

MailCat: An Intelligent Assistant for Organizing E-Mail

Richard B. Segal and Jeffrey O. Kephart

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
rsegal@watson.ibm.com, kephart@watson.ibm.com

Abstract

While most mail reader applications allow users to file messages into folders, in practice this task tends to be tedious. For each message, the user must first decide which folder is most appropriate. Then, the user must inform the mail reader of that choice by selecting the appropriate icon or menu item from among what is typically a set of several dozen choices. The combined effort of choosing a folder and conveying that choice to the application often discourages users from filing their mail, resulting in unmanageable inboxes that contain hundreds or even thousands of unfiled messages. MailCat encourages users to file their mail by simplifying the task. Using an adaptive classifier, it predicts the three folders that are most likely to be appropriate for a given message, and provides shortcut buttons that permit the user to file it into a predicted folder effortlessly. For typical users, MailCat's predictions are accurate over 80% to 90% of the time, resulting in a substantial reduction in the time and cognitive burden required to file messages.

1 Introduction

Active users of electronic mail may receive dozens or even hundreds of messages every day. To facilitate retrieval of messages weeks, months or years after their original receipt, most mail reader applications allow users to organize their messages into user-defined folders. Typically, a user may maintain somewhere between 10 and 100 separate folders; some applications permit these folders to be arranged hierarchically. Users who are reasonably consistent in their filing behavior are likely to select the right folder when attempting to retrieve a particular message or set of messages. Effective use of folders can complement or even supplant the alternative approach of performing a keyword search over one's entire mail archive.

Appears in the Proceedings of the Third International Conference on Autonomous Agents. Copyright © 1999 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

Most applications provide a simple interface for filing messages into folders. For instance, Lotus Notes provides a *MoveToFolder* button that pops up a dialog box that allows the user to select the destination folder. It also displays a list of folders next to the message and allows the user to move a message into a folder using drag-and-drop. Other applications use similar methods to facilitate message filing. Regardless of the detailed arrangement of the user interface, in practice the cognitive effort required to decide upon an appropriate folder and locate the icon or menu item representing it is substantial enough to discourage users from filing at all, particularly when faced with the prospect of repeating the exercise dozens of times per day.

MailCat is an intelligent personal assistant that reduces the cognitive burden and the time required for organizing electronic mail into folders. Using a text classifier that adapts dynamically to a user's observed mail-filing habits, it predicts the three folders that are most likely to be appropriate for a given message and provides shortcut buttons that facilitate filing that message into one of its predicted folders. When one of the folders predicted by MailCat is correct, the user's task is greatly simplified. Rather than having to derive a best choice from a large set of folders, the user can merely confirm one of MailCat's suggested choices, and moreover they can express that confirmation as a single mouse click. Given that MailCat's prediction accuracy is roughly 80% to 90% even for users with as many as 60 folders, MailCat offers a significant qualitative enhancement that is likely to encourage users to file their mail — enabling them to enjoy the advantages of an organized mail archive.

MailCat offers these advantages without demanding anything in return. Users have nothing new to learn or do. They only need to use MailCat's self-explanatory buttons. When a user first installs MailCat, it analyzes their existing folders, and from this constructs a text classifier that is tuned to that user's mail-filing behavior. Thereafter, the user may use the shortcut buttons, or may opt to use the standard message-filing interface if none of the suggested folders are appropriate. Regardless of whether its suggestions are accepted, MailCat's classifier updates itself promptly as each new message is filed.

This paper is organized as follows. In the next section, we give an overview of MailCat's design philosophy and structure. We also illustrate its integration with Lotus Notes, a commercial application that handles mail and various groupware functions. In section 3, we describe a modified TF-IDF text classifier that supports low-cost incremental updates; this is the key to MailCat's adaptivity. In section 4, we dis-

cuss some implementation issues that arise when integrating MailCat with Lotus Notes, some of which are likely to extend to other mail applications as well. An evaluation study reported in section 6 establishes that MailCat performs well in practice. We conclude with a comparison to related work in section 6 and some final remarks in section 7.

2 Overview of MailCat

A fundamental MailCat design principle is *simplicity*. MailCat must provide a substantial benefit to users without demanding anything extra from them. Users should not be required to learn or do anything new in order to use MailCat. Furthermore, any errors made by MailCat should have no negative impact on the user, and in particular the user should be able to ignore MailCat and use the application just as if MailCat were not present. As shall be seen, this basic principle has directly influenced several of the design choices that define MailCat.

