An Email Assistant that Learns to Suggest Reusable Replies

William R. Hewlett
University of California, Los Angeles
billyh@ucla.edu

Michael Freed
SRI International
333 Ravenswood Avenue
Menlo Park, CA, USA 94025-3493

Abstract

Reusing previously authored email message text in new messages is an especially useful way to reduce workload for people such as IT administrators and event coordinators who tend to make repetitive and complex replies. Previous approaches to helping users find reusable message text have emphasized automation, minimizing the need for user interaction but requiring extensive training on examples of text reuse before achieving high performance. Our work focuses on extending this kind of assistance to handle "spiky" patterns of reuse in which the user receives numerous similar queries over a limited time window, possibly related to some transient event. In this case, the system needs to learn rapidly to be useful. We have implemented an approach in which rapid improvement in performance at suggesting reusable text is achieved by learning from limited user feedback. The paper describes the implemented system, learning approach and an experiment to assess the impact of learning with a small number of training examples.

Keywords

Email, Information Retrieval, Machine Learning

Introduction

Reading, writing and managing email represents a large and increasing source of workload for knowledge workers at every level in nearly every industry (Balter 1998). The RADAR project (Freed *et al.* 2008) encompasses a range of research efforts aimed at reducing email-related workload to a more manageable level. This paper describes research on reducing the effort required to author email reply messages in cases where previously generated message text can be wholly or party reused.

Cutting and pasting old email text into new messages is a useful technique for email users generally, but it is especially valuable for people such as IT administrators and event coordinators who often need to repeat themselves and whose messages are likely to contain information that is complex or hard to recall. There are several ways to help such individuals with improved email technology. For example, an email client might allow users to mark "favorite" messages they are likely to reuse and provide menu access to these messages, analogous to "bookmarking" a site in a web browser. However, this simple approach becomes impractical if there are many potentially reusable messages. Approaches for handling a large set of messages vary in effectiveness in different conditions. Consider three typical use cases:

1. One shot. The user receives a message and recalls having once authored something relevant. The challenge is then to find that message in their SENT mail folder.

2. Routine. The user receives a message that entails a reply they've made many times before e.g. an IT administrator telling someone how to reset their password.

3. Spike. The user receives a message related to a recent, ongoing or upcoming event that provokes numerous senders to make similar inquiries over a limited time period.

These cases present different cost and benefit considerations for the user and suggest correspondingly different ways to assist them. In the One Shot condition, search capabilities available in most current email clients usually provide an adequate solution. Although it may take time to find the right search terms, especially if the send date, recipient and text content are not well remembered, this effort is only done once.

In the Routine case however, the need for repeated use of the same content means that the time cost of search is multiplied and can easily become onerous. Existing approaches that address this problem include the eResponder (Carmel, Shtalhaim, & Soffer 2000) system, which relies on nearest neighbor techniques, and the work of Bickel and Scheffer (Bickel & Scheffer 2004), which clusters sets of email queries and returns responses typical of those sets.

As with routine replies, the Spike reply case is repetitive; automation to help a user avoid periodically searching for the same reusable content can provide substantial benefit. However, there are several important differences arising from the limited time window in which a given message is likely to be reused. First, the user needs to start receiving useful reply suggestions with only a minimal amount of training. The prolonged training interval for matchers typical of the routine case is inappropriate. Similarly, the ef-

fort required to craft templates or FAQ content is unlikely to pay off since the number of times the material is likely to be reused is relatively small. Spikes in novel message content often correspond to spikes in email-related workload, so handling this case can be especially beneficial.

The remainder of the paper illustrates our approach with a walkthough of the system followed by a review of basic techniques used in most response composition assistance approaches. We then explain the novel learning algorithm used to accelerate learning and discuss a series of experiments on a help-desk email corpus in which our learning technique significantly improves our results. Finally, we discuss the current strengths and weaknesses of this approach in the context of future work.

Approach

For the user's perspective, interaction with the response composition assistance capability begins as they read an email message that needs a response and decide that some previously authored response to an earlier message is likely to contain helpfully reusable text (see example Figure 1). The user then presses a button marked "Recommend Reply", causing the system to generate a relevance-ranked list of messages previously authored by the user. The highest ranked messages are then presented in abbreviated form in a set of text boxes placed to the right of the message compose field (Figure 2). The user can examine messages in detail by placing the cursor over the text box. Clicking the box selects the message, pasting it into the compose field where the user can edit and send Figure 3.

