Towards Hierarchical Email Recipient Prediction

Jacob Bartel
Department of Computer Science
University of North Carolina
Chapel Hill, NC, USA
bartel@cs.unc.edu

Prasun Dewan
Department of Computer Science
University of North Carolina
Chapel Hill, NC, USA
dewan@cs.unc.edu

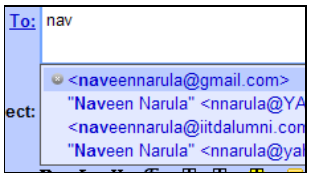
***Abstract*—Previous email prediction algorithms generate individual predictions based on the past groupings of recipients or the contents of past emails. Our work builds on this research by (a) introducing new algorithms for extending and combining previous techniques and generating hierarchical recipient predictions and (b) comparing the previous algorithms with each other and the new algorithms. We used standard metrics and developed new metrics to measure three kinds of user effort: scanning predictions, selecting predictions, and manually entering recipients. The new metrics are based on a new abstract model of recipient prediction that applies to existing schemes and the new ones developed by us. Our evaluations, based on the Enron mail database and the Gmail user-interface for recipient prediction, show that (a) content is less effective than groups, (b) the combination of content and groups is less effective than groups alone, and (c) hierarchical recipient prediction reduces user effort.**

Keywords- recommender systems; email; privacy

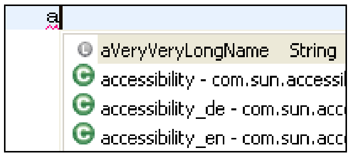
I. INTRODUCTION

A number of GUIs require the entry of text, which can typically be broken up into tokens – string sequences delimited by whitespace and special separator characters. To aid the entry of such text, some of these interfaces provide token completion and prediction, illustrated in Figure 1. Token completion recommends a set of choices that complete the current token based on the prefix entered by the user. In Figures 1(a) and (b), the Gmail and Eclipse user-interfaces provide lists of recommendations based on the token prefixes 'nav' and 'a', respectively. Token prediction, on the other hand, recommends one or more future tokens based on the tokens entered so far. In Figure 1(c) and (d), the Gmail and Eclipse user-interfaces recommend tokens which represent additional recipients to whom a message should be addressed and alternative valid method calls, respectively.

In this paper, we consider a special case of token prediction illustrated in Figure 1(c): prediction of email recipients. Recipient prediction has several potential advantages. It saves the user effort when entering long email addresses (ids/names). Moreover, it allows the user to find recipients whose email addresses they cannot recall, which is particularly likely with listserv groups, long email addresses, or large collections of email addresses. In this case, the sender knows who should receive the message but cannot remember their addresses. Recipient prediction can also allow the sender to be reminded of forgotten recipients who should receive the email [1]. This is an important use of such predictions, as missing a recipient can be costly for senders, missed receivers, and others. Finally it can prevent leakage of information to unintended recipients, and, thus, forms a new kind of tool for ensuring privacy. The importance of identifying forgotten recipients and information leakage was recently illustrated in a class the second author taught in Fall 2011. Students in the class sent the instructor emails that should have also gone to his teaching assistants, resulting in unnecessary forwards or missed requests. Perhaps even more alarming, solutions to five of the twelve assignments, and reports of personal issues, were mailed accidentally by different students to the whole class rather than to the instructors, because tthe students confused the class-help listserv with the class listserv. These are not isolated problems. A CMU study by Carvalho et al of a recipient prediction email tool found that at least 9.27% of emails did not include a desired recipient [2]. With accurate and effective recipient predictions, users could be reminded of the correct recipient and, thus, not fall prey to such issues. These potential benefits could have a significant impact given the popularity of email and recent research advocating the use of email as the primary collaboration user interface [3].

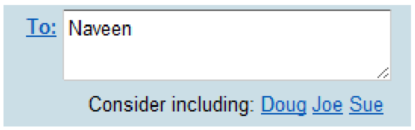

To: nav
<naveennarula@gmail.com>
"Naveen Narula" <nnarula@YA
<naveennarula@iitdalumni.con
"Naveen Narula" <nnarula@yal
ect:

(a) Gmail

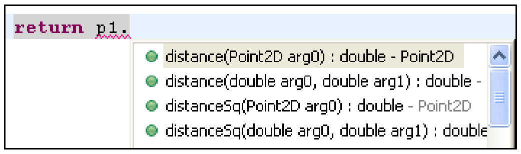

a
aVeryVeryLongName String
accessibility - com.sun.accessi
accessibility_de - com.sun.acc
accessibility_en - com.sun.acc

(b) Eclipse

Token Completion: Recommending current token


To: Naveen
Consider including: Doug Joe Sue

(c) Gmail


```
return p1.
distance(Point2D arg0) : double - Point2D
distance(double arg0, double arg1) : double -
distanceSq(Point2D arg0) : double - Point2D
distanceSq(double arg0, double arg1) : double
```

(d) Eclipse

Figure 1. Token Completion vs. Prediction

Past work that has directly addressed the email recipient prediction problem can be separated into two distinct categories: content and group based prediction. These approaches are email applications of the more general user-interface idea of using the history of user interaction to automate the entry of user commands [4]. In content-based predictions, individual recipients are predicted based on the text in past messages. The intuition behind this approach is that similar documents will be shared with the same user(s). For example, in a university all resumes may be shared with members of the hiring committee and all promotion material with members of the promotion committee. Group-based predictions, on the other hand, associate possible recipients into groups based on how they were associated in past sent and received messages. Thus, if a message was sent to all members of the hiring committee, then, if in the future a message is addressed to some subset of this committee, a subset of the remaining members will predicted

These schemes are special cases in the more general area of automatically clustering users into social groups based on characteristics they share. In particular, they are related to schemes that predict named contact-lists [5-7]. The difference between recipient and contact-list prediction is that the former predicts a group of users who should receive a specific message based on the properties of the message, while the latter identifies multiple groups of users based on general relationships among the users such as whether two users are "friends" in a social network [5, 7] or if they have sent a certain number of messages to each other [6]. As our examples show, both individual addresses and contact-lists can be components of email recipient predictions.

