SwiftFile: 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 e-mail clients allow users to file messages into folders, the process they must go through to file each message is often tedious and slow. For each message, the user must first decide which folder is most appropriate. Then, the user must inform the e-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. SwiftFile 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 effortlessly file it into a predicted folder. For typical users, SwiftFile'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.

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 e-mail clients allow users to organize their messages into user-defined folders. Typically, a user may maintain somewhere between 10 and 100 separate folders — possibly organized into a hierarchy. Users who diligently file their e-mail can often immediately select the correct folder when attempting to retrieve a particular message, greatly enhancing retrieval speed and accuracy.

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. Often, the user must scroll down to select the desired folder, which can be tedious if there are many folders. Notes also 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, the cognitive effort required to decide upon an appropriate folder and locate the icon or menu item representing it is often substantial enough to discourage users from filing at all, particularly when faced with the prospect of repeating the exercise dozens of times per day.

SwiftFile is an intelligent personal assistant that reduces the cognitive burden and the time required for organizing electronic mail into folders (Segal and Kephart 1999). 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 them. When one of the folders predicted by SwiftFile 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 SwiftFile's suggested choices, and they can do so with just a single mouse click. This amounts to a significant qualitative enhancement that is likely to encourage users to file their mail.

SwiftFile was designed with careful attention to making the system as easy to use as possible. Users are notorious for having short attention spans. If a personal assistant has a steep learning curve or it takes an extended period for the assistant's learning algorithm to develop a sufficient model of the user to be useful, most users will give up on it before it can provide any benefits. The beauty of SwiftFile is that it helps users file e-mail without demanding anything in return. Users have nothing new to learn or do. When first installed, SwiftFile analyzes a user's existing folders to construct an initial text classifier that ensures that its predictions are useful right from the start. SwiftFile continuously updates this classifier as the user interacts with the system without requiring any explicit training from the user. Learning to use SwiftFile is easy — the user need only press one of SwiftFile's self-explanatory buttons to quickly move a message into the appropriate folder.

We also designed SwiftFile to be risk-free. Many existing e-mail assistants will perform automatic operations on incoming e-mail, such as automatically placing a message in a folder or automatically deleting junk

mail (Maes 1994, Boone 1998, Spertus 1997). These systems use a variety of different learning algorithms to decide which actions needs to be performed for which messages. However, even if their learning algorithms are 95% accurate, these systems will incorrectly handle 5% of your e-mail. If 5% of your e-mail was misfiled or inappropriately deleted, the assistant quickly becomes more of a nuisance than a help. SwiftFile avoids this problem by *suggesting* actions rather than taking them. When its predictions are incorrect, the only adverse effect is that the user must revert to filing messages using the standard interface.

This paper is organized as follows. In the next section, we give an overview of SwiftFile's design philosophy and structure. In section 3, we describe a modified TF-IDF text classifier that supports low-cost incremental updates; this is the key to SwiftFile's adaptivity. In section 4, we discuss some implementation issues that arise when integrating SwiftFile with Lotus Notes, some of which are likely to extend to other mail applications as well. An evaluation study reported in section 5 establishes that SwiftFile performs well in practice. We conclude with a comparison to related work in section 6 and some final remarks in section 7.

Overview of SwiftFile

The SwiftFile concept is sufficiently simple and general to apply to any of today's major mail applications. We have implemented SwiftFile as an add-on to the Lotus Notes mail client. Figure 1 shows how SwiftFile simplifies the task of organizing messages in Lotus Notes. SwiftFile 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 SwiftFile predicts to be the message's most likely destinations. The buttons are ordered from top to bottom, with the topmost representing SwiftFile's best guess as to where to file the message and the bottommost representing its third-best guess. When one of the three buttons is clicked, the message is immediately moved to the indicated folder.

SwiftFile's predictions are made using a text classifier. Classifiers often require a large number of training messages before they yield accurate predictions, but SwiftFile circumvents this potential problem. During installation, it treats previously-filed messages as a corpus of labeled documents and uses standard TF-IDF techniques to construct its text classifier. After its initial training is complete, SwiftFile is immediately ready to make accurate predictions.