The MailCat concept is sufficiently simple and general to apply to any of today's major mail applications. Our first (and so far only) implementation of MailCat is as an add-on to the Lotus Notes mail client. An important factor in this choice was the extensive C++ API provided by Notes to facilitate the development of add-ons such as MailCat.

As do several modern-day mail reader applications, Notes allows users to organize messages into a hierarchy of folders. Notes provides a *MoveToFolder* button for moving messages from the Inbox to the desired folder. As shown in Figure 1, pressing the *MoveToFolder* button brings up a dialog box that enables the user to select the destination folder. Typically, the user must scroll down to select the desired folder, which can be tedious if there are many folders. Notes also permits filing via a drag-and-drop paradigm. However, regardless of how the choices are displayed, there are typically enough such choices to make the decision process and the search for the appropriate menu item or icon burdensome.

Figure 2 shows how MailCat simplifies the task of organizing messages. MailCat places three *MoveToFolder* shortcut buttons above each message. The shortcut buttons allow the user to quickly move the message into one of the three folders that MailCat predicts to be the message's most likely destinations. The buttons are ordered from left to right. The leftmost button represents MailCat's best guess as to where to file the message, the middle button represents its second-best guess, and the third button represents its third-best guess. When one of the three buttons is clicked, the message is immediately moved to the indicated folder.

MailCat uses a text classifier to predict the most likely destination folders for each message in the inbox. The text classifier must be trained on previously-filed messages to learn how messages are typically organized. Maes [Maes, 1994] has noted that acquiring the training data for learning can take a substantial amount of time and effort, and has pointed out that this is a significant barrier against acceptance of mail agent technology. Maes has proposed collaborative learning as a solution to this problem, and has implemented those ideas in an e-mail agent called Maxims. Maxims seeks out users with similar mail-filing habits and learns directly from their mail-filing agents. While the Maes' experiments suggest that this can work among users with similar mail-filing habits, it seems unlikely that any two individuals would use filing schemes that were sufficiently

similar to take advantage of collaborative learning.

MailCat employs an alternative solution to the start-up problem: it learns from messages previously filed by the user. Most e-mail users already have a large database of previously-filed messages which can be used to bootstrap the text classifier — the messages currently stored in their folders. Often, this database provides ample training data. When MailCat is first installed, it treats the user's database of previously-filed messages as a corpus of labeled documents and uses standard techniques to develop a TF-IDF text classifier. For a corpus of one thousand filed messages, the initial training takes about four minutes on a Pentium II 400 MHz processor. After the training is complete, MailCat is ready to make useful predictions about the desired placement of mail that is already in the inbox, or about mail that is just arriving.

MailCat's use of previously-filed messages as a training corpus is motivated by our emphasis on simplicity. The user's interaction with MailCat is greatly simplified by eliminating any need for explicit instruction, and by providing good classification accuracy from the start. Our drive for simplicity has also influenced our choice of the learning task. Many systems, Maxims included, have focused on the problem of predicting the *action* the user will take when a message is received. The actions a user may take include sending automatic replies and deleting a message, in addition to filing a message in a folder. The task of learning actions is more difficult in at least two senses. First, the learning task itself is intrinsically more difficult because the set of possible actions includes more than just movement of a message from the inbox to another folder. Second, the data required to learn this task are difficult to obtain. Mail readers usually do not store histories of replies and message deletions in a manner that can be used for training. Since these histories are not available, Maxims and similar systems cannot be effective until they have been running for some time and have seen sufficient training data. By focusing on a simpler problem, MailCat is able to use a simpler solution that demands little of the user or of the mail application in which it is embedded.

Training of the classifier from pre-existing data is only half the battle. Users are constantly creating, deleting and reorganizing their folders. Even if the folders remain the same, the nature of the messages within a folder may well drift over time. For instance, the nature of MailCat-related messages placed in the authors' folders has changed noticeably during the course of the last year. Early messages were about the design and implementation of the classifier. Later messages discussed a patent application based on the MailCat concept. More recent messages have discussed recent experiments and this paper. Had the text classifier learned a model of our MailCat folders when we first started using MailCat, that model would probably perform badly today.

MailCat adapts to changing conditions by using a classifier that supports incremental learning. Once the classifier has been trained, the classifier's model can be updated by presenting to the classifier the messages that have been added or deleted from each folder. The cost of the update is linear in the length of the message. After updating, the classifier's predictions are indistinguishable from what they would have been had the classifier been trained from scratch on the entire mail database.