The system is given a query stimulus email, a question to which the user has replied to before. The user asks the system to find the previously authored response to a similar question. The system has access to the set of stimulus/response pairs that are the user's previous received messages and replies to those messages. To find a response/answer email to a given question/stimulus email, we attempt to find the previous stimulus email that is closest to the query stimulus email, and return the response associated with that previous stimulus. We use the most common approach (detailed in Salton and McGill's Introduction to Information Retrieval (Salton & Mcgill 1986)) to this problem, which is a *bag of words* combined with nearest neighbor matching.

All problems of this type, including ours, involve a common set of setup steps. First, we stem the emails, removing common suffixes such as "ing", and remove stop words, which words like "the" which are so commonly used that they obscure differences between documents. The system then converts each message into a *bag of words* – i.e. a vector of word counts. Word order information is not retained. For example, the message "Jim thanked Sue" would have the same vector representation as "Sue thanked Jim". From this point on, the goal of finding a match to the original email is achieved by finding the "closest" email vector in the sent mail archive to the query email vector. As is common for this type of problem, we performed a Term Frequency/Inverse Document Frequency (TF/IDF) conversion, so that words that appear frequently in the corpus receive less weight in the vector. Finally, we perform a 8-Nearest Neighbor search, to find the eight closest emails to the target email, where the "distance" of the nearest neighbor search is the cosine difference between the query stimulus email and each of the previous stimulus emails. Formally, if the query stimulus is vector $\overrightarrow{q}$ and a each pre-existing stimulus email is a vector $\overrightarrow{s}$, we search for the pre-existing stimulus email such that $\overrightarrow{q} \cdot \overrightarrow{q}$ is smallest. This comparison, common in the information retrieval literature, is used instead of the euclidian distance between the vectors because it normalizes over the length of the vector. We chose to find the eight closest vectors (rather than 3 or 12 closest), because it fit on a webpage comfortably and preliminary user feedback suggested that a user was comfortable choosing from approximately that many different choices.

As described so far, this work is most similar to that of Carmel, Shtalhaim, and Soffer, whose eResponder system (Carmel, Shtalhaim, & Soffer 2000) took questions from a user and attempted to match those questions against previous questions by comparing TF/IDF vectors that represented the questions. Our work distinguishes itself from such earlier attempts by also incorporating learning. Additionally, we feel our experiments more clearly show the benefit of both the TF/IDF comparisons as well as our own learning component.

The nearest neighbor algorithm described above provides adequate results in some cases, but machine learning is needed to achieve high levels of performance overall. We avoid burdening the user with irksome training of our system; we incorporate our training in the user's normal work patterns. Computational efficiency is a concern; we want the training to be virtually instantaneous, so that it does not affect the workflow of the user.

Our machine learning approach derives from Crammer's work on Ultraconservative Online Algorithms for Multiclass Problems (Crammer & Singer 2001). "Online" algorithms update their training as new data arrives, which is useful because we can train on each decision that the user makes. "Ultraconservative" algorithms only update training examples that are ranked as high as a correct result. For example, if we are training such a system on our problem, we show the user a ranked list of eight possible emails that are good responses to the query email. If the user picks one of them, we change only the chosen email and the emails that our system had ranked above it. If the user chooses an email not on that list, we only update the chosen email and the eight presented choices. This limits the amount of computation that we need to perform. Additionally, we can quickly calculate an exact result for the quadratic optimization problem at the heart of Crammer's equations, whereas other classification techniques, such as Support Vector Machines, normally require a more expensive calculation process such as gradient descent.

