This paper has shown that mail filtering is at a stage that it can be effectively integrated into modern mail clients. However, there is still room for improvement. For example, **ifile** takes no special care to track threads even though messages that are part of the same thread are highly likely to be filed into the same folder. Lewis and Knowles study methods for recognizing threads and give near-perfect retrieval results by searching for matching quoted/unquoted text [11]. Such techniques could be added to **ifile** to improve its ability to recognize and correctly classify e-mail messages that are part of the same thread. Another area of possible research involves compensating for organizational changes and semantic shifts over time. Users may change their folder organization and/or change their individual folder filing patterns over time. The classification model could benefit from detecting and compensating for these changes. Mail filtering may be able to incorporate the already large body of work that exists on the problems of topic detection and tracking (TDT) [18] [1].

6. CONCLUSIONS

This paper discusses practical concerns surrounding the application of text classification to the problem of mail filtering. Much research has recently been vested in the problem of text categorization. There is even a significant body of work on the application to mail filtering. And yet, few use the automated filtering techniques developed by this research. Here, I have discussed some of the barriers separating research from reality and suggested reasonable solutions to those problems. Furthermore, I have presented a freely available, functioning e-mail filtering system, **ifile**, that has already been in use for a number of years and have reported

on experiments which suggest that such a system performs well enough to be accepted as a beneficial tool by a large number of e-mail users.

7. REFERENCES

[1] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-98)*, 1998.

[2] A. Berger. Error-correcting output coding for text classification. In *Proceedings of IJCAI-99 Workshop on Machine Learning for Information Filtering*, 1999.

[3] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[4] W. Cohen. Learning rules that classify e-mail. In *AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[5] H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5), 1999.

[6] D. Harris and H. Clark. Worldtalk releases first Internet e-mail corproate usage report; concludes e-mail abuse at epidemic levels. http://www.worldtalk.com/Corporate%20Information/press%20releases/iecur.htm, 1999.

[7] J. Helfman and C. Isbell. Ishmail: Immediate identification of important information. http://www.research.att.com/∼jon/ishmail, 1995.

[8] T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. Technical Report AITR-1668, MIT, 1999.

[9] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML-98)*, 1998.

[10] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of Twelfth International Conference on Machine Learning (ICML-95)*, 1995.

[11] D. D. Lewis and K. A. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 33(2):209–217, 1997.

[12] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[13] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/∼mccallum/bow, 1996.

[14] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *Proceedings of Workshop on Learning for Text Categorization*, 1998.

[15] S. Scott and S. Matwin. Feature engineering for text classification. In *Proceedings of Sixteenth International Conference on Machine Learning (ICML-99)*, 1999.

[16] R. B. Segal and J. O. Kephart. Mailcat: An intelligent assistant for organizing e-mail. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[17] R. B. Segal and J. O. Kephart. Incremental learning in swiftfile. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, 2000.

[18] Y. Yang, T. Ault, T. Pierce, and C. W. Lattimer. Improving text categorization methods for event tracking. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2000.

[19] Y. Yang and J. Pedersen. Feature selection in statistical learning of text categorization. In *Fourteenth International Conference on Machine Learning (ICML-97)*, 1997.