The low cost of incremental learning allows MailCat to update the classifier as the user interacts with their mail client. If the user creates a new folder and adds a few mes-

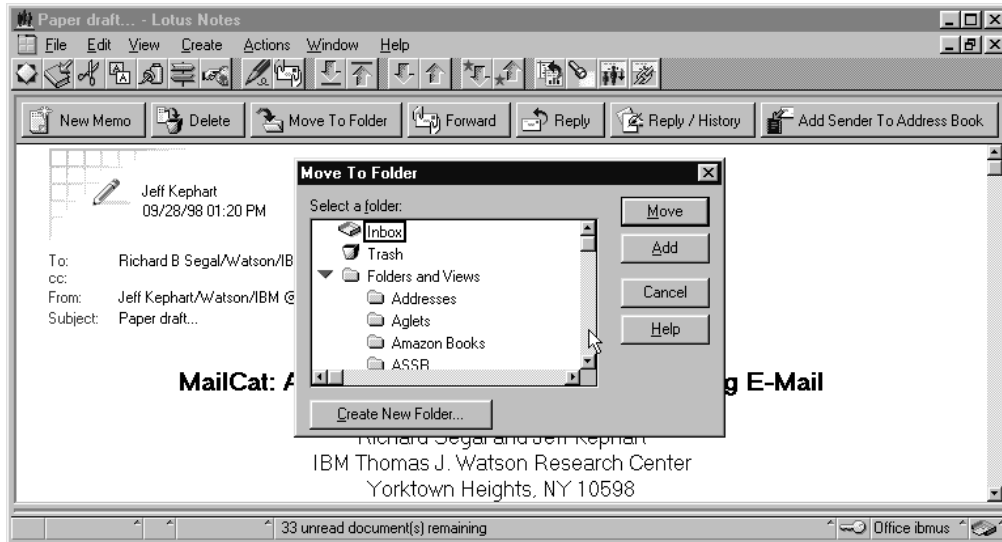


Figure 1: Dialog box for filing messages in Lotus Notes. Typical users must select an appropriate folder from among dozens if not hundreds of different choices.

sages, MailCat instantly learns about the folder and can start predicting which messages are likely to be placed in the new folder. The use of incremental learning allows MailCat to eliminate the delay that one would expect of mail-filing agents that used batch learning algorithms that required periodic (perhaps overnight) re-training.

While MailCat’s use of three buttons rather than one may seem trivial, it is in fact quite significant because it substantially improves MailCat’s usefulness. As will be shown later in the section on experimental results, the use of three buttons cuts MailCat’s failure rate in half without introducing any adverse side-effects. MailCat’s use of three buttons is a direct result of our decision to provide an assistant that *facilitates* rather than *automates* message filing — a choice that itself arose from our emphasis on avoiding negative impacts on users. Since a message can be automatically filed in only one folder, automatic categorization systems have to rely solely on the accuracy of their first prediction.

3 Text Classifier

MailCat’s text classifier is a modified version of AIM [Barrett and Selker, 1995, Kirsche and Barrett, 1996], a TF-IDF style classifier [Salton and McGill, 1983] developed at IBM’s Almaden research laboratory and used extensively in WBI [Barrett *et al.*, 1997].

AIM represents each training or test message \mathcal{M} as a word-frequency vector $F(\mathcal{M})$, in which each component $F(\mathcal{M}, w)$ represents the total number of times the word w appears in \mathcal{M} .

AIM represents each folder \mathcal{F} using a *weighted* word-frequency vector $W(\mathcal{F}, w)$. Several steps are involved in computing $W(\mathcal{F}, w)$. First, the folder \mathcal{F} ’s centroid vector $F(\mathcal{F}, w)$ is computed by summing the word-frequency vectors for each message contained in \mathcal{F} :

$$F(\mathcal{F}, w) = \sum_{\mathcal{M} \in \mathcal{F}} F(\mathcal{M}, w). \quad (1)$$

This folder centroid vector is then converted to a weighted word-frequency vector using the TF-IDF principle: the weight assigned to a word is proportional to its frequency in the folder and inversely proportional to its frequency in other folders. We define $FF(\mathcal{F}, w)$ to be the fractional frequency of word w among messages contained within folder \mathcal{F} , or the number of times word w occurs in folder \mathcal{F} divided by the total number of words in \mathcal{F} :

$$FF(\mathcal{F}, w) = \frac{F(\mathcal{F}, w)}{\sum_{w' \in \mathcal{F}} F(\mathcal{F}, w')}. \quad (2)$$

