EmailValet: Learning User Preferences for Wireless Email

Sofus A. Macskassy[†‡], **Aynur A. Dayanik**[†], **Haym Hirsh**[†]
{sofmac,aynur,hirsh}@cs.rutgers.edu

[†]**Department of Computer Science**
Rutgers University
110 Frelinghuysen Rd
Piscataway, NJ 08854-8019

[‡]**Information Architects**
Morrocroft II
2064 Colony Road
Charlotte, NC 28211

Abstract

This paper presents EmailValet, a system that learns users' email-reading preferences on email-capable wireless platforms – specifically, on two-way pagers with small "qwerty" keyboards and an 8-line 30-character display. In use by the authors for about three months, it has gathered data on email-reading preferences over more than 8900 email messages received by the authors during this period. The paper presents results comparing the ability of different learning methods to form models that can predict whether a given message should be forwarded to the user's wireless device. Our results show that the best performance of one method, over a range of established learning methods developed on the information retrieval and machine learning communities, was able to achieve a break-even point of over 53% for one user that had received over 5000 messages. We also find that, in general, all methods are able to achieve better performance than what would be achieved by a baseline of simply forwarding all messages to the wireless device, and that many methods are able to find procedures that, although they forward only a small fraction of the messages that a user would want, achieve 100% precision on those messages that it does actually choose to forward.

1 Introduction

Imagine that while commuting home you receive an email message on your pager from your spouse asking you to pick up some milk on your way home. Compare this with commuting home and receiving an email message on your pager from a coworker with a monthly reminder to use the recycle bins at work. One of these messages would likely be relevant to you at that particular point in time, while the other would not be. It would be nice if we could automatically decide which messages to send to the pager in such a way that the first message would be forwarded and the second one would not. Furthermore, we would like such forwarding to take place in a transparent fashion for the people sending and receiving the email. For example, it would be desirable to have a single, centralized email address to which email would be sent, and some software agent residing at that location could then forward each message to where the receiver would like to read it, eliminating the need for both receiver or sender to figure out where to read a message from or send a message to. Ideally, this agent would work in a way much akin to a secretary, knowing where the user is and knowing which messages to forward to where. Thus, for example, if the user had a palmtop device with a wireless data service, a pager, a computer at home, and a desktop at work, the agent would need to decide where a message should be delivered, based on a number of factors, such as what the message is about, and when and where the user would want to read this message. Obviously different people would have different preferences even in identical circumstances, so the agent must tailor itself the user's email-reading preferences. We call such an agent an EmailValet, in that it provides "valet"-like functionality to help users manage email.

In this paper, describing work in progress, we discuss a first instance of such an EmailValet. More specifically, this EmailValet interacts with its user through a two-way pager that can both receive alpha-numeric pages sent to the pager's email address, and send messages composed on the device's small "qwerty" keyboard. The EmailValet tries to learn which messages to forward to the pager by "looking over the shoulder" of the user, keeping track of user actions as email messages are sent and received. The end goal for this type of system would be for the EmailValet to eventually take over and only take the actions that the user would want, such as only forwarding those messages that it knows the user would want forwarded to the pager. This paper presents experimental studies on the ability of a range of learning approaches to learn from the history of interactions the authors have taken with the EmailValet over a period of three months. One key question we ask is the extent to which non-content information (for example, the time-stamp of a message, as opposed to its text body) can improve learning. Thus, for example, if you are sitting in front of your desktop (probably during normal business hours), you are less likely to want a message on your pager, and more likely to want it sent to your desktop. Conversely, if you have left work, you might be more likely to want an important message sent to your pager than have it wait on your desktop for reading the following business day. This is in contrast to much of the existing work on email filtering, which focuses on the text content of email. Although the EmailValet is currently being used to send and receive email, this paper discusses an "offline" analysis of a range of learning approaches on the email-processing actions of the three authors over a period of three months use of the EmailValet.