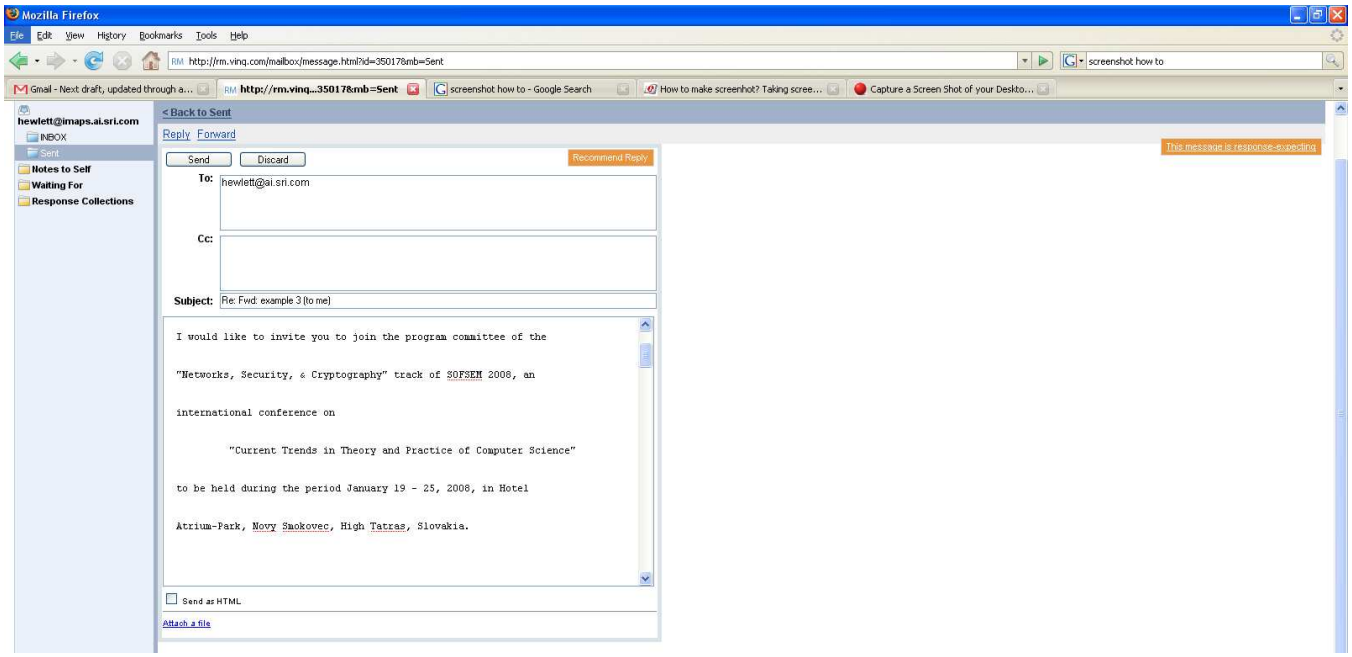

Figure 1: Initial Question

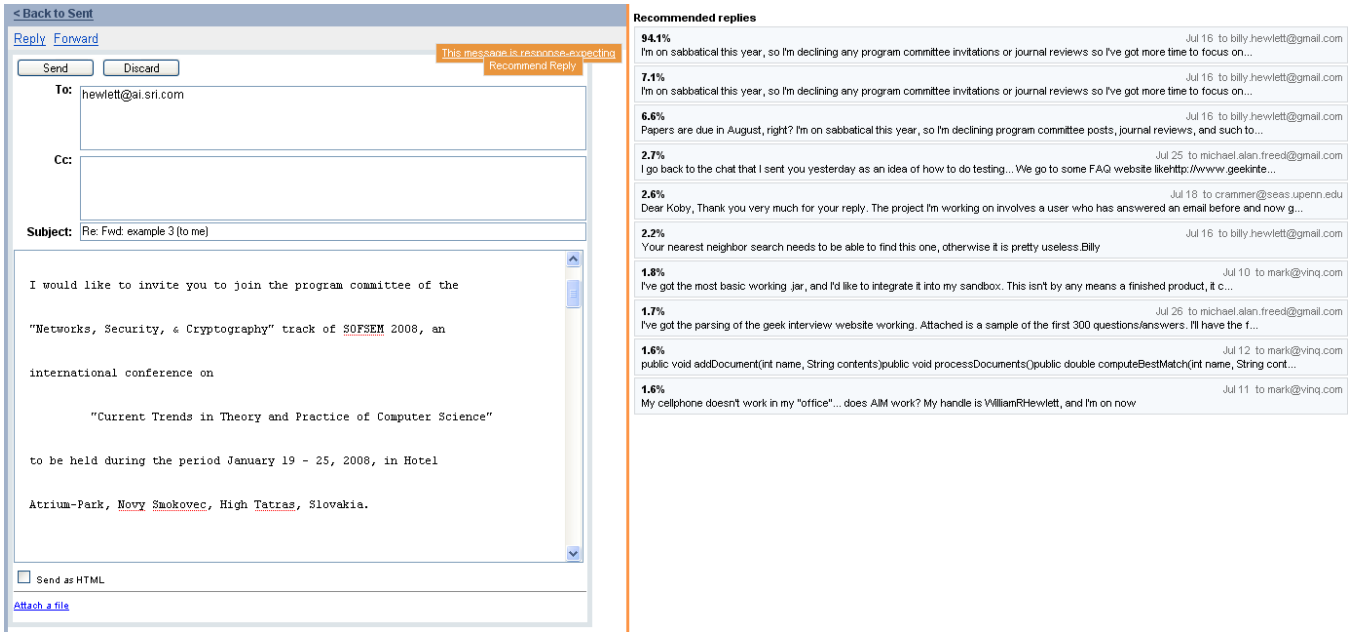

Figure 2: Recommended replies appear on the right

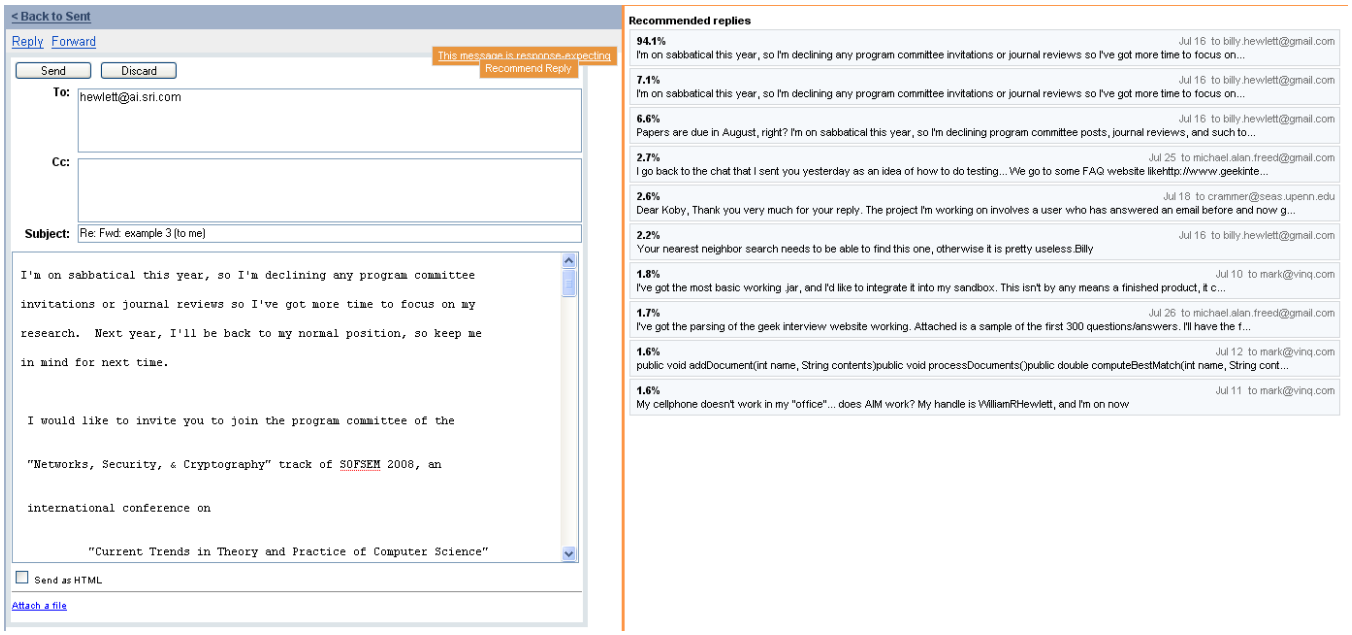

Figure 3: The selected reply is pasted into the compose window

Algorithm

For machine learning, we use a variation of Crammer's Margin Infused Relaxed Algorithm (MIRA) (Crammer & Singer 2001). Given a stimulus email, the user chooses a response to go with it. Learning is only performed on the response that the user chooses and responses that the system labeled as better than that response. Unlike Crammer's method, we limit the number of email messages that we train on. Because of our user interface design, where the user chooses an email response from eight different choices, we only rank the eight closest stimulus matches to the query stimulus. Additionally, if the user chooses a response outside of the eight choices, we can only train on our eight ranked messages (and the user chosen message) because otherwise we might unintentionally downgrade a message that was also a match in addition to the user chosen message. In effect, when the user chooses a message outside the original eight, we only know that the eight were incorrect and the outside message was correct, and we gain no information about the rest of the stimulus messages. The change to training only on the top eight messages change drastically improves our computational performance because the learning is no longer affected by the number of messages in the dataset.