The definition of term frequency $TF(\mathcal{F}, w)$ used by AIM is

$$TF(\mathcal{F}, w) = FF(\mathcal{F}, w) / FF(\mathcal{A}, w), \quad (3)$$

where \mathcal{A} represents the set of all messages (the entire database of organized mail). We define the document frequency $DF(w)$ to be the fraction of folders in which the word w appears at least once. The definition of inverse document frequency $IDF(w)$ used by AIM is

$$IDF(w) = \frac{1}{DF(w)^2}. \quad (4)$$

Finally, AIM combines these two formulas to define the weight for word w in folder \mathcal{F} :

$$W(\mathcal{F}, w) = TF(\mathcal{F}, w) \times IDF(w). \quad (5)$$

The weight vectors for each folder are used to classify each new message. When a message \mathcal{M} arrives to be classified, it is first converted into a word-frequency vector $F(\mathcal{M})$. Then, AIM computes the similarity between \mathcal{M} and the weighted word-frequency vectors for each folder, $W(\mathcal{F})$. AIM computes the similarity between the message vector and the weighted folder vectors using a variation of cosine

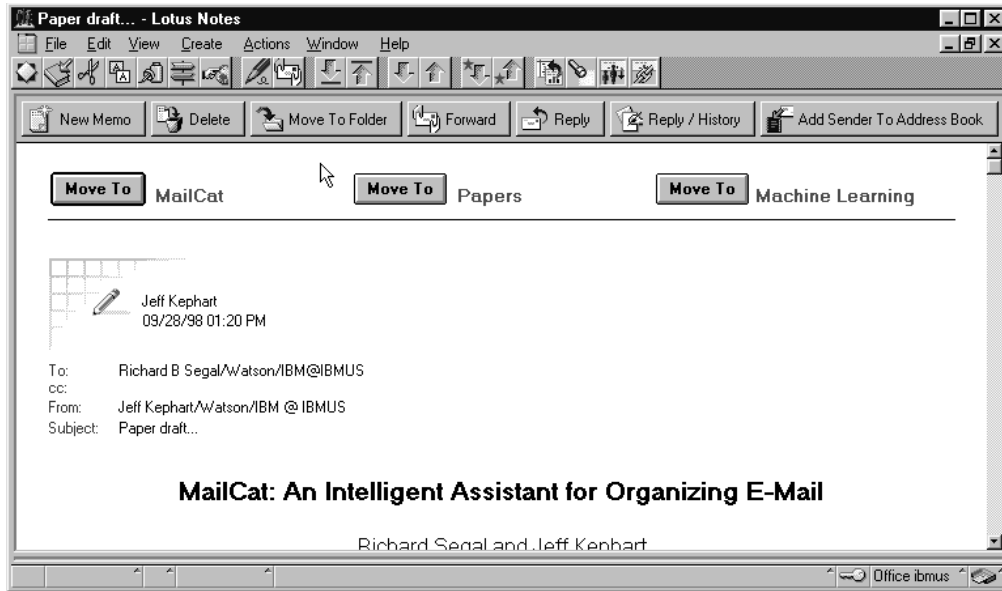


Figure 2: Shortcut buttons provided by MailCat. MailCat creates shortcut buttons for the three folders that it predicts to be the most likely destinations. When MailCat’s predictions are correct, the buttons enable the user to file messages into the indicated folder with a single button click.

distance proposed by Salton [1983] called SIM_4 :

$$SIM_4(\mathcal{M}, \mathcal{F}) = \frac{\sum_{w \in \mathcal{M}} F(\mathcal{M}, w) W(\mathcal{F}, w)}{\min(\sum_{w \in \mathcal{M}} F(\mathcal{M}, w), \sum_{w \in \mathcal{M}} W(\mathcal{F}, w))}. \quad (6)$$

Here the sums are taken only over words that are contained only within \mathcal{M} . Finally, AIM takes the three folders for which the similarity is greatest as its predictions.

AIM implements the above algorithm as follows. AIM maintains a database of the centroids $F(\mathcal{F})$ for each folder \mathcal{F} . Whenever a new message \mathcal{M} is added to a folder \mathcal{F} , AIM updates the centroid by adding to it the word-frequency vector for that message:

$$F(\mathcal{F}, w) \leftarrow F(\mathcal{F}, w) + F(\mathcal{M}, w). \quad (7)$$

Similarly, when a message is removed from a folder, the word-frequency vector for that message is subtracted from the folder’s centroid. The centroids of each folder are stored in an inverted index for fast access.