Past works have compared different schemes for email recipient prediction, but only in limited ways. Specifically, they have not compared group and content-based schemes, and different comparisons have not used consistent data sets or metrics. Our work seeks to offer a more complete comparison of past approaches and to build on this work in other ways by answering several new, interesting questions:

1. Dimensionalized design space: Can we identify a design space of prediction algorithms that includes existing schemes? Such a space can lead to more effective email prediction by including previously unexplored subspaces. Moreover, its dimensions can be used to succinctly compare and contrast existing and new algorithms, leading to a better understanding of them.

2. Hierarchical prediction: Previous (content and group-based) approaches predict a flat list of recipients, requiring individual recipients to be selected, one at a time. Is it possible to predict a hierarchical tree of recipients to allow users to atomically select groups of recipients? Hierarchical prediction can potentially reduce the overall user effort in selecting predictions. Each time a prediction is made, users must make some effort to determine whether the generated prediction is correct and either accept the prediction or perform some rejection action, which usually consists of manually entering some other recipient. Accepting a group rather than an individual reduces the number of times a user has to process and accept or reject a prediction. On the other hand, based on how it is implemented, it can potentially also increase the overall effort as predictions of groups are riskier than those of individuals, and, thus, can lead to more rejections.

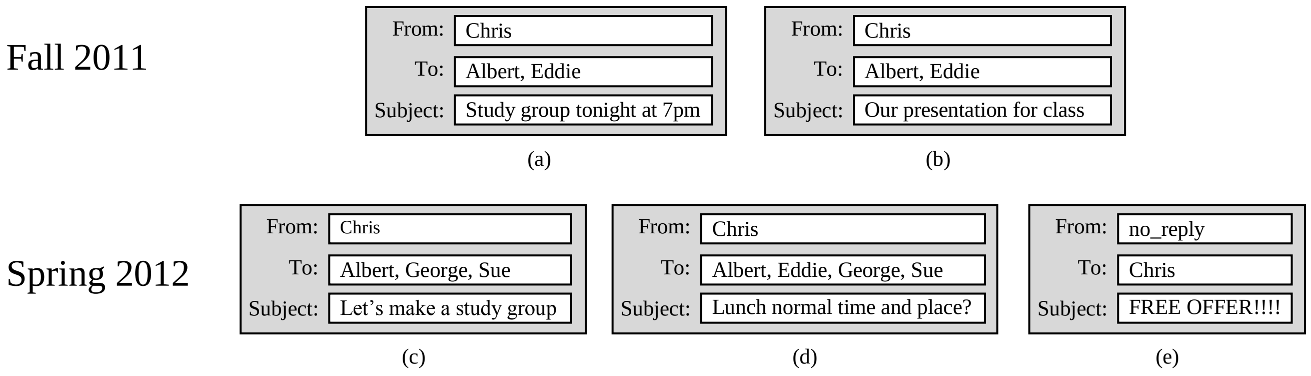
3. Comparison: Hierarchical prediction adds an important new dimension to the design space mentioned above. How do various points in the prediction design space compare with each other? In particular, how do group-based and content-based approaches compare with each other and with a hybrid approach that combines the two; and how does individual prediction compare with the hierarchical prediction?

4. Effort metrics and recipient prediction model: What metrics should be used to evaluate the user effort required in the compared techniques? The answer to this question depends on the user-interface provided for suggesting and accepting or rejecting predictions. We do not innovate in the design of this user-interface and instead assume a small extension of the one used in Gmail (Figure 1c) that uses parentheses to group predicted recipients (Figure 6(b)). Such a user-interface requires three kinds of user effort: scanning predictions, selecting predictions, and manually entering recipients. Previous work has used classical metrics to determine the degree of false positives and negatives, which do not directly measure these user efforts. Is it possible to develop new quantitative metrics that address this problem?

We answer these questions in several stages. We start by describing dimensions that describe the existing two algorithms. Next we motivate and define new metrics for evaluating the effectiveness of recipient prediction, which are then used to compare the existing schemes. This comparison is used to motivate a new algorithm for predicting individuals and sets of individuals. The new algorithm is then compared with the existing ones using both our class metrics and our new metrics. The best of the algorithms are then used to further improve recipient prediction by extending any individual recipient prediction to support hierarchical prediction, which is then evaluated. A running example is used to illustrate the similarities and differences among these schemes and motivate them, which is a contribution in its own right.

II. Design Space to Describe Current Schemes

In our model of recipient prediction schemes, individual recipients or groups are associated with some set of past email messages, and properties of these sets are used to determine the likelihood of a correct prediction. This model is illustrated by Figures 2 and 3, which are used as our running example. Figure 2 lists the past messages of an email account belonging to the user Chris, some of which occurred in Fall 2011 and some of which occurred in Spring 2012. The Fall 2011 messages were addressed to the same receivers but had different content. The Spring 2012 messages were addressed to three different groups of recipients, some of which overlap with previous groups.



Fall 2011

(a)
From: Chris
To: Albert, Eddie
Subject: Study group tonight at 7pm

(b)
From: Chris
To: Albert, Eddie
Subject: Our presentation for class

Spring 2012

(c)
From: Chris
To: Albert, George, Sue
Subject: Let's make a study group

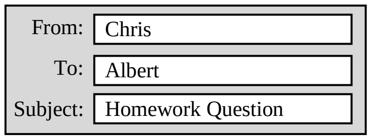
(d)
From: Chris
To: Albert, Eddie, George, Sue
Subject: Lunch normal time and place?

(e)
From: no_reply
To: Chris
Subject: FREE OFFER!!!!