Training 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 them may drift over time. For instance, the nature of SwiftFile-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. More recent messages have discussed ongoing experiments and this paper. Had the text classifier learned a model of our SwiftFile folders when we first started using SwiftFile, that model would probably perform poorly today.

SwiftFile 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 only linear in the length of the message. After updating, the classifier's predictions are identical to those that would be obtained by training the classifier from scratch on the entire mail database.

Interestingly, while SwiftFile's use of three buttons rather than one may seem trivial, it is in fact quite significant because it substantially improves SwiftFile's usefulness. As will be shown later in the section on experimental results, the use of three buttons cuts SwiftFile's failure rate in half without introducing any adverse side-effects. SwiftFile's use of three buttons is a direct result of our decision to provide an assistant that facilitates rather than automates message filing. 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.

Text Classifier

SwiftFile's text classifier is a modified version of AIM (Barrett and Selker 1995), a TF-IDF style classifier (Salton and McGill 1983) developed at IBM's Almaden research laboratory.

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

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

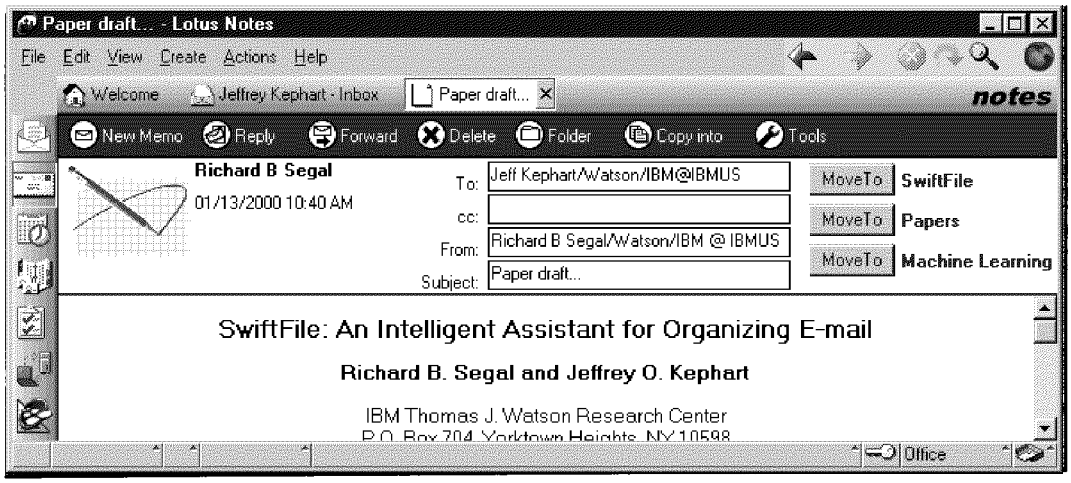

Figure 1: Shortcut buttons provided by SwiftFile. SwiftFile creates shortcut buttons for the three folders that it predicts to be the most likely destinations. When SwiftFile's predictions are correct, the buttons enable the user to file messages into the indicated folder with a single button click.

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 distance called *SIM4* (Salton and McGill 1983):

$$SIM4(\mathcal{M}, \mathcal{F}) = \frac{\sum_{w \in \mathcal{M}} F(\mathcal{M}, w)\, W(\mathcal{F}, w)}{\min(\sum_{w \in \mathcal{M}} F(\mathcal{M}, w), \sum_{w \in \mathcal{M}} W(\mathcal{F}, w))}. \quad (6)$$

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

AIM implements the above classification algorithm in a way that allows it to support full incrementality. 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.

To classify a message $\mathcal{M}$, AIM computes the *SIM4* similarity between $\mathcal{M}$ and each folder's weight vector. The weight terms $W(\mathcal{F}, w)$ required to compute the *SIM4* metric can be computed on the fly from the folder centroids. Once the *SIM4* similarity score has been computed for each folder, AIM predicts the message as belonging to the three folders with the highest score.