The original AIM package was not incremental. Before classification could occur, AIM computed the weighted word-frequency vectors for each folder and stored them in a second inverted index. Once this was done, AIM could quickly classify messages by retrieving the needed weights from this second index. Use of a second index is time-efficient, provided that any changes to the classifier occur very infrequently. However, it is grossly inefficient in the sort of dynamic environment for which MailCat is designed. The computation of the weighted word-frequency vectors $W(\mathcal{F}, w)$ takes an amount of time proportional to the size of the index, which can grow to ten megabytes or more. Under conditions of frequent updating, practically none of these

computed entries will ever be used — they will be overwritten by new values before they are needed. This is extremely wasteful, and much too slow to support incrementality.

We modified AIM to support incrementality by eliminating the weight pre-computation. Instead, when a message \mathcal{M} is to be classified, the classifier computes on the fly only the weights for words occurring in \mathcal{M} , since these are all that are required by the SIM_4 metric. The on-the-fly weight computation is performed as follows. First, for each folder \mathcal{F} , the components w of the centroids $F(\mathcal{F}, w)$ corresponding to words which appear at least once in \mathcal{M} are retrieved from the inverted index of centroids. For efficiency, the inverted index also maintains the sums $\sum_{w' \in \mathcal{F}} F(\mathcal{F}, w')$, and these too are retrieved. The fractional frequencies $FF(\mathcal{F}, w)$ are computed from these quantities using Eq. 2, and then from these the term frequencies $TF(\mathcal{F}, w)$ and the quantities $DF(w)$ are computed. Finally, $IDF(w)$ is computed, and then Eq. 5 is used to compute the weight vectors $W(\mathcal{F}, w)$ that are needed for the computation of the similarity between \mathcal{M} and each folder \mathcal{F} . In practice, we find that the extra computational overhead of computing weights on the fly is only about 15%.

With this extension, AIM is fully incremental. The time required to add a document to its index, to remove a document from its index, and to classify a new document is linear in the length of the document being added, removed, or classified.

It should be pointed out that MailCat does not depend in any essential way on AIM and its specifics. It does not even require that the classifier be based upon TF-IDF. In order to support MailCat, the only requirements are that the classifier be efficient in time and space, reasonably accurate in its classification, supportive of incremental learning, and capable of producing a ranking of several possible categories, rather than just producing a single answer.

4 Lotus Notes Implementation

As has been mentioned, we implemented MailCat as an extension to Lotus Notes. The high degree of customizability and the existence of several tools that facilitate the creation of Notes extensions make Notes an attractive partner for MailCat. In Notes, all data, including electronic mail, is stored in databases. Each database has associated with it a design template that determines how the database is presented to the user and what actions the user can perform. The design template is easily modified to add new features. In addition, there is a C++ API for directly accessing and modifying Notes databases. MailCat was implemented using a combination of these two techniques.

MailCat's user interface was written by modifying the standard Notes design template to include three extra buttons for moving messages into files. However, integration with the classifier required being able to call external procedures. We therefore implemented MailCat by creating two daemons that use the C++ API to directly read and write the Notes mail database.

The first daemon is responsible for classifying new messages. Ideally, MailCat would classify messages just before they were displayed to the user. This would ensure that the buttons presented with each message were generated using MailCat's latest knowledge of the user's mail-filing habits. However, there are two difficulties with this approach. First, on a 166 MHz Pentium II machine, classification can take as long as 0.3 seconds, which may be just enough of a delay to annoy some users. Therefore, MailCat uses a daemon to classify new messages. The new-mail daemon checks for new mail every sixty seconds. When the daemon finds a new message, the daemon classifies it using MailCat's current knowledge and adds the appropriate shortcut buttons to the message. The shortcut buttons are then available to the user when the message is opened.

The use of a daemon to classify new messages can reduce the accuracy of the classifier. The daemon only classifies messages when they first arrive. However, MailCat is continuously updating its classifier as messages are filed. If there is a substantial delay between when a message is classified and when the message is filed, the shortcut buttons added to the message may differ from the classifier's latest predictions and therefore may be less accurate. To partially alleviate this problem, we have added to the Notes interface a button to re-classify all messages in the inbox. This enables users to manually refresh the computed shortcut buttons to ensure that they remain as accurate as possible.

The second daemon is responsible for incremental learning. It monitors the mail database for changes. Whenever it detects that a message has been added or removed from a folder, it updates the text classifier accordingly. Since in the Notes C++ API it is more expensive to check for changes to folders than to check for new messages, this daemon is run less frequently — at ten minute intervals. A delay of up to ten minutes is unlikely to be a problem because mail-filing habits tend not to change that quickly.