In the remainder of this paper we first describe the architecture of our EmailValet in more detail, followed by a discussion of the experimental design for our study. We then present our main results, in which we compare the performance of a range of established methods from the machine learning and information retrieval community, including the Naive Bayes, TFIDF, and Ripper learning methods. In the best case, we are able to achieve a break-even[1] point of 53%. We also present some more detailed analyses of the various methods, including how different ways of encoding the email for the learner affect learning performance. We conclude the paper with a discussion of future work and final remarks.

2 EmailValet Architecture

Our EmailValet is based on two-way paging devices available from BellSouth Wireless Data Services (www.bellsouthwd.com/ips). The equipment used by all three authors were RIM Inter@ctive Pagers, whose functionality through the BellSouth Wireless services includes sending and receiving text messages both through BellSouth's proprietary interactive paging^SM network, as well as to and from arbitrary internet email addresses.

Through the BellSouth interactive paging^SM service these devices each have a unique email address. Rather than having email senders address these devices directly, we instead maintain our Rutgers email account as the central location to which email is sent. However, we created an agent, the EmailValet, at the Rutgers mailserver that filters all of the recipient's email transparently to both the message's sender and receiver. The headers of all received messages are then passed on to the user's pager, with the headers modified so that any reply from the device goes back to the agent (i.e., to the user's own email address) rather than directly to the original message sender. Upon receipt of the message headers at the pager, the user can then reply, requesting (among other options) the full text of the message from the EmailValet, again with the "reply-to" field reset to the user's own EmailValet agent. Upon receipt of the full message the user can then reply to the email, with the reply actually going to the EmailValet. The EmailValet then "repackages" any outgoing email to appear to be from the user's Rutgers account and sends it to the intended email recipient. The Rutgers EmailValet has been in constant use through this interface by the three authors for roughly three months, representing over 8900 messages received by the authors.

3 Experimental Methodology

This paper presents the results of a range of learning methods on data obtained from the authors' use of the EmailValet over a three month period. This section presents the different learning algorithms used, the encoding of the data for the learners, and the evaluation methodology.

3.1 Learning Algorithms

Our experiments concern the use of five different learning algorithms often used for text categorization: Naive Bayes [Mitchell, 1997], TFIDF, Probabilistic TFIDF [Joachims, 1997], Ripper [Cohen, 1995; 1996], and Winnow [Littlestone, 1988; Blum, 1997]. We used publicly available versions of the first four systems (Rainbow [McCallum, 1996] for the first three, and the version of Ripper available from www.research.att.com/~wcohen). We used our own implementation of Winnow.

The **Naive Bayes** classifier uses Bayes' rule to estimate the probability of each category for a given document, based on the prior probability of a category occurring, and the conditional probabilities of particular words occuring in a document given that it belongs to a category, assuming that these probabilities are conditionally independent. Joachims [1997] and Mitchell [1997] give further details on the use this algorithm for text categorization.

The **TFIDF** ("Term Frequency times Inverse Document Frequency") classifier [Joachims, 1997] is based on the relevance feedback algorithm by Rocchio [1971] using vector space retrieval model [Salton, 1991]. This algorithm represents documents as vectors so that documents with similar content have similar vectors. Each component of such a vector corresponds to a term in the document, typically a word. The weight of each component is computed using the TFIDF weighting scheme, which tries to reward words that occur many times but in few documents. In the learning phase, a prototype vector is formed for each class from the positive and negative examples of that class. To classify a new document d, the cosines of the prototype vectors with the corresponding document vector are calculated for each class. d is assigned to the class with which its document vector has the highest cosine.

The **Probabilistic TFIDF** classifier [Joachims, 1997] is a probabilistic version of the TFIDF classifier, based on estimation of the probability of a category C given document d, $Pr(C|d)$, using the retrieval with probabilistic indexing method proposed in [Fuhr, 1989]. To classify a new document d, $Pr(C_j|d)$ is estimated for each class, C_j, (as described in more detail by Joachims [1997]). d is assigned to the class whose probability is the highest.