The following is the nearest neighbors comparison algorithm (common to many approaches):

1 Convert all existing stimulus messages and the query stimulus message into TF/IDF vectors

2 Find the eight nearest neighbors to the query message

3 Display the eight responses associated to the stimulus messages to the user

After the user chooses a response from the eight displayed messages (or chooses a response outside of the eight displayed messages), we want to train that response and every one of the displayed responses ranked above it. This is the modified MIRA, the machine learning portion of our algorithm:

4 Given one correct response and some number of ranked incorrect responses, find the stimulus emails associated with those responses.

5 Retrieve the vector associated with the query stimulus. Call this vector $\overrightarrow{x}$.

6 Find the vectors associated with those stimulus emails to be changed and order them from highest ranked to lowest ranked, with the correct email vector after the lowest ranked incorrect email vector.

Let $\overrightarrow{m_r}$ be the rth such vector. Let k be the number of these vectors (i.e. $\overrightarrow{m_k}$ is the correct vector). Let $a = \|\overrightarrow{x}\|$. Let $b_r = \overrightarrow{m_r} \cdot \overrightarrow{x}$.

Now we must find $\overrightarrow{\tau}$ that solves the minimization problem:

$$\frac{1}{2}a\sum_{r}^{k}\tau_r^2 + \sum_{r}^{k}(b_r \times \tau_r) \tag{1}$$

Subject to:

$$\tau_k = 1 \tag{2}$$

$$\sum_{r}^{k}\tau_r = 0 \tag{3}$$

$$\forall(r < k)\tau_r < 0 \tag{4}$$

7 Update each vector by letting $\overrightarrow{m_r} = \overrightarrow{m_r} + \tau_r \times \overrightarrow{x}$

Note another difference between our approach and Crammer's. We force the coefficient of the correct result (τ_k) to be one. This allows us to analytically solve the minimization

problem. First, we take the partial derivative of Equation (1) with respect to each τ_r and set it to 0. This is the minimum of the unconstrained problem because a is always positive.

$$a \times \tau_r + b_r = 0 \tag{5}$$

or

$$\tau_r = -\frac{b_r}{a} \tag{6}$$

This is the point which minimizes Equation (1), but we need to find the point within the constraints that is closest to this point. First we set $\tau_k = 1$, find $\sum_r^k \tau_r$, divide this sum by k-1, and subtract the result from every $\tau_r | r < k$. This is the closest point in the plane created by Constraint (2) and Constraint (3) to the minimum of the unconstrained minimization problem.

$$\tau_r | (r < k) = -\frac{b_r}{a} - \frac{\sum_r^k -\frac{b_r}{a}}{k-1} \tag{7}$$

$$\tau_k = 1 \tag{8}$$

This enforces that $\sum_r^k \tau_r = 0$. This new $\overrightarrow{\tau}$ is the minimum of Equation (1) that satisfies Constraint (2) and Constraint (3). Next, we have to ensure Constraint (4), that $\forall (r < k), \tau_r < 0$. Values of τ_r are in one of four sets:

$$\tau_r | (r = k) \tag{9}$$

$$\tau_r | (r < k \;\; \tau_r > 0) \tag{10}$$

$$\tau_r | (r < k \;\; \tau_r = 0) \tag{11}$$

$$\tau_r | (r < k \;\; \tau_r < 0) \tag{12}$$

We need to find the closest point to the overall minimum where no τ_r is in Set (12), since τ_r values in Set (12) violate Constraint (4). We employ the following iterative procedure: We find $S = \sum (\tau_r | \tau_r \in$ Set (12)). Set every τ_r from Set (12) to 0. Let q equal the size of Set (10). If $q = 0$ then we are done and $\overrightarrow{\tau} = 0$ is the optimal point. Otherwise we add $\frac{S}{q}$ to each τ_r from Set (10). At this point, each τ_r that was in Set (12) has been moved to Set (11), however there might be some τ_r that have moved from Set (10) to Set (12). If the new Set (12) has any members, we repeat this process. Eventually, there will be no τ_r in Set (12). Note that there are a maximum of eight iterations of this process since there are a maximum of 9 different τ_r vectors.

Now we have an exact solution to the MIRA equation.