Figure 2. Sample email message history

Figure 3 specifies a message that the user Chris composed in Spring 2012. In this newly composed message, Chris has already addressed Albert, one of the recipients of a previous message, and is now asking for predictions of other possible recipients.

As discussed above, past schemes for email recipient prediction can be grouped into two categories, groups and content. A group-based algorithm bases its predictions on the set of recipients in the current message and in each previous message. Likely predictions are then identified by similarities between the set corresponding to the current message and any set corresponding to a past message or messages. For example, consider messages (a) and (b) in Figure 2, where Chris sent two messages to Albert and Eddie. Since the two users were addressed together in the past, and Albert has already been addressed in Figure 3, Eddie is a likely predicted recipient for the current message. As this example illustrates, group-based prediction looks at both *groups* of users addressed in previous messages and the *seed set* of users addressed so far in the current message, either by manual entry or prediction selection.



From: Chris
To: Albert
Subject: Homework Question

Figure 3. Example message in need of recipient predictions

However, as one can see by looking across all messages in Figure 2, email accounts are not this simple. Groups can overlap, which can lead to users being members of multiple groups at the same time. Therefore, predictions need to be ranked in some form. Such rankings use the SOYLENT [8] idea that (a) the rank of a group is proportional to the strength of the connections between its members; and (b) the connection strength can be derived using various properties of email exchanges. Two such properties used in past work are time and direction. Time captures the change of groups. For example, consider the messages (a) and (c) in Figure 2. The user, Chris, changes classes from semester to semester, and thus his old group of (Albert, Eddie), should be made less important than a new group of (Albert, George, Sue) as time passes.

Direction captures whether the owner of an email account implicitly created a group by sending a message to others, or whether some outside source made this specification by sending the message to the owner. When the owner of the account sends an email to a set of recipients, it is reasonable to assume that those individual recipients have some sort of association. However, if the message was received, it is more difficult to make that assumption as the sent message may have incorrectly been sent to the owner, the message may be spam (Figure 2(e)), or the sender may not be able to receive messages.

The algorithm of Roth et al. [9], implemented in Google's Gmail, combines these four properties (groups, seed, time, and direction). It works in two stages. First it assigns a weight to each group based on these four properties, and then, from these group weights, it determines the individual weight.

Let us consider, first, group weight, which is a function of several individual weights corresponding to the four properties. A direction weight is captured by sent and received constants, whose values are unspecified. A time weight is computed using the standard half-life formula, which makes older messages exponentially less important based on a constant. The impact of the seed is determined using the intersection of the group with the seed. The Roth et al. [9] paper identifies four alternatives for calculating the weight of a group based on these three properties, which define a space of group-based schemes.

(1) Top Score - The seed is ignored, and the group weight is a sum of products of direction and time weights for past messages of the group. (2) Intersection Count - If the intersection is non-empty, then the group weight is calculated as 1, and if not, it is calculated as zero. Thus, in this approach, direction and time are ignored. (3) Intersection Score - If the intersection is non-empty, then the score is the sum of the products of direction and time weights; otherwise, it is zero. (4) Intersection Weighted Score - The group weight is the size of the intersection multiplied by the sum of products of time and direction weights.

The scores of individual recipients were computed by the summing of group scores. An individual, i, has a set of groups G, of which it is a member. The score of this individual is calculated as

$$\sum_{g \in G} score(g).$$

Thus, as we see above, group-based prediction, as implemented by Roth et al, in fact uses four different properties of email: group, seed, time, and direction. One property it does not use is the content of the email. Carvalho and Cohen [1] use a combination of content and direction of the email. As mentioned before, the intuition behind this scheme is that people tend to receive or send email of "similar" content. We illustrate this intuition with messages (c) and (d) in Figure 2. Chris has two groups he emails. One is for a class study group, and one is for a lunch group. If it is a particularly difficult class, he may be sending and receiving emails a few times a week, and, in the case of the lunch group, he eats on a regular basis. Therefore time and direction offer no help in differentiating between the two groups. However, if one were to examine the content of the two emails, one could find that these emails are very different from each other.

In order to effectively use content for prediction making, an email prediction scheme must define the similarity between two messages. The algorithm in [1] is based on the TF-IDF (Term Frequency-Inverse Document Frequency) text mining technique [10]. Given a set of documents, TF-IDF first computes a list of words or terms, $t_1...t_n$, that occur in these documents. Then, given a specific document, it calculates a weight vector, $w_1..w_n$, where w_i is the weight of term t_i in that document. (How exactly the weight is computed is beyond the scope of this paper.) Carvalho and Cohen compute TF-IDF based on all but a small subset of ignored words, which we replicate in our evaluation, described later. Their scheme treats each message as a document, and, thus, associates each email with a weight vector, $w_1..w_n$. As in group-based predictions, each possible prediction is associated with a set of past email messages, E. A weight vector, $W_1..W_n$, is then formed for each prediction by summing together the weight vectors for all messages in E.

After a new message is composed, there exists a weight vector for each possible recipient and the newly composed message. The likelihood of a particular prediction, p, is then calculated as the cosine of the angle between the vector for that prediction, v_p, and the vector for the new message, v_m. The cosine can be computed with the following equation, where $v_p \bullet v_m$ is the dot product between the two vectors:

$$score(p) = \frac{v_p \bullet v_m}{|v_p||v_m|}$$

Based on this equation, each prediction has a score, just as it did in the group-based case, where a higher score indicates a more likely prediction.

Thus, we see two different ways of predicting recipients. While there has been study within the categories of group and content-based predictions, no work has been done to compare the two spaces with each other. Such a comparison requires appropriate evaluation metrics.

III. METRICS

As mentioned above, past work has measured the effectiveness of predictions through the classic metrics of precision (P) and recall (R). However, these two metrics only determine the correctness of predictions - they do not directly measure how a user's effort is reduced. This is due to two fundamental differences between classic prediction systems and recipient prediction. In the latter, the prediction step is followed by a user acceptance or rejection of a prediction, which, in turn, implies that the user knows if a prediction is correct or not. Thus, the cost of a false positive (predicting an unintended recipient) incurs the additional effort required to reject it, and not a wrong user conclusion such as a wrong medical diagnosis. Second, as we see in the group-based scheme, predictions can be made incrementally, based on past user actions.