Ripper [Cohen, 1995; 1996] is a learning method that forms sets of simple rules for data described by sets of attribute-value pairs. Each rule tests a conjunction of conditions on attribute values. Rules are returned as an ordered list, and the first successful rule provides the prediction for the class label of a new example. Importantly, Ripper allows attributes that take on sets as values, in addition to numeric and nominal features, and a condition can test whether a particular item is part of the value that the attribute takes on for a given example. Rules are formed in a greedy fashion, with each rule being built by adding conditions one at a time, using an information-theoretic metric that rewards tests that cause a rule to exclude additional negative data while still hopefully covering many positive examples. New rules are formed until a sufficient amount of the data has been covered. A final pruning stage adjusts the rule set in light of the resulting performance of the full set of rules on the data.

Winnow[2] [Littlestone, 1988; Blum, 1997] learns linear threshold functions using a multiplicative weight-updating scheme. It works with a set of experts, each of which can either make some prediction about the class of an example or abstain from predicting. Each prediction is given some weight, or vote, and the class that gets the highest overall vote is re-

[1] As will be discussed later, this metric, commonly used in the information retrieval literature, says that 53% of all messages that should be forwarded do get forwarded, and that 53% of all messages that were forwarded were from the set of messages that should have been forwarded.

[2] Strictly speaking, we are making use of the basic "Winnow II" strategy [Littlestone, 1988]. Since, over time, the number of specialists can increase unboundedly, we are using the *infinite attribute model* by Blum [Blum, 1992; Blum *et al.*, 1991].

turned as the final class. The goal of learning is to find a suitable weighting scheme for each expert. In this work each specific expert corresponds to some property of a message. For a new message it consults the last five times messages with this property were previously seen (or fewer if there are less than five previous occurrences),[3] and returns a vote for whichever class was the majority amongst the previous examples. For a given message, only those experts whose associated properties are in the instance to be classified cast a vote; all other experts abstain, as do experts appearing in an example for the first time, since there are no previous examples of this property. When a new property is seen for the first time, a new expert is created with a weight of 1. Learning is achieved by updating the weights of an expert *only* when the overall prediction for an example is incorrect. In such cases, the weights of experts who predicted correctly are increased by multiplying them by some $\alpha > 1$, and the weights of experts who predicted incorrectly are decreased by multiplying them by some $\beta < 1$. Based on informal studies, since no values of α or β seemed clearly superior to any other, their values were somewhat arbitrarily set to $\alpha = \frac{3}{2}$ and $\beta = \frac{2}{3}$. The weights of abstaining experts are unchanged in case of incorrect predictions. Finally, Winnow is an incremental learning method, updating the weights after each example, and given a fixed collection of examples, our implementation makes only one pass through the data, running through the examples in order and updating weights after each incorrect prediction.

3.2 Experimental Setup

Full traces of the EmailValet were maintained for all users. Included in the user logs were records of all new email received by the EmailValet. Recall that the headers of each new message are then forwarded to the remote pager, and the user can select to receive the full text of each such message. Information on which messages were so selected are thus also available from these logs. As a result, two sets of email were created for each user, one containing those messages that the user chose to see, and a second that the user did not choose to see. We set for ourselves the task of learning to distinguish these classes, to learn which messages should have their full body sent to their user's pager.

Since data are chronological in nature, it is not appropriate to divide the data into random training and testing sets, since a later message should not be part of the training data that led to a classifier used on an earlier message. Instead, data were divided into two contiguous segments. The first two-thirds of a user's data were taken as the training set for all learning methods, and the last one third was taken as the test set for each method. In the case of Winnow, after learning has taken place on the first two-thirds of the data, the weights are frozen for the testing stage on the test data. All experiments reflect a two-third/one-third split *after* the first 5% of the data has been removed, since this reflects a period of time in which the user was first learning how to use the EmailValet.