Intuitively, the modified MIRA algorithm works by moving the correct stimulus email "towards" the query, while moving incorrect stimulus that were ranked above the correct stimulus "away" from the query. The vectors representing the trained stimulus emails are altered as the result of this training. For example, suppose that the user receives a new email message and requests automatic composition assistance. Furthermore, suppose that the nearest neighbors approach finds the incorrect match for this query as in Figure 4. The user chooses the correct match from the list and this match moves towards the query, while close incorrect results move away from the query as in Figure 5. Finally, a second query email arrives, and because of the training the correct match is made without user intervention as in Figure 6.

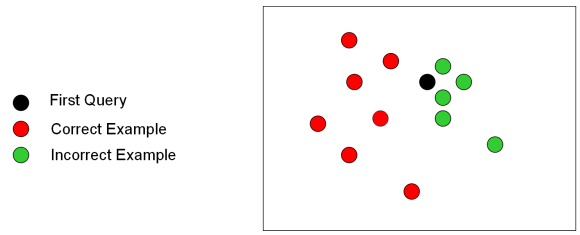

Figure 4: A new query is made

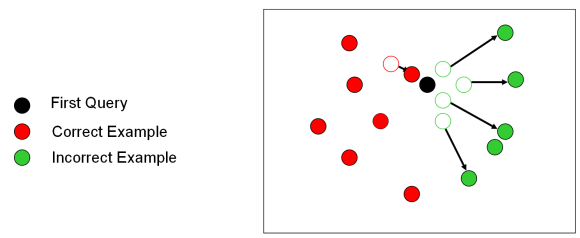

Figure 5: The user chooses the closest correct response to this query, training the system

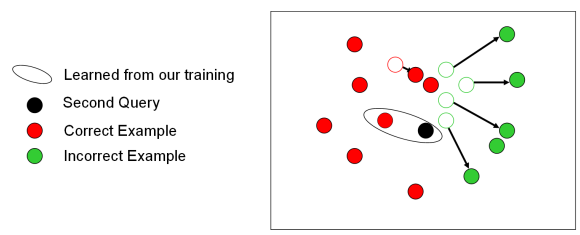

Figure 6: A second query is made. Because of the training, this second query matches a correct example

This learning is extremely fast, and if there are k different terms in the lexicon then the learning has a computational complexity of $O(k)$. The most expensive operation involves taking the dot product of the vectors of the query and each of the ranked responses, and in fact the first training example has a computational complexity of $O(\text{max number of terms in a reply})$. As more training is performed, terms are added to each reply, so the eventual complexity becomes $O(k)$.

The computational complexity of the 8-nearest neighbors search in our implementation is currently $O(k \times n)$, where n is the number of stimulus/response email pairs in the users corpus, however there are methods of reducing this to $O(k \times \log n)$ such as Yianilos' work (Yianilos 1993) on improving the efficiency of nearest neighbors search.

Experimental Results

To evaluate our approach, we used a corpus of 30,000 emails sent to and from a Hewlett Packard (HP) helpdesk. Of these, we selected a sub-corpus of conversations that involved a single stimulus and single response email. There were 8604 such stimulus/response email pairs.

The next step was to identify a subset of the stimuli (query emails received by an HP employee) that could be used to test performance and, for each, identify correct replies the system might suggest. For simplicity in this in initial evaluation, we took a simple and very conservative approach. Given two stimulus-response pairs (S1, R1) and (S2, R2) from the reduced HP corpus, we treat R1 as a valid response for S2 if the *bag of words* representations for R1 and R2 were identical. Our results almost certainly understate performance since potential reply suggestions (R1) that differ in trivial ways from the actual reply (R2) are treated as incorrect. Similarly, we do not distinguish cases where semantically different queries happen to use the same reply. For example, replies advising the sender to "check that all their drivers are up to date" that might be made in response to a broad range of problem reports. Again, the effect on our results is to understate actual performance. In general, the system will correctly treat semantically different queries as separate. However, this will be treated as a failure to suggest a good reply according to our evaluation criterion. To strengthen this evaluation framework, we plan to enlist subject matter experts to label semantically equivalent queries and replies as part of future work.

Given a set of emails pairs with equivalent replies (i.e. where R[i]'s have identical *bag of words* representations), we consider the stimulus emails that elicited those responses. Ideally, a response composition system would be able to match one stimulus from that set to another stimulus from that set so that when queried it would be able to produce the correct response. There were 36 such response groups, consisting of a total of 475 response messages.