A false negative (not predicting an intended recipient) can indeed lead to a wrong conclusion, as a user may forget an intended recipient because the system has predicted that no more recipients are necessary. Thus, the classic metrics remain relevant. However, additional metrics are needed to more directly measure the reduction in user effort.

In order to better measure the effort required during the prediction process, we first had to develop a model of how predictions are generated, accepted, and rejected. Predictions are generated as a list of items, where an item may be an individual recipient or a (potentially hierarchical) group. For a non-empty top-level list, a user may select an individual or a group, or reject all predictions in the list. If a list is empty or the user has rejected all predictions in a list, a user must manually enter some recipient and then ask for a new prediction list. The maximum number of leaf nodes in a predicted list is kept constant in each prediction.

The previously defined metrics can be used to measure certain aspects of this process. An empty list corresponds to a false-negative, because there are recipients still left to address, but the prediction algorithm can find no predictions. By measuring the ratio of non-empty lists (positives) to total requests for lists (candidates), recall determines the degree of false negatives. By computing the ratio of lists that contain a correct prediction (correct positive) to the total number of non-empty lists generated, precision determines the degree of false positives. The higher the recall/precision, the lower the degree of false negatives/positives. We denote precision and recall with the variables P and R, respectively.

We introduce a new metric, average acceptance size, which we denote with the variable A. It measures the average number of individuals chosen when a user accepts some prediction from a particular list. In an algorithm that only predicts individuals, this average acceptance size is 1. However, if groups are made available as predictions, then this average acceptance size can be larger, because a single user action can accept multiple recipients.

A higher value of A reduces the number of clicks (selections) made by users to select predicted lists. However, making a click is not the only way users exert effort in an email system supporting recipient prediction. They must also scan the recipients in the prediction lists and manually enter recipients. To determine the cost of these three kinds of user efforts in an email that is addressed to X recipients, we use the variables s, c, and m, respectively, to denote the number of non-empty lists scanned by the user, the number of clicks made, and the number of recipients entered manually.

The values of the variables P, R, A, X, s, c, and m are related to each other by the following system of equations:

(1) $A \cdot c + m = X$

(2) $R \cdot (c + m) = s$

(3) $P \cdot s = c$

Equation (1) says that the total number of recipients, X, is the sum of the number, m, manually entered, and the number, $A \cdot c$, selected through c clicks. Equation (2) evaluates the scanning cost, s, by determining the number of times a user must scan non-empty lists that are generated in the process of addressing the email. Each time a user accepts a prediction (by clicking) or manually enters a recipient, the user has implicitly requested a list prediction beforehand. Thus, the total number of such requests is c + m. Only R of these requests are non-empty, and thus only R · (c + m) of them must be scanned. This scanning can further vary based on the size of the list where larger lists may take significantly more effort to scan. However, we strictly restrict our lists to contain at most 4 individuals, and thus we assume our scanning costs to be constant for our different list sizes. Finally, equation (3) determines the total number of clicks, c, which corresponds to the total number of correct positives, which, by definition is the number of non-empty lists, s, multiplied by the precision, P.

These equations 1, 2, and 3 can be solved to compute the three effort values s, c, and m:

(4) $s = \frac{R}{PR(A-1)+1} X$

(5) $c = \frac{PR}{PR(A-1)+1} X$

(6) $m = \frac{1-PR}{PR(A-1)+1} X$

We can divide these three numbers by X to lead to the normalized fractional values, S, C, and M, respectively. Through these values, we now have measures of how often certain types of effort are exerted. An algorithm requires less user effort than another if it leads to lower (average) values of S, C, and M for the same email data set. When two algorithms are partially ordered by these three metrics, we make the following assumption: clicking a correct prediction takes the least amount of work and manually entering a recipient takes the most amount of work. Our justification for the assumption is the following: When selecting a correct prediction, we assume the user has already scanned a list and knows which prediction is correct. When clicking, a user only needs to determine where to click and perform the clicking action. On the other hand, manually entering a recipient requires the user to remember the intended recipient and the spelling of the recipient's name or email address.

IV. Content vs. Group

The existing and new metrics allow objective, quantitative comparisons between different points in the recipient prediction scheme design space. The previous metrics have, in fact, been used to compare some of these points. Carvalho et al. compared content and direction-based predictions with time-based predictions using precision and found that content and direction yielded the best results [1]. Similarly, Roth et al. compared group, time, and direction-based predictions with time and direction-based ones through use of their Top Score variation and found the combination of all three properties fared better both in terms of precision and recall [9].

Thus, previous work has not compared group and content-based schemes, and different comparisons have not used consistent data sets or metrics. Moreover, as mentioned earlier, the metrics they have used do not address certain types of user effort. Therefore, we expand on this work by using both our new metrics and classic metrics to perform direct comparisons between content and group-based prediction schemes.

In order to effectively compare various prediction schemes, we also had to select appropriate values for the half-life constant and sent vs. received mail constants. As Roth et al did not release the values they used, we varied them in the following manner: For time, we experimented with half-life values of one hour, one day, one week, four weeks, six months, one year, and two years. We found the best values vary depending on which variation of the algorithm is used. For

Table 1. Results of comparison of past algorithms

		Half Life	Relative Sent Importance	P	R	C	M
Content		N/A	N/A	0.066	0.980	0.065	0.935
Top Score	**Best Precision**	One Week	0.25	0.105	1.000	0.105	0.895
	Best Recall	Four Weeks	0.25	0.103	1.000	0.103	0.897
Intersection Count	**Best Precision**	One Hour	0.25	0.131	0.959	0.126	0.874
	Best Recall	One Hour	0.25	0.131	0.959	0.126	0.874
Intersection Score	**Best Precision**	One Week	0.25	0.190	0.958	0.182	0.818
	Best Recall	Four Weeks	0.25	0.186	0.958	0.178	0.822
Intersection Weighted Score	**Best Precision**	One Year	0.5	0.284	0.958	0.273	0.727
	Best Recall	Four Weeks	2.0	0.111	0.995	0.110	0.890

direction, we defined a constant, relative_sent_importance, which is defined as sent_importance/received_importance; where sent_importance is the constant applied for sent messages and received_importance is the constant for received messages. Our relative_sent_importance was varied to 0.25, 0.5, 1.0, 2.0, and 4.0, allowing testing with sent messages held in higher importance in some cases, and received messages in others.