For evaluation purposes we compute precision/recall curves for each method. The precision of a classification method is the proportion of data that is labeled positive that really is positive. In our case it corresponds to the proportion of messages that were labeled as forward that really should have been forwarded. The recall of a classification method is the proportion of all truly positive data that are labeled as positive. Here this corresponds to the proportion of messages that should have been forwarded that were labeled forward. Together these two values reflect a trade-off between the coverage of a classifier and its accuracy on the class that is of interest. Since for different people different points on this trade-off may be more or less desirable, we use precision/recall curves to better reflect the range of possibilities that a method provides.

We also consider a second evaluation method, the break-even point of a method. This method attempts to find a point where precision and recall are roughly equal. Although it focuses on only one point on the precision/recall curve, it yields a way to obtain single-value evaluations of a method, simplifying the process of making direct comparisons between methods.

Precision/recall curves are created for the Rainbow methods by sorting instances based on the score assigned by the method for the forward class for that instance. Setting a threshold on this score allows varying the number of examples predicted as forward. This threshold can be varied so as to include examples in the list one-by-one,[4] yielding a precision/recall point for each threshold. Thus, for example, at one extreme all messages are forwarded, yielding a baseline of 100% recall, but typically low precision. The various precision/recall points can then be pruned, where a point is deleted if there is another observed point with better precision and recall. The resulting points can then be plotted as a curve, yielding our final precision-recall graph. Break-even points are obtained directly from Rainbow, yielding the precision and recall that come closest to being equal. The values we report as break-even points are the average of these two quantities.

Precision/recall curves were created for Ripper by varying its misclassification-cost parameter This number was varied from 0.05 to 2.0 in increments of 0.05, with each run giving one precision/recall data point. Precision/recall curves were created from these points in the same fashion as was done for Rainbow's methods. Break-even points were found by the average of the precision and recall for either the last point on the curve, if it didn't cross the precision=recall line, or the last point in the curve before crossing that line.

Precision/recall curves were created for Winnow by sorting the weighted forward vote given to each instance. Precision/recall curves were computed as for the Rainbow methods, and the break-even point was determined as for Ripper.

3.3 Data

As discussed in the preceding subsection, a simple classification problem — whether a message should be forwarded — was set up from the traces of data obtained from the EmailValet. Table 1 shows the amount of data obtained for each user, and the proportion of email that each user requested forwarded.

To create the data that we passed on to our learning methods, whenever words were extracted, they were tokenized by converting them into all lower case, with all punctuations,

[3] We selected five after informal studies showed that results were not strongly affected by this value, as long as it was over one.

[4] If the scores were equal, we drop down to the next score that is different, since we can obviously not put a threshold to distinguish between instances that have the same score.

Table 1: Dataset properties

User	Size	% of messages forwarded in full set	first $\frac{2}{3}$	last $\frac{1}{3}$
AD	1730	13.70%	11.43%	18.22%
HH	5651	17.02%	16.67%	17.72%
SM	1526	10.41%	10.69%	9.86%

Table 2: Vocabulary sizes used in the experiments for text categorization methods

User	Naive Bayes	TFIDF	Pr-TFIDF
AD	500	4000	4000
HH	500	2000	1000
SM	1000	2000	4000

numerics, and special characters removed. Stemming (performing a morphological analysis of each word and only maintaining its root word) and stop-list removal (ignoring words with little information content, such as "the") were also performed. Email addresses were tokenized to create multiple tokens — one for the whole email address, and one for each lexical item in the address (names, username, machine name, domain name, etc.). Thus, for example, "`Sofus Macskassy <sofmac@cs.rutgers.edu>`" became the tokens "`Sofus Macskassy sofmac@cs.rutgers.edu sofmac cs.rutgers.edu rutgers.edu`").

Each learning method has its own requirement about how data should be represented. The Rainbow methods require that a message be represented as a bag of words. Ripper requires that data be a set of attribute-value pairs over a fixed set of attributes, where an attribute may take on a set of values. Winnow requires an (open-ended) set of binary features, where, when present, each makes a prediction concerning the label of an example.