For every same-response group, we took a stimulus from that group and ran our nearest neighbors search over all 8604 stimulus emails. Then we counted the percentage of such trials where a stimulus email from the matching group appeared in the top eight. In 231 out of 475, or 49% of the cases, a matching stimulus email appeared in the top eight. Another way of saying this is that if a user had received a stimulus email which should elicit a response that they had already composed and they asked for assistance, 49% of the time the response composition assistant found that response.

For our next experiment, we added the learning component to calculate how much a single training example could improve our results. This experiment is similar to the previous experiment, except that for each same-response group we picked a single stimulus email and associated it with the closest stimulus email from the set.

For example, if the response for the group was "*This department only handles English language questions, please contact...*", we would pick an example stimulus from this group and run our nearest neighbor matcher on it. We would then choose the closest matching stimulus email from the group of questions that elicited this reply. This procedure mimics a user who receives a question in a foreign language, asks the automatic system for a response, and then chooses the highest ranked example of the correct response.

After performing this learning, for 302 of 475, or 64% of the emails, we found an example response in the top eight. A single training example was enough to improve our results from 49% to 60%. See Figure 7 for these results.

There is another interesting feature of our experimental results. Larger groups of identical email responses had much higher similarity between their associated stimulus emails than smaller groups of identical email responses. Our training was much more effective on these larger groups as well. Of the 475 emails in groups, 344 belonged to one of the 10 largest groups. Each of these groups had more than 15 members. Before training on the larger groups, 65% of the correct stimulus emails appeared in the top eight, whereas after one training example these groups 86% of the correct stimulus emails appeared in the top eight. Our performance on the small groups was poor, going from 7% without training to 9% with it. See Figure 7.

We believe the reason for this disparity is simple. Larger groups of responses give a higher chance of similarity coverage. For example, imagine that each email consisted of only three letters, such as (A, Q, Y) or (T, R, H). Furthermore, imagine a group of stimulus emails that only used the letters A, B, and C. The chance that any two stimulus emails from this group having exactly the same A, B, and C count would be low, but if we had enough examples from it, we it would be easy to match an incoming email to the rest of the group. If we consider this phenomenon in the context of our use cases from the introduction, TF/IDF matching and our learning algorithm work well when confronted with a large number of emails that require the same answer, and poorly when matching single emails against each other.

Recall that our matching groups consisted of stimulus emails which all elicited the exact same response. If the criteria for group membership was instead stimulus messages where the same response email would be appropriate, the groups would be much larger. The fact that we do well on larger groups suggests that our results would be even better in practice than in our experiments.

Most work on email response composition (e.g. Bickel and Scheffer (Bickel & Scheffer 2004)) uses the Information Retrieval measures *precision* and *recall* to measure how ef-

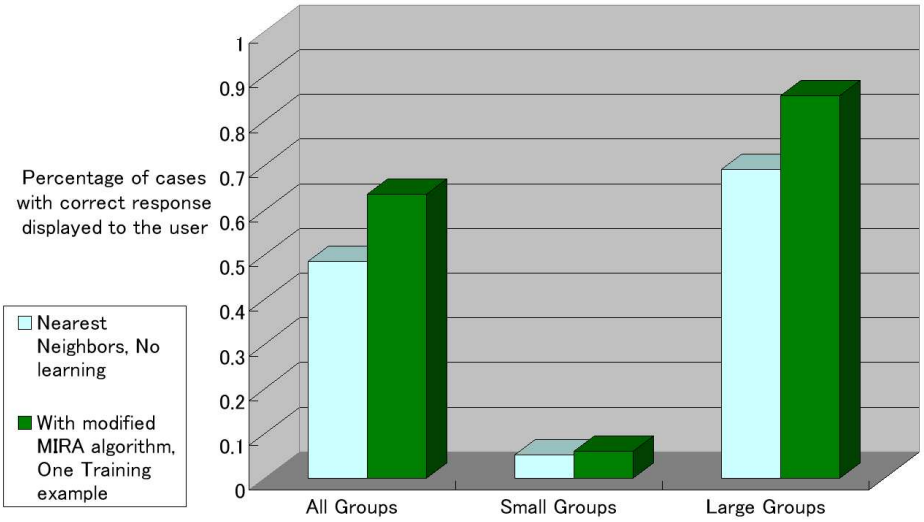

Figure 7: Comparing Nearest Neighbors with modified MIRA, after one training example