We generated predictions using both the content-based scheme and all four of the group-based variations. For the dataset, we used the version of the Enron email database retrieved from [11], which contained 127 accounts in total. The content-based scheme of Carvalho and Cohen also used Enron accounts, while the group-based scheme of Roth et al. used Gmail accounts available to Google. For each account, we ordered the messages by time and removed any non-email based messages, such as those marked as calendar entries for outside applications. We then used the first 90% as the set of past messages for an account. The final 10% of the messages for an account were used to model prediction making.

For each message, we assumed a seed value of 2 (the sender and one other intended recipient) at the start of all predictions. Each time a prediction list was generated, we assumed the user would select the first correct individual in a generated predicted list. If no such individual existed, then it was assumed that the user would manually enter the first recipient as ordered originally in the email message. We assume the ordering of recipients was first TO, then CC, and finally BCC. No such specification of the ordering exists in past work, so we have no source of comparison for this assumption.

As discussed above, because the past work of [9] and its implementation in the Gmail product restricts list sizes to 4 individuals, we do the same. This is an effort to keep our results comparable to those of past work, as well as avoiding generating lists that are too difficult to parse.

Using this methodology, we arrived at the results displayed in Table 1. There is no S value presented in the table due to the fact that there are only individual predictions, which means that A = 1 in all cases. Because of this value of A, the value of S reduces to R, making S superfluous in the table.

Table 1 shows that as M decreases, C increases, which falls in line with the definitions of C and M. If some individual was not manually entered, then it must have been selected as a correct prediction, which means an additional click had to take place. It also shows that content-based prediction performs worse than group-based prediction in that there is at least one group-based prediction algorithm that performs better with respect to both precision and recall. Additionally, when sorting by recall or precision, no group-based prediction has as high of an M value as that of content-based predictions, which indicates that content-based predictions require more effort for manual entries of recipients.

We also attempted predictions made by combining both groups and content. This combined attempt used the previously computed separate content and group scores. The two scores were each scaled using adjustable weights and then summed together to form a cumulative score. In some cases, we also scaled the content vectors according to the half-life values and the relative_sent_importance as used in group scores to include time and direction with content. Regardless of whether the scores included time or direction or how scores or vectors were scaled, the combined group and content-based predictions underperformed those of group-based predictions.

In our best case of all these combinations of groups and content, we had a recall of 1.00 and a precision of .269, which imply clicking and manual entry values of .269 and .731, respectively. This does improve over groups with respect to the clicking and scanning metrics, but underperforms with respect to manual entries. As stated above, we assume manual entry to be the metric which requires the most effort on the part of the user, which, in this case, implies combined group and content predictions are less effective than group predictions. Despite the comparatively low effectiveness compared to group-based predictions, these values are better than content alone. This in the very least implies that the TF-IDF approach benefits from including groups in prediction making, but as a general approach, if predictions are made using content, TF-IDF is not a comparatively effective approach.

Thus, our results show that (a) content is less effective than groups and (b) the combination of content and groups is less effective than groups alone. It is possible that the Gmail implementation of group-based prediction uses more optimal parameters, which would make our conclusion even stronger.

V. Intersections vs. Subsets

Because of the low effectiveness of content, we focused on the use of groups, not content, in recipient predictions. Our goal was to offer new ways to generate predictions that are more effective according to both classical metrics and our own newly developed metrics. Both our results and those of Roth et al. show that it is important to seek improvements to their scheme.

Table 2. Results of subset based use of seeds

Algorithm	Half Life	Relative Sent Importance	P	R	C	M
Subset Count	One Week	0.25	0.837	0.314	0.262	0.738
Subset Score	One Week	1	0.849	0.314	0.267	0.733
Subset Weighted Score	One Week	0.25	0.853	0.314	0.268	0.732

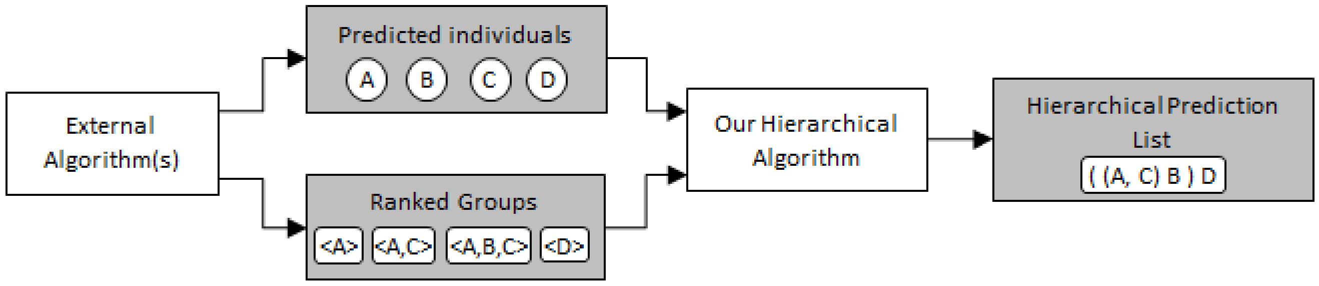


Figure 4. Input and output of our hierarchical algorithm

One improvement, which we present here, is motivated by applying group-based prediction (with our weights) to the second author's email account. He found that, regardless of seed, every message had the same group of predicted recipients. Upon further investigation, he discovered that the reason this group was constantly predicted was because he had sent them messages frequently over multiple years, including some messages sent very recently. When making predictions, since the first author's email address was always a part of the seed, this group always intersected with the seed and, thus, outranked more appropriate predictions due to its high recency and frequency of contact.