We begin by describing the data representation for Ripper, since it is the easiest to describe. For Ripper, 10 features were used. The Body feature contains all words found in the text body of the message. The Subject feature contains all words found in the subject of the message. The From and To features contain the tokens from email addresses in the From: and To: fields of the message, respectively, and the ToCc feature contains the tokens from the email addresses in the union of the To: and Cc: fields. The Length feature contains the length of the body in bytes. The Date feature contains the date of the month. (We do not include a feature for month, since all the data come from less than a years worth of data.) The Dayname feature includes the day of the week that a message was sent, and the Day feature is the day in numerical form, 0–6 for Sun–Sat. The Daytype feature represents whether a day is a weekday or on the weekend.

Rainbow effectively requires that an example be a single bag of words. To achieve this all words in the body are included, as are all words in the subject with the token "subject" prepended (to distinguish words in the subject from words in the body). Similarly, all tokens from the From: and To: fields are included, with the tokens "from" and "to" prepended to each as appropriate. All tokens from the unions of the To: and Cc: fields are also included with the token "tocc" prepended. The day of the week and weekday or weekend are also included, with suitable prepended tokens to distinguish them from when those words occur elsewhere in the message. For the length and time of day (represented in minutes since midnight) bins were created, with a different token included if a message fell into that bin. For length there were bins for whether a message was smaller than 0.5, 1, 2, 4, 8, and 16 kilobytes, and a similar set for whether the length was larger than each of these values. Similarly, for the time of day we created 48 bins, one for each half-hour mark, half of them representing whether the time was before a given mark, and a second set for whether it was after a given mark. Finally, the lastcommand, lastmailfrom, and lastmailto features were also split into 26 bins each, representing adhoc cutoff points ranging from less (or more) than a minute to less (or more) than a day.

In all experiments using the Rainbow systems we performed feature selection, wherein only some top evaluated set of words are kept and used for learning and classification — results have shown that vocabulary size can often have a significant impact on learning [Lewis, 1994]. In each case a range of vocabulary sizes were used, finding whichever yielded the highest break-even point when trained on the first $\frac{2}{3}$ of the training set and then evaluated on the last $\frac{1}{3}$ of the training set. The vocabulary sizes for each dataset and method is given in Table 2.

Finally, for Winnow, each of the "words" that could potentially be used in Rainbow were considered an expert. However, one of the advantages of Winnow is that it is able to learn from data with many experts, so in addition to these experts we added experts that correspond to pairs of "base-level" features. More specifically, for Winnow we used an expert for all combinations of any of the time-of-day bins with the dayname, daytype, length-bin, subject, and from values; any of the from values with any of the to, tocc, and subject values; any of the to values with any of the to, tocc, and subject values; any of the tocc values with any of the tocc and subject values; and any of the subject values with any other subject values. Thus, for example, there were experts for all pairs of words that occur in a Subject: field, for any pair of users in the To: or Cc: fields, and for any pair of a user in the from and word in the subject.

4 Results

In this section, we present the results of our experiments using each of the five learning methods on each dataset. Table 3 shows the precision/recall break-even points of the five learning methods for each dataset.

The graphs allow certain conclusions to be made about the use of these five methods. First, a very simple baseline approach is to forward all email to a user. This method yields 100% recall, but a precision that equals the proportion of messages that should be forwarded in the data (see Table 1). In all cases, for all methods, this default figure is met or, most often, surpassed, demonstrating that even the less-successful methods still achieve some modicum of success on this task.

The graphs show that for each user there are learning methods that can enable the EmailValet forward a small number of messages all of which should have been forwarded, that is, for cases when 100% precision is desired, with small, but nonzero recall. Unfortunately, however, there was no consistency

Table 3: Break-even points for all methods

User	Naive Bayes	TFIDF	Pr-TFIDF	Ripper	Winnow
AD	41.71	36.02	39.81	36.5	**43.36**
HH	46.94	30.02	46.34	**53.42**	47.15
SM	45.55	39.61	**47.53**	37.56	27.28

Table 4: Results of removing body and date information

	# Wins		# Losses		
Features Used	Clear	General	Clear	General	# Unclear
+body+date	1	2	0	0	12
+body-date	0	2	0	0	13
-body+date	1	1	0	1	12
-body-date	0	1	2	2	10