AIM's learning algorithm is fully incremental in the sense that 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 SwiftFile 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 SwiftFile, 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.

Lotus Notes Implementation

SwiftFile has been implemented as an extension to Lotus Notes. In Notes, all data, including electronic mail, are 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. There is also a C++ API for directly accessing and modifying Notes databases.

SwiftFile'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 SwiftFile 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, SwiftFile would classify messages just before they were displayed to the user. However, the slight 300 millisecond delay required to classify a message on a 166 MHz Pentium II is just long enough to annoy some users. Therefore, SwiftFile uses a daemon that checks for new mail every sixty seconds. When the daemon finds a new message, the daemon classifies it using SwiftFile'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 classifier accuracy. The daemon only classifies messages when they first arrive. However, SwiftFile is continously 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 the user to manually refresh the computed shortcut buttons to ensure that they remain as accurate as possible.

The second daemon is responsible for updating the classifier as changes are made to the user's mail database. 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.

Experiments

We performed two sets of experiments on SwiftFile. The first assesses the performance of SwiftFile's classifier on static mailboxes, while the second analyzes how well SwiftFile's classifier performs over time.

In the first experiment, we applied SwiftFile's text classifier to the mailboxes of six employees at IBM Research. 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.

Database	# Folders	# Messages
R. Segal	66	814
J. Kephart	56	1,420
User #3	43	2,433
User #4	34	473
User #5	15	553
User #6	14	3,020

Table 1: Characteristics of the six mail databases used to test SwiftFile. Generally, the classification task becomes more difficult as the number of folders increases.

Figure 2 shows the results of this experiment. The graph shows SwiftFile's accuracy for each user under the assumptions that SwiftFile provides from one to five shortcut buttons. The accuracy for N buttons is defined as the frequency with which one of the N buttons represents the correct folder. SwiftFile is fairly accurate even with just one button, achieving between 60% and 80% accuracy. If SwiftFile were automatically filing messages, an error rate of 20% to 40% would be unacceptable. However, since SwiftFile's incorrect predictions are easily overridden, the 20% to 40% error rate is only a minor annoyance that is more than compensated by the benefit of receiving the right suggestion 60% to 80% of the time.

With three shortcut buttons for each message, the accuracy of SwiftFile'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? According to Figure 2, the use of 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 they 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.

In the second experiment, we evaluated SwiftFile's incremental learning algorithm by analyzing how well it performs over time. We used the same mail databases that were used in the first experiment to simulate a dynamic environment in which new messages arrive in a continuous stream and are filed by the user in the order in which they are received. The simulator works by sorting the messages in a static mail database by date and presenting each message to SwiftFile in turn. As each message arrives, it is classified by SwiftFile using its latest classifier, and the classification is compared against the known correct classification. The message