One way to counter this situation is to exclude he sender from the seed or the ranked groups. (Roth et al. do not indicate whether they include the sender in their seeds.) While that approach would work in this specific scenario, it still allows a small intersection to overwhelm a larger one. Therefore, we explored an alternative seed-based approach that does not compute intersections but, instead, looks at subset relationships between the seed and the ranked groups.

In this subset-based approach, it is possible to develop variations that are analogues to the *Intersection Count* and *Intersection Score* variations of the intersection-based approach. The variations are as follows: (1) *Subset Count* – If the seed is a subset of a group, then the score of that group is 1, and if it is not a subset, then the score is 0. Just as with *Intersection Count*, time and direction are ignored in *Subset Count*. (2) *Subset Score* – If the seed is a subset of a group, then that group's score is the direction weight multiplied by the time weight.

It is more difficult to create a subset-based analogue of the *Intersection Weighted Score* variation. If we simply multiply the time and direction product by the size of the subset, all values would remain the same relative to each other because the size of the seed never changes during a single round of prediction making. However, what is important is the relation of the size of the subset to the size of the group. Consider a seed of size 2 that is a subset of two groups whose sizes are 3 and 100. To predict the group of size 100 based solely on the seed value, the algorithm would, in essence, be guessing 98 individuals. However, if the group of size 3 were predicted solely based on the seed value, the algorithm would only be guessing one individual, which ultimately leaves a smaller uncertainty. Therefore we define the *Subset Weighted Score* for a seed and group to be time and direction weights multiplied by |seed|/|group|.

The results of using the subset-based approach to seeds are displayed in Table 2. As in the intersection-based approach, weighing scores gives best results. The best case of *Intersection Weighted Score* had a much higher recall value than the best case of *Subset Weighted Score*, while the reverse was true for precision. With the use of our new metrics, we are able to distinguish the effects such results would have on user effort.

The metrics C and M yield similar values in both cases, indicating that users will have to exert roughly the same amount of effort for clicking predictions and manually entering recipients. However, the metric S, which is equal to recall in the case of individual predictions, is much lower when using a subset-based approach. Thus, our new metrics show that subset-based approaches reduce user effort with respect to scanning prediction lists, and, as a result, these approaches outperform the intersection-based approaches.

VI. HIERARCHICAL PREDICTIONS

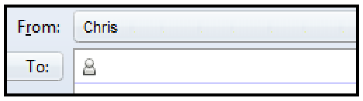
The fact that the subset-based treatment of seed values outperforms that of intersection-based treatment indicates that there is a hierarchical tree of groups in the set of possible predictions. If this is indeed true, users should be able to reduce their click count by selecting not just leaf nodes in the tree but also intermediate nodes. This feature, in turn, requires a scheme for computing the hierarchy and a user-interface for displaying and selecting both leaf and non-leaf nodes in the hierarchy. There are several approaches for doing so – the one we settled on makes few changes to the algorithm and user-interface for individual predictions.

Instead of developing a new scheme from scratch, we create hierarchical prediction lists from individual prediction lists that were generated by some other group-based algorithm external to and, thus composable with, our algorithm. This relationship is illustrated in Figure 4. Our algorithm generates hierarchical prediction list using predicted individuals and ranked groups from some external algorithm(s). In general, a hierarchical prediction list can contain overlapping groups, as in the case of [6]. We constrain the hierarchy to a tree, where a node has a single parent, which allows us to make few changes to the user-interface.

```
object types:
PredictionList: ordered set of predictions
Prediction: {group, //mapped group of individuals
             rank} //order in prediction list
Grouping ISA Prediction:  ordered set of predictions
Individual ISA Prediction: {id} // name of individual

global vars:
 indivList = list of individual predictions of non-hierarchical scheme

functions:
buildHierarchicalPredictionList():
   treeList = new PredictionList // create empty list
   forall p in indivList do addToPredictionList(treeList, p)

addToPredictionList(treeList, new):
   merged ← false
   next  ← new;
   forall old in treeList where old != next do
      if old.group ⊆ next.group  | next.group ⊂ old.group then
         if next.group ⊂ old.group then
            forall other in treeList where other!=old & other != next
               & other.group ⊆ next.group do
            //grouping of all other predictions with group ⊆ other.group
               mergedGrouping ← merge(other, next)
               remove next and other if they were in treeList
               list.add(mergedGrouping)
               next ← mergedGrouping
            endfor
         endif
         merged ← true
         mergedGrouping ← merge(old, next)
         remove old and next if they were in treeList
         list.add(mergedGrouping)
         next ← mergedGrouping
      endif
   endFor
   if !merged then list.add(new)
   elseif treeList.size == 1 & tree_list is within a Grouping then
      //all members of a Grouping were merged into a subgroup
      treeList.members = members of only child
   endif

merge (p1, p2):  //assumed p1 ⊆ p2
   if p2 is Grouping then
      addToPredictionList(p2.members, p1)
      p2.rank ← max(p1.rank, p2.rank)
      return p2
   else //p2 is individual
      g = new Grouping with p1 and p2 in members
      g.group ← p2.group
      g.rank ← max(p1.rank, p2.rank)
      return g
   endif
```

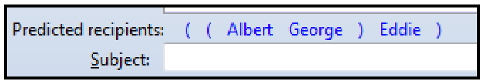
Figure 5. Pseudocode for hierarchical grouping

Our hierarchical algorithm builds a tree out of the individual prediction list, re-ordering the predictions if necessary. We assume parentheses (or some other marker symbol) are used to show the groupings, and that the parentheses do not significantly add to the scanning cost, because our experiments, like those of Roth et al. [9], limit our prediction lists to at most 4 individuals. An example of this interface is show in Figure 6(b), which is a part of a Mozilla Thunderbird extension that we developed.