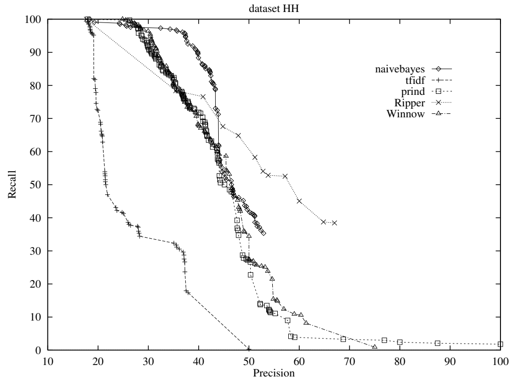

Figure 2: Recall vs. Precision for HH

as to which learning method was able to achieve this.

One subtle point of these experiments is that the learning method has access to the full body of all messages, whereas the user is making decisions based only on the headers. We justify this choice by the fact that the email recipient typically has a tremendous amount of background knowledge that can be brought to bear in interpreting the headers of a message, whereas the EmailValet does not. The hope was that the body, since it is accessible at the server, where the EmailValet reside, can make up for some of this absent knowledge.

A second issue concerns the use of date and time information in the learning process. We would ideally like the learning algorithm to know where a user is — at home, work, in a car, etc. However, this is not available to the EmailValet, but we conjectured that date and time information may be a helpful surrogate for such information about the message's *context*, as opposed to its content. For example, a person is typically at work certain times of day and days of the week.

To understand the contribution these two sources of information played in the classification process we performed three additional runs for each user with each learning method, one with the body features removed, one with date and time information removed, and one with both removed. All other features were kept unchanged. Each case generates its own precision/recall curve, yielding a total of 15 graphs — one graph for each learner and each user, each containing four precision/recall curves, one for each of the combinations of including or excluding the body and date information. Due to space limitations, we are only able to present summaries of these results.

Table 4 shows some summary statistics of these runs. Each row corresponds to a different choice of whether body or date information is removed, and reports the number of times the resulting curve for a method and user surpasses the other curves for that method and user using the other combinations. In each case the number of times the curve was above all others, either clearly or most of the time, clearly or generally below the other curves, or the relative superiority was ambiguous, is shown. In none of the cases does including the body harm learning, while in three cases does excluding the body improve results, and in five of the cases does it clearly harm learning.

Finally, we observe that in the total of 60 curves we generated across these experiments, in seven of the cases did the default of forwarding all messages exceed the performance of a learning method, supporting our similar observation made earlier. Further, in one third of the runs (22 of the 60), we were able to attain a classifier with 100% precision, with varying degrees of recall (at worst, only one message was forwarded), which is roughly comparable to the 27% (4 out of 15) achieved in the graphs presented earlier when both body and date were included. Unfortunately, here, too, there was no consistency to the learning method that achieved this performance.