5 Experiments

We performed two sets of experiments on MailCat. The first experiment was designed to assess the performance of MailCat's classifier, while the second evaluated MailCat under conditions of actual usage.

Database	# Folders	# Messages
R. Segal	66	814
J. Kephart	56	1420
User #3	43	2433
User #4	34	473
User #5	15	553
User #6	14	3020

Table 1: Mail databases used to test MailCat. The databases include the mailboxes of the two authors as well as the mailboxes of several colleagues at the IBM Thomas J. Watson Research Center. Generally, the classification task becomes increasingly difficult as the number of folders increases.

In the first experiment, we applied MailCat's text classifier to the mailboxes of six researchers at the IBM Thomas J. Watson Research Center. Table 1 presents the characteristics of each of the six databases. The databases range in size from 473 to 3,020 messages that have been filed in 14 to 66 folders. The number of folders is an important quantity because, as it increases, so does the difficulty of the classification problem. The experiment was conducted using each user's previously-filed messages as data for the experiment. We randomly sampled 70% of each user's previously-filed messages for training and used the remaining 30% of the messages for testing. We repeated the experiment ten times and averaged the results.

Figure 3 shows the results of this experiment. The graph shows the accuracy of MailCat for each user under the assumptions that MailCat provides from one to five shortcut buttons. The accuracy of MailCat with N buttons is defined as the frequency with which one of the N buttons represents the correct folder. The results indicate that MailCat is fairly accurate even with one button, achieving between 60% and 80% accuracy. If MailCat were automatically filing messages, an error rate of 20% to 40% would be absolutely unacceptable. However, since MailCat's incorrect predictions are easily overridden, the 20% to 40% error rate is only a very minor annoyance, and more than compensated by the benefit of receiving the right suggestion 60% to 80% of the time.

With three shortcut buttons on each message, the accuracy of MailCat's suggestions improves to 80% to 98%. The use of three buttons rather than one reduces the error rate by a factor of two to four. The substantial improvement in going from one button to three is countered only very slightly by the need for the user to consider three choices rather than one.

What is the optimal number of shortcut buttons? The experiment above shows that using five buttons rather than three cuts the error rate in half for only two of the databases. For the four other databases, the error rate is cut by about 65%. While this is a substantial reduction, it is not as dramatic as the gain achieved going from one button to three. With five buttons, the adverse effects become more pronounced. The two extra buttons threaten to crowd the screen, and also require the user to select from among five choices. This may become a noticeable cognitive burden. For these reasons, three buttons seem to be about right.

Figure 3 also compares the performance of MailCat to the naive strategy of providing shortcut buttons for the N most frequently used folders. For the first four databases,