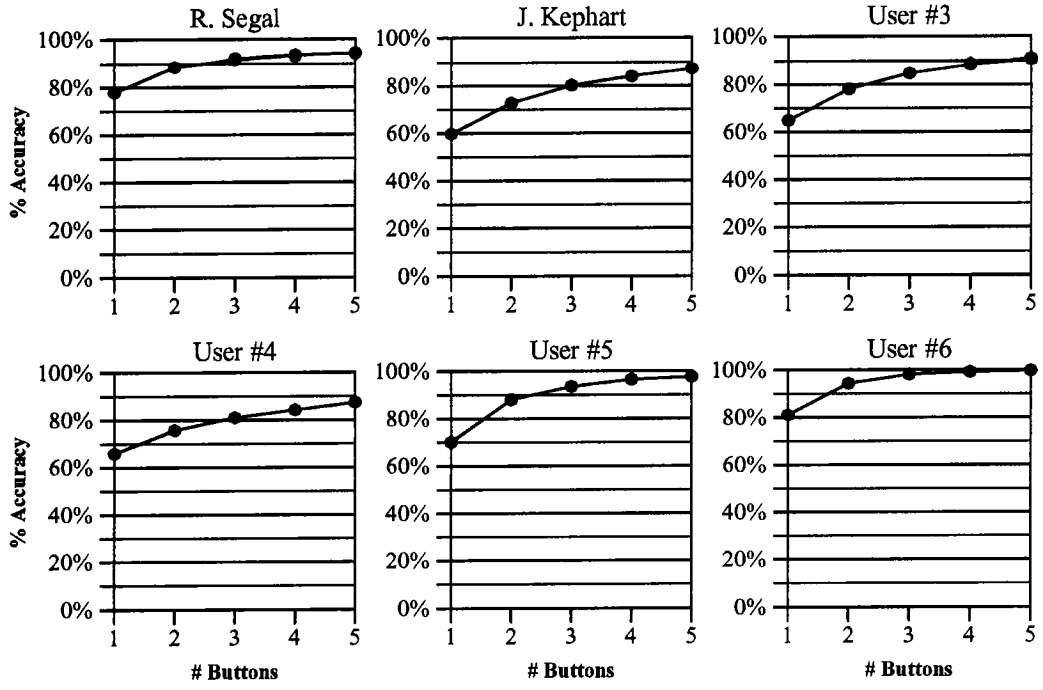

Figure 2: Results for simulating SwiftFile on six mailboxes. The graph shows the accuracy of SwiftFile with one to five buttons. Note, SwiftFile is over 80% to 90% accuracy with three buttons.

is then filed into the destination folder, and the classifier's model is updated accordingly. The simulator assumes there are no folders at the beginning of the simulation and assumes a new folder is created whenever the simulator processes the first message for each folder.

Figure 3 shows the results of this experiment when applied to the database of R. Segal. The solid line represents a moving average of the classifier's three-button accuracy as a function of the number of classified messages. Interestingly, SwiftFile's accuracy starts out at 100% and then slowly degrades to a respectable 90% before leveling off. What is suprising is that SwiftFile's accuracy starts at 100% even though it has yet to see any data! The reason is that, when a user first starts using e-mail and has only one or two folders, SwiftFile's three predictions are guaranteed to cover one of them. As a result, SwiftFile is perfect until the user has seen enough e-mail to warrant more than three folders. This trend continues such that once the user has created enough folders to make the classification problem challenging, SwiftFile has seen enough examples to have learned a good classifier. The result is that the user never experiences poor performance.

Figure 3 also shows that SwiftFile's incremental learning algorithm maintains a fairly high level of accuracy throughout the experiment. The accuracy of SwiftFile stays above 80% through most of the experiment, and it is always above 70%. Incremental learning ensures that the user experiences a consistent level of performance.

Designers of e-mail assistants often assume that incremental learning is not necessary because batch learning can be performed overnight. However, overnight learning cannot adapt quickly enough to instantaneous changes in how a user files e-mail, such as the creation of a new folder or the reorganization of an old folder. We can test this empirically by repeating our experiment with a slight difference: to simulate batched learning, the classifier's update frequency is reduced from once per message to once per thirty messages.

The dotted line in Figure 3 shows the results of this experiment. While the performance of periodic learning is often comparable to that of continuous learning, there are several extended episodes during which periodic learning is substantially less accurate. Furthermore, the performance of overnight learning is much less consistent, with the accuracy rate dropping as low as 50% at one point. The cumulative accuracy of periodic learning is measured to be 7% less than for continuous learning. While it may be possible to implement overnight batch learning, doing so can result in a large reduction in classification performance.

Related Work

There is a substantial body of work investigating the use of text classification to help users perform various tedious operations with their e-mail (Maes 1994, Payne and Edwards 1997, Boone 1998, Cohen 1996, Spertus 1997). Much of the work has focused on accurately predicting which actions should be performed on newly-arriving e-mail, while ignoring equally important issues