From: Chris

To:

(a) Addressing recipients


Predicted recipients: ((Albert George) Eddie)

Subject:

(b) Predicting Recipients

Figure 6. Hierarchically Predicted Recipients in a Mozilla Thunderbird Extension

With the assumption of at most 4 individuals, there are at most 3 groupings and, therefore, at most 6 parentheses (2 parentheses per grouping). With this relatively small number of characters added to the prediction list, we assume that our scheme does not add a significant amount of effort with respect to scanning an individual prediction list. However, if prediction lists contained no limit or a much higher limit on the number of individuals, the number of groupings, and thus parentheses, could increase significantly, which could drastically change the scanning costs of a hierarchical prediction list compared to a flat list.

Because our individuals are generated using external algorithms, individuals are selected as they would in previous UIs, by clicking the name of that individual. To select a grouping, the user must click one of the parentheses associated with that grouping.

To illustrate the generation of a hierarchical list, we will use a variation of our running example. Chris has a larger group of friends with whom he has lunch (Figure 2(e)). This group is subdivided into smaller study groups based on who is enrolled in his various classes. For this illustration, we will also assume that an external algorithm finds the list of predicted individuals {Albert, Eddie, George} and finds the following set of groups: <Albert> <Albert, George>, <Albert, Eddie, George >. Our goal is to organize these three nodes into a tree based on the ranked groups.

Figure 5 gives our algorithm for meeting this goal. In this algorithm, individual and hierarchical lists are defined by the type PredictionList, which is a list of objects of type Prediction. A Prediction can be a Grouping or an Individual, and has two fields, group and rank. The former field is the top ranked group of which the prediction is a member/subset, and the latter is the rank of said group. The variable indivList contains Prediction objects for the individuals predicted by the external algorithm. In our example, the indiv list would contain the following Individual objects:

{id: Albert, group: <Albert>, rank: 0}

{id: Eddie, group:<Albert, Eddie, George>, rank: 2}

{id: George, group:<Albert, George>, rank: 1}

The function buildHierarchicalPredictionList() builds a hierarchical list from indivList. This is done by calling the function addToPredictionList(), which adds each member of the original list, indivList, to the hierarchical list. During this process, as each individual is added to the hierarchical list, the function attempts to merge the new individual with the existing members of the hierarchical list. This merging, which occurs because of subset relationships between the group field of the new individual and of existing members, arranges the list into a hierarchy.

Because these merges occur based on subset relationships, a newly added individual may not merge with any existing members due to a lack of such a relationship. In this case, the new individual is added to the end of the hierarchical list. For example, in Figure 7(a), when the first individual, Albert, is added to the hierarchy, there are no other members of the hierarchical list, and thus Albert cannot be merged with an existing member. Thus, the hierarchy will be a single leaf node and the list would be displayed as: Albert.

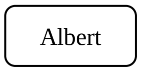
However merges will occur in cases when a subset relationship exists between the group fields. In one such case, the newly added Individual has a group field that is a superset of an existing member's group field. The algorithm will perform a merging by placing both the new Individual and the existing member in a new Grouping and that Grouping replaces the old member in the hierarchy, thus occupying its original position. In our example, in Figure 7(b), Eddie is initially added as a leaf node to the hierarchical list. Then, since his group field, <Albert, Eddie, George>, is a superset of Albert's group field, <Albert>, the two are put in a new Grouping which replaces the Albert node in the hierarchy, resulting in the hierarchy shown in Figure 7(c). The newly created Grouping's group field takes the value of the largest group fields from its members. This hierarchy gives us (Albert, Eddie) as our displayed list at this stage.

Finally, in the last case we consider, the newly added Prediction has a group field that is a strict subset of an existing member's group field. If the existing member is a Grouping, the new Prediction can be added to the existing member. However, if the existing member is an individual, the previous approach of creating a new Grouping containing the old and new Predictions is used. In our example, we must next add George to the hierarchy in Figure 7(d). His group field, <Albert, George>, is a subset of the top level Grouping's group field, <Albert, Eddie, George>, and thus he is added to the existing Grouping in Figure 7(e). This Grouping will retain the same group field, since it was the largest group field of all of its members.

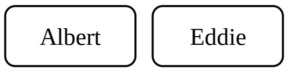
To support multi-level hierarchies, after an initial merging of a new recipient with a Grouping the algorithm recursively attempts to merge him/her with other members of the Grouping. In the example, since George is the newly added Prediction, and his group field is a superset of Albert's group field, a new Grouping containing both Albert and George is formed, the Albert and George nodes are deleted, the new Grouping is placed at Albert's position, leaving us with the hierarchy in Figure 7(f), and the displayed list of:

((Albert, George), Eddie).

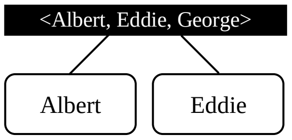
The groupings are ordered by the highest ranking individual contained in the grouping. Albert came before Eddie in the original individuals prediction list, so the grouping (Albert, George), comes before the individual Eddie.