fective their email answering system is. What percentage of all messages can a response composition system answer, and of those, what percentage are answered correctly? Under these performance measures our algorithm performs poorly, because it doesn't typically attempt to answer all email. For example, our recall for our experiment would be 291 correctly answered emails over 8604 emails that needed answering, which is extremely low. On the other hand, we make the assumption that the user would only ask our system to answer questions for which they had already authored a response. Furthermore, in our experiment we limit ourselves to the 475 questions where we know for sure that a response has been authored, because answers to these questions exactly match other responses that the user has authored. We are trying to solve a distinct subproblem of the larger automatic composition task, which is to retrieve answers that a user has already given when the user believes that such an answer exists. Our experiments are significantly different from previous work because they assume that a user has self-selected emails to answer with our system. We don't consider the question of whether it is appropriate to generate a response to an email, we assume that our system will be invoked when a response to a stimulus already exists. The traditional measures of *precision* and *recall* are less useful for measuring the value of an interactive system.

Future Work

Future work will extend the system, improve its performance and provide a more roust evaluation framework. Developing a more extensive and informative evaluation framework for examining new technical approaches to this problem would provide a more accurate prediction of learning impact and overall performance in new domains. In particular, we will engage subject matter experts to annotate the data for semantic equivalence, allowing a much better estimation of overall system performance and some improvement in estimated learning performance. Additionally, the corpus used in this work was very useful, but a corpus that incorporated more varied data, rather than just emails in the help-desk domain could be enlightening.

Other efforts, currently ongoing, are aimed at providing more extensive assistance and user workload reduction in formulating responses. In particular, future versions of the system will learn response templates based on observations of user editing behavior, an important step in assisting the user to modify old email for new uses. Another approach under consideration is to make use of smaller units of test, as demonstrated by Marom and Zukerman(Marom & Zukerman 2007). Their approach attempts to construct email results sentence by sentence, rather than by returning a whole message. This extends the cases that an email client can answer, but it seems unclear how one might involve a user in the email construction process. A system that could scale to the amount of available training would be desirable. Marom, Zukerman, and Japkowicz citeAustralians2007 describe a meta-level strategy which chooses the most effective algorithm for answering email from a series of possible algorithms. Their work has promise as a bridge between our work and previous work in this field.I tr

Acknowledgments

I would like to thank Hewlett Packard for making the help-desk email corpus available for research purposes. I would also like to thank SRI for giving me the opportunity to work on this project during my internship. Special thanks to Eric Gimon, who provided mathematical advice, and to Mark Torrance of VINQ LLC, who provided assistance in integrating this system into a general email client. This system was built on top of the Minorthird classification toolkit, which was written by Professor William Cohen at Carnegie

Mellon University. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010.

References

Balter, O. 1998. Electronic mail in a working context.

Bickel, S., and Scheffer, T. 2004. Learning from message pairs for automatic email answering. In Boulicaut, J.-F.; Esposito, F.; Giannotti, F.; and Pedreschi, D., eds., *ECML*, volume 3201 of *Lecture Notes in Computer Science*, 87–98. Springer.

Carmel, D.; Shtalhaim, M.; and Soffer, A. 2000. eresponder: Electronic question responder. In Etzion, O., and Scheuermann, P., eds., *Cooperative Information Systems, 7th International Conference, CoopIS 2000, Eilat, Israel, September 6-8, 2000, Proceedings*, volume 1901 of *Lecture Notes in Computer Science*, 150–161. Springer.

Crammer, K., and Singer, Y. 2001. Ultraconservative online algorithms for multiclass problems. In *14th Annual Conference on Computational Learning Theory, COLT 2001 and 5th European Conference on Computational Learning Theory, EuroCOLT 2001, Amsterdam, The Netherlands, July 2001, Proceedings*, volume 2111, 99–115. Springer, Berlin.

Freed, M.; Carbonell, J.; Gordon, G.; Hayes, J.; Myers, B.; Siewiorek, D.; Smith, S.; Steinfeld, A.; and Tomasic, A. 2008. Radar: A personal assistant that learns to reduce email overload. In *AAAI*. AAAI Press.

Marom, Y., and Zukerman, I. 2007. Evaluation of a large-scale email response system. In *5th Workshop on Knowledge and Reasoning in Practical Dialogue Systems, IJCAI 2007*.

Salton, G., and Mcgill, M. J. 1986. *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc.

Yianilos, P. N. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, 311–321. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.