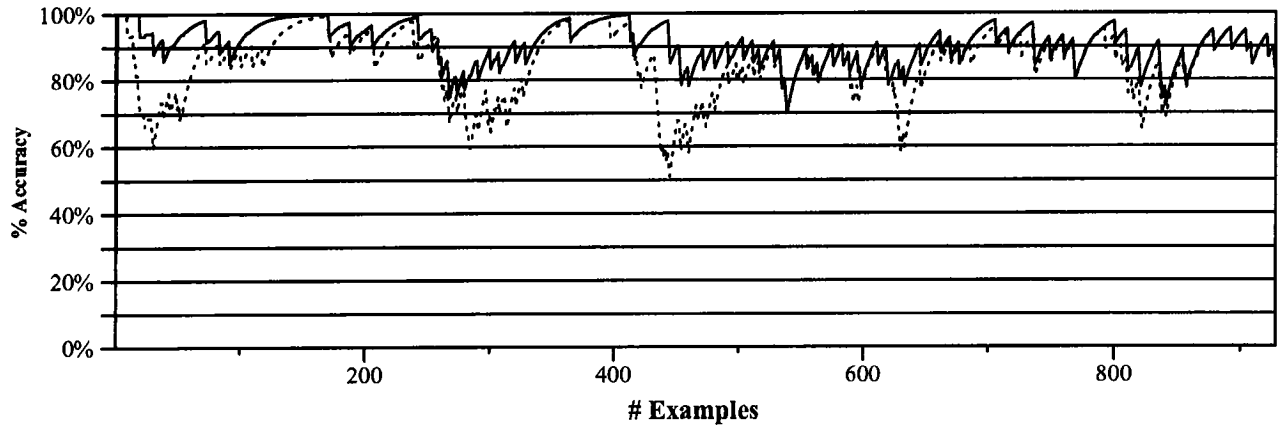

Figure 3: Simulation of SwiftFile's performance on a continuous stream of messages. Solid line: performance of SwiftFile when its classifier is updated after each message is filed. Dotted line: performance of SwiftFile when the classifier is only updated every thirty messages to approximate the effects of overnight learning.

regarding how the classifier fits into the system as a whole. There are a few notable exceptions. Maes (Maes 1994) advocates using collaborative filtering to quickly bootstrap a classifier for new users. While this method can work well for some tasks such as filtering junk e-mail, it is doubtful that it would work well for mail filing because mail-filing schemes are very much an individual matter. Payne and Edwards (Payne and Edwards 1997) do consider the possibility of incremental learning, but very few details are provided.

SwiftFile's philosophical underpinnings are probably most related to those of CAP (Mitchell *et al.* 1994), a calendar-scheduling application. CAP learns to predict how users will respond to the various questions it must ask in scheduling a meeting and offers its best prediction as a default value. If CAP's predictions are correct, the user can simply hit return to accept the suggested value. Otherwise, the user may override the default by typing in a different response. Both SwiftFile 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.

Final Remarks

SwiftFile is an easy-to-use personal assistant that helps users file their e-mail into folders. SwiftFile makes very few demands on users; they have nothing extra to learn or do and can reap the benefits of SwiftFile as soon as it is installed. SwiftFile has virtually no adverse side-effects. If its predictions are wrong, it may be bypassed easily. SwiftFile's classification performance is excellent (80% to 90%) on real users' databases with thousands of archived messages and as many as sixty folders. Informal interviews with users indicate that SwiftFile substantially lowers the barriers that have prevented many users from filing their mail and thus encourages more users to become diligent filers.

While SwiftFile was developed for electronic mail, the same technique can easily be used to organize other types of electronic documents such as bookmarks, audio recordings, disk files, and other text-based documents that are placed into a hierarchy of folders.

References

Robert Barrett and Ted Selker. AIM: A new approach for meeting information needs. Technical report, IBM Research, October 1995.

Gary Boone. Concept features in Re:Agent, an intelligent email agent. In *Proceedings of the Second International Conference on Autonomous Agents*, 1998.

William W. Cohen. Learning rules that classify e-mail. In *Proceedings of the 1996 AAAI Spring Symposium on Machine Learning and Information Access*, 1996.

Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 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.

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.

Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, 1983.

Richard B. Segal and Jeffrey O. Kephart. MailCat: An intelligent assistant for organizing e-mail. In *Proceedings of the Third International Conference on Autonomous Agents*, May 1999.

Ellen Spertus. Smokey: Automatic recognition of hostile messages. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, July 1997.