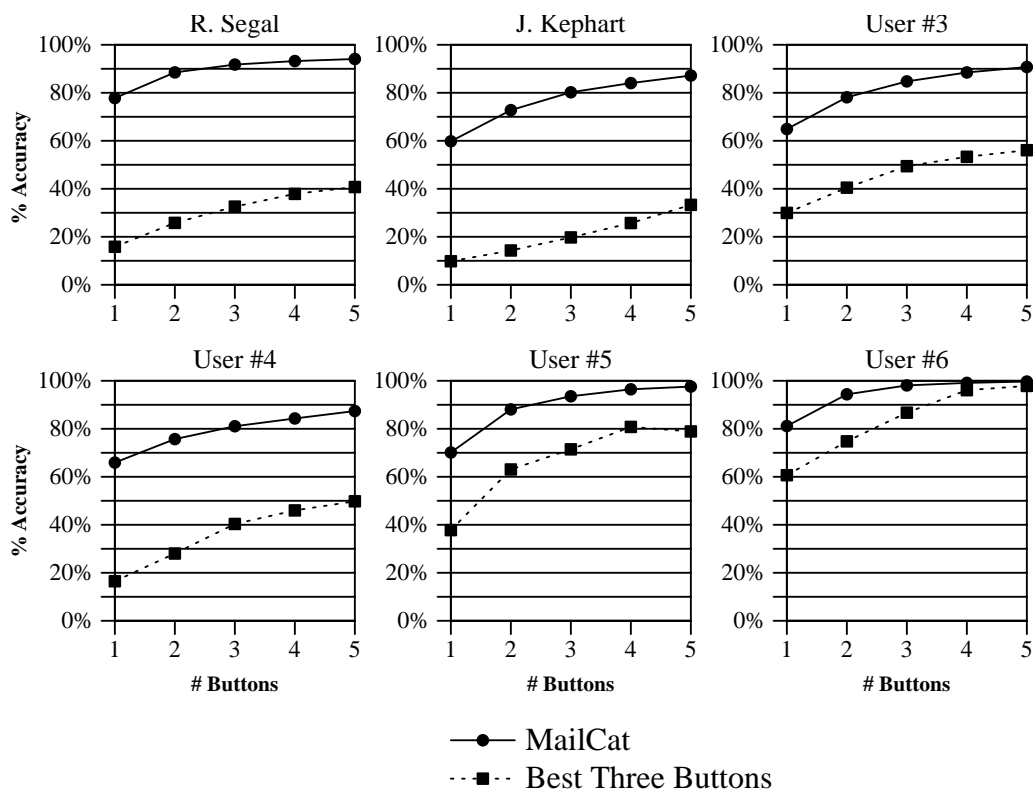


Figure 3: Results for simulating MailCat on six mailboxes. Solid lines show the accuracy of MailCat with one to five buttons. Dotted lines compare these results to the naive strategy of providing shortcut buttons for the N most-frequently-used folders. MailCat performs substantially better than the naive strategy, providing over 80% to 90% accuracy with three buttons.

which all have greater than thirty folders, the accuracy of the naive strategy for three buttons ranges from 20% to 50%. While this is surprisingly good, it is substantially less than the 80% to 90% accuracy provided by MailCat on these same four databases. The naive strategy performs well on the remaining two databases, which both have about fifteen folders. For three buttons, the naive strategy is 71% accurate for user #5 and 87% accurate for user #6. These users are apparently only actively using just a few of their folders. MailCat improves substantially over these already high accuracies — providing a factor of four or more reduction in the total error rate. However, it is unclear whether users would perceive these large differences in the error rate, given that the accuracy rate is already quite high.

In the second experiment, both authors used a specially instrumented version of MailCat to record its performance for a period of two months. Evaluating MailCat on real databases is important because actual use is much more dynamic than what can be captured with static experiments. Real users dynamically create, delete, and rearrange folders. Furthermore, the meaning and content of individual folders may drift over time. Evaluating MailCat under actual usage is also important to evaluate the inaccuracy introduced by using daemons to classify messages and track changes. The daemons introduce a time delay between when changes are made to the database and when the classifier is updated to reflect these changes. They also introduce a delay between the time at which a message is classified and the time at which the shortcuts derived from that classification are used to file the message, particularly if messages are allowed to

sit in the inbox for a long period without being re-classified.

Figure 4 shows the performance of MailCat on our databases over a two-month period. The results are based on 183 filings for Segal’s database and 353 filings for Kephart’s database. The results show performance similar to the static experiment. The performance during actual usage is slightly worse for Segal’s database and slightly better for Kephart’s database. These results suggest that the dynamics of real usage and the inaccuracies introduced by daemons are not problematic. MailCat is effective in practice.

6 Related Work

MailCat’s philosophical underpinnings are probably most related to those of CAP [Mitchell *et al.*, 1994], a calendar scheduling application. CAP uses its learning of user preferences to assist users in scheduling meetings. When a user wishes to create a new meeting, CAP queries him for details such as the desired meeting time, location and duration. CAP uses its predictions to provide default values for each of its questions. When CAP’s predictions are correct, the user can simply hit return to accept the default value. The user may override these defaults simply by typing in their own desired response. Both MailCat and CAP are unobtrusive assistants because they offer convenient, overridable shortcuts to the user rather than taking possibly incorrect actions on the user’s behalf.

Maxims [Maes, 1994] helps users perform a variety of operations on their electronic mail, of which folder orga-

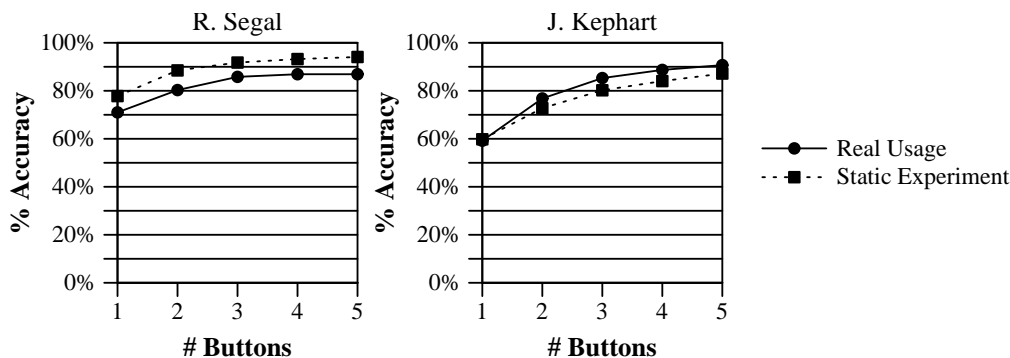


Figure 4: The performance of MailCat during two months of actual usage.