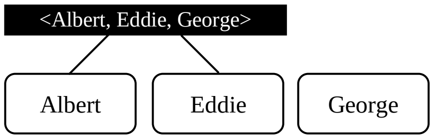
(a) Adding Albert



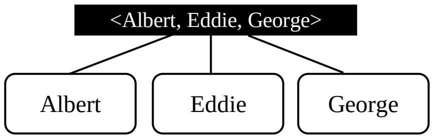
(b) Eddie is the next prediction



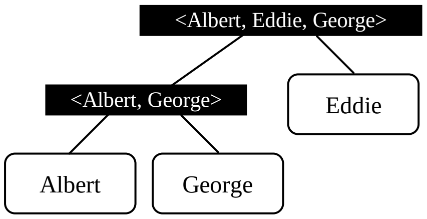
(c) Merging Eddie at the top level



(d) George is the next prediction



(e) Merging George at the top level



(f) Merging George within the hierarchy

Figure 7. Steps of generating the example hierarchical list

To test the effectiveness of these groupings, we composed our algorithm with two best variations of the intersection and subset approach. We ran a similar modeling scheme to the one used in the individual predictions. Our only change was in how we assumed a user would accept predictions. Using the past approach, we assume the user accepted the first correct prediction. In the hierarchical approach, we assume the user will pick the largest grouping that contains all correct predictions, because by doing so, the user is attempting to reduce their work as much as possible.

The results of our testing are detailed in Table 3. P and R values are lower than those seen in purely individual predictions, which is to be expected if the tree-based scheme identified some of the intended groupings. As multiple predictions are accepted at once, such a scheme reduces the number of times a prediction list containing a correct match is generated.

Table 3. Hierarchical Results

	Half Life	Relative Sent Importance	P	R	A	S	C	M
Intersection Score	One Week	0.25	0.13	0.956	1.666	0.883	0.114	0.809
Intersection Weighted Score	One Week	0.25	0.189	0.953	1.75	0.84	0.159	0.722
Subset Score	One Week	0.25	0.74	0.21	1.975	0.183	0.135	0.733
Subset Weighted Score	One Week	0.25	0.748	0.211	1.959	0.184	0.137	0.731

The M value remains approximately the same, which is also to be expected. Because our grouping algorithm is orthogonal to the generation of individual predictions, we still generate empty lists and lists with no correct predictions at the same rate as in the external algorithm.

The S and C values are reduced by a significant amount. Specifically, our S values are reduced by half in the case of subset-based treatment of the seed and the click count is reduced by about half in all cases. This indicates that the user will have to select predictions half as often in all cases and will have to scan prediction lists half as often in the best case, which is a significant reduction in effort.

VII. Conclusion and Future Directions

This paper makes several related contributions. It (a) classifies existing email prediction schemes into content and group-based; (b) defines the new metrics for capturing the user effort required to scan, click, or manually enter a recipient; (c) compares the existing schemes using new and old metrics; (d) identifies parameter values for the group-based schemes that gave the best results. In addition, it describes and evaluates new algorithms that (a) extend content-based prediction with direction and time; (b) combine the extended content-based prediction with group-based prediction; and (c) convert individual prediction lists created by a group-based prediction into a tree in which arbitrary nodes can be selected by the user. Its evaluations lead to several conclusions. (1) Group-based prediction algorithms perform far better than content-based ones. (2) Combining content and group into a single algorithm outperforms the results of content-based predictions, but ultimately fails to achieve better results than the best cases of group-based predictions. (3) An intersection-based treatment of seeds in prediction making performs worse than a subset-based, which implies a hierarchy of groups. (4) Grouping individuals leads to a significant reduction in user effort with respect to scanning lists and clicking correct predictions.

While this paper answers several important questions in recipient prediction, it also leaves numerous other questions unanswered: Can other schemes of content analysis be incorporated to create effective prediction lists? We have not been able to make content schemes work efficiently or effectively but are more hopeful about template-based analysis. Can prediction algorithms take into account the fact that groups grow and shrink? The current algorithms, including ours, create a new group with each membership change. Are there schemes to effectively predict groups previously unseen in any one email message? One approach for addressing the two group-based questions above is to make predictions based on training from multiple accounts rather than a single account, possibly combining the schemes presented here with community detection algorithms.

The Gmail user-interface assumed by our work presents a non-scrollable linear list of at most four items displayed dynamically for each message. There are several other alternative user-interfaces possible, one of which is shown in Figure 1(d). It would be useful to compare the usability of existing and new user-interfaces for token prediction in general and email-recipient prediction in particular. Such work could determine the impact of increasing the size of the recommended list, providing a scrollable list, providing a static message-independent area for displaying and selecting recipients, showing hierarchical lists using a hierarchical display, and integrating token completion and prediction by showing for each completed email address the associated recipient predictions.

By comparing past techniques with both classic and novel metrics and by expanding into new areas and techniques, this paper provides a basis for investigating these intriguing questions.

References

[1] Carvalho, V.R. and W.W. Cohen. Ranking Users for Intelligent Message Addressing. in *Proc. of ECIR*. 2008.

[2] Carvalho, V.R., R. Balasubramanyan, and W.W. Cohen. Information Leaks and Suggestions: A Case Study using Mozilla Thunderbird. in *Proc. of Conference on Email and Anti-Spam*. 2009.

[3] Bellotti, V., et al. FLANNEL: adding computation to electronic mail during transmission. in *Proc. of UIST*. 2002.

[4] Greenberg, S. and I.H. Witten. How users repeat their actions on computers: Principles for design of history mechanisms. in *Proc. of CHI*. 1988.

[5] Bacon, K. and P. Dewan. Mixed-Initiative Friend-List Creation. in *Proc. ECSCW*. 2011.

[6] MacLean, D., et al. Groups without tears: mining social topologies from email. in Proceedings of IUI. 2011.

[7] Friggeri, A., G. Chelius, and E. Fleury., Triangles to Capture Social Cohesion. in Proc. of *The Third IEEE International Conference on Social Computing*. 2011.

[8] Fisher, D. and P. Dourish. Social and Temporal Structures in Everyday Collaboration. in *Proc. CHI*. 2004.

[9] Roth, M., et al. Suggesting Friends Using the Implicit Social Graph. in Proc. KDD. 2010.

[10] Salton, G., E.A Fox, and H., Wu, Extended Boolean Informational Retrieval. Communications of the ACM, 1983. 26(11).

[11] The Electronic Discovery Reference Model, EDRM Enron PST Data Set. http://www.edrm.net/resources/data-sets/enron-data-set-files.