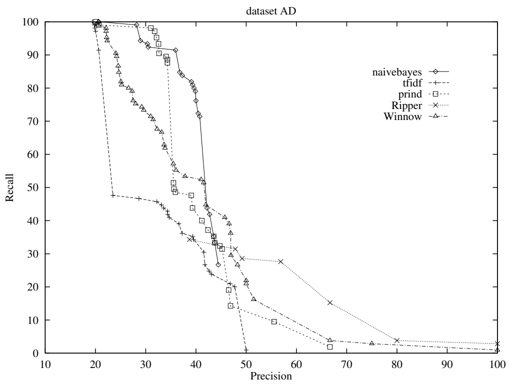

Figure 1: Recall vs. Precision for AD

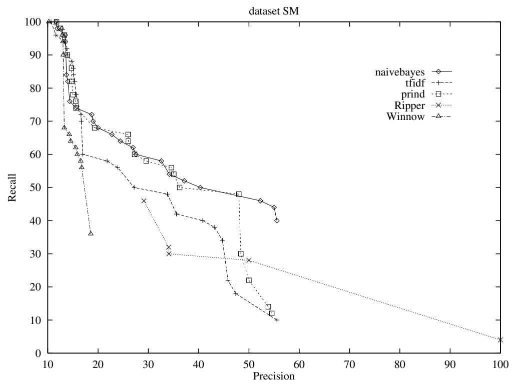

Figure 3: Recall vs. Precision for SM

5 Prospects for the Future

This paper has described the use of machine learning and information retrieval methods to predict whether to forward email to a user's wireless email device. Our results, based on actual use of our EmailValet system, show that, while no single method tested is superior to all others, we are able to achieve modest break-even points with most methods, suggesting further research in this area is warranted.

We also found that including the body of a message — information available at the server, but not to the user deciding whether to a message based on its headers — tends to improve the results of learning. These results are also shown to be substantial improvements over the default of sending all messages to a user.

In the short term, we would like to understand whether other features already discernable by the EmailValet system, can further improve performance. In particular, there are many features about the context of an email message, such as when you last read a message from its sender, whether it is part of an ongoing email discussion, or whether you recently sent email to this person, that may be helpful in deciding how to handle a message. We would also like to be able to order pending email messages effectively, so that if the person has not read email for a while, when the messages do appear they are presented in a prioritized fashion.

On a longer time frame, we would like to explore the use of the EmailValet across a range of wired and wireless platforms and with a wider range of information sources. For example, forwarding a message should depend on what devices are available to a user at a point in time – if the user is in front of a workstation email probably need not be sent to a pager. Ideally, we would like an EmailValet to have access to information such as a user's calendar or physical location (as should become possible in the future via GPS systems on wireless devices). This could make it possible for the EmailValet to know to forward a message from the person to whom you are walking across campus for a scheduled meeting if the message might be canceling the meeting. We would also like to expand the range of interaction modalities that a user can have with the EmailValet — eye tracking (which is becoming possible for desktop workstations) would make it possible to identify the portions of a message that a user had read, and voice interaction also provides the opportunity for finer-grained interaction between the user and EmailValet. Similarly, permitting a wider range of interaction between the user and EmailValet — such as via natural-language dialog — could provide a valuable resource to an EmailValet, such as making it possible to ask the user questions about his or her decisions.

Finally, although the EmailValet concerns email, the very same architecture would allow a user to interact with an agent that can access the Web or other sources of information from a range of platforms. In this light, the EmailValet represents a very first step in our overall vision of software "valets" that more generally serve our heterogeneous information needs across a range of interaction devices.

Acknowledgments

This work was in part funded by the Rutgers Distributed Laboratory for Digital Libraries (RDLDL) and the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). We thank our many colleagues in the Rutgers Machine Learning Research Group and at WINLAB for their feedback on this work. Equipment for this work was provided by BellSouth Wireless Data.

References

[Blum *et al.*, 1991] Avrim Blum, L. Hellerstein, and Nicholas Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 157–166, Santa Cruz, California, 1991. Morgan Kaufman.

[Blum, 1992] Avrim Blum. Learning boolean function in an infinite attribute space. *Machine Learning*, 3:373–386, 1992.

[Blum, 1997] Avrim Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 1997.

[Cohen, 1995] William Weston Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995.

[Cohen, 1996] William Weston Cohen. Learning trees and rules with set–valued features. In *AAAI96*, 1996.

[Fuhr, 1989] N. Fuhr. Models for retrieval with probabilistic indexing. *Information Processing and Management*, 25(1):55–72, 1989.

[Joachims, 1997] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

[Lewis, 1994] D. Lewis. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, Las Vegas, Nevada, April 1994.

[Littlestone, 1988] Nicholas Littlestone. Learning quickly when irrelevant features abound: A new linear–threshold algorithm. *Machine Learning*, 2:285–318, 1988.

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

[Mitchell, 1997] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Rocchio, 1971] J. Rocchio. Relevance feedback in information retrieval. In Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, pages 313–323. Prentice–Hall, 1971.

[Salton, 1991] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–979, 1991.