nization is just one example. Maxims monitors each interaction between a user and the Eudora mail reader and stores a record of each interaction as a situation-action pair. Memory-based reasoning is used to anticipate a user's actions. MBR searches for close matches between the current situation and previously-encountered situations. When a close match is found, MBR uses the previously-taken action or actions associated with that situation to predict what action the user is likely to take. Maxims either carries out the predicted action automatically or provides a single shortcut to the user that facilitates that action.

There are several drawbacks to the approach taken by Maxims. First, it can take some time for its learning algorithm to gain enough experience to be useful. Maxims addresses this problem by allowing a newly-instantiated agent to learn from more established agents. However, it is doubtful that one user's e-mail agent could teach another user's e-mail agent anything useful about mail filing because mail-filing schemes are based on individual preferences. Maxims cannot learn from previously-filed messages because it tries to predict the more general concept of action rather than the simpler concept of destination folder. Actions include steps such as replying to messages and deleting messages. Since most mail readers do not keep histories of these events, Maxims cannot bootstrap itself using data stored by the mail reader. Finally, Maxims' learning algorithm is not incremental. As Maxims grows in experience, so does the amount of time required for learning. While Maes suggests that the classifier can be updated overnight, this may be too infrequent when the user is actively creating folders and reorganizing messages.

Payne and Edwards [1997] describe an e-mail agent similar to Maxims. Although they allow for the possibility of incremental learning, they do not address the issue of jump-starting the classifier by training from existing mail. The systems proposed by Payne and Edwards and by Maes do not offer multiple suggestions. MailCat's use of three recommendations substantially improves its accuracy without any cost to the user.

Finally, Cohen [1996] compares the relative merits of two procedures for text classification and uses several sets of e-mail corpora to test those procedures. The emphasis of his work is on comparing the performance of the two classifiers; he neither describes nor evaluates a complete agent.

7 Final remarks

MailCat's hallmark is simplicity. Users have nothing extra to learn or do, and can reap the benefits of MailCat as soon as it is installed. MailCat has virtually no adverse side-effects. If its predictions are wrong, the user can bypass MailCat effortlessly. MailCat's classification performance is excellent (80% to 90%) on real users' databases with thousands of archived messages and as many as sixty folders. MailCat's classification and learning algorithms consume only a fraction of a second per message, and this small impact is further lessened by daemons that carry out these activities in the background. Informal interviews with users indicate that MailCat substantially lowers the barriers that have prevented many users from filing their mail, and may encourage more users to become diligent filers. This will enable them to retrieve mail more efficiently.

While MailCat was developed for electronic mail, it can easily be used to organize other types of electronic documents. The concepts behind MailCat can be applied to organizing bookmarks, audio recordings, files, and other text-based documents that are placed into a hierarchy of folders.

References

- [Barrett and Selker, 1995] Robert Barrett and Ted Selker. AIM: A new approach for meeting information needs. Technical report, IBM Research, October 1995.
- [Barrett *et al.*, 1997] Rob Barrett, Paul P. Maglio, and Daniel C. Kelleem. How to personalize the web. In *Proceedings of the 1997 Conference on Human Factors in Computing Systems*, March 1997.
- [Cohen, 1996] William W. Cohen. Learning rules that classify e-mail. In *Proceedings of the 1996 AAAI Spring Symposium on Machine Learning and Information Access*. AAAI Press, 1996.
- [Kirsche and Barrett, 1996] Thomas Kirsche and Rob Barrett. Subject based searching using automatically extracted metadata. Technical report, IBM Research, December 1996.
- [Maes, 1994] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31-40, July 1994.

- [Mitchell *et al.*, 1994] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, July 1994.
- [Payne and Edwards, 1997] Terry R. Payne and Peter Edwards. Interface agents that learn: An investigation of learning issues in a mail agent interface. *Applied Artificial Intelligence*, 11:1–32, 1997.
- [Salton and McGill, 1983] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1983.