Information Leaks and Suggestions: A Case Study using Mozilla Thunderbird

Vitor R. Carvalho
Microsoft Live Labs
One Microsoft Way
Redmond, WA 98052
vitor@microsoft.com

Ramnath Balasubramanyan
William W. Cohen
Language Technologies Institute
Carnegie Mellon University
{rbalasub,wcohen}@cs.cmu.edu

ABSTRACT

People often make serious mistakes when addressing email messages. One type of costly mistake is an "email leak", i.e., accidentally sending a message to an unintended recipient — a widespread problem that can severely harm individuals and corporations. Another type of addressing error is forgetting to add an intended collaborator as recipient, a likely source of costly misunderstandings and communication delays in large corporations.

To address these problems, various data mining techniques have been proposed recently [3, 4]. In this paper we describe the deployment of some of these techniques in a popular email client (*Mozilla Thunderbird*), and report how users responded to such data mining techniques in their everyday lives. In spite of interface, privacy and speed constraints, results were fairly positive. More than 15% of the users reported that the client prevented real cases of email leaks, and more than 47% of them accepted recommendations provided by the data mining techniques. We then conclude by presenting a few lessons learned from this deployment, and discussing costs and benefits of making these techniques permanent additions to email clients.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Delphi theory

Keywords

Privacy leaks, System deployment

1. INTRODUCTION

The network of contacts maintained via email has been growing steadily over the years. Until relatively recently, email was used primarily for work-related contacts by most people. Now email is used as the primary contact point in many overlapping social circles, including friends, colleagues, neighbors, relatives, as well as social circles associated with various on-line communities. Moreover, the number of people with multiple email addresses (e.g. separate work and personal emails) has grown. All of these changes make it harder for users to choose the right address to send a message they have composed.

A critical issue related to message addressing is how to prevent email information leaks, i.e., when a message is accidentally addressed to non-desired recipients. *Email leaks* are an increasingly common problem that can severely harm individuals and corporations — for instance, a single email leak can potentially cause expensive law suits, brand reputation damage, negotiation setbacks and severe financial losses. Email leaks can be caused by typos, as well as by the aggressive email address auto-completion in some clients, people with similar first and/or last names, accidentally hitting the reply-all button, to name a few reasons. Even though it is not easy to estimate the amount of loss caused by information leaks, such incidents should be avoided at all costs.

Another important email-related problem is forgetting to add an important collaborator or manager as a message recipient (e.g., "Sorry, I forgot to CC you"), which can potentially cause costly misunderstandings, communication delays and missed opportunities. One possible way to prevent this problem is by means of *recipient recommendation*, i.e., intelligently recommending email addresses that are potential recipients for a message under composition given its current contents, its previously specified recipients or a few initial letters of the intended recipient contact. Recommending or suggesting recipients can be a valuable addition to email clients, particularly in large corporations, where negotiations are frequently handled via email and the cost of errors in task management is high.

To address these problems, different machine learning and information retrieval techniques have been advocated in recent literature [3, 4]. Using language patterns extracted from previous email exchange, these techniques create prediction models able to help prevent email leaks, as well as to perform recipient recommendation for messages under composition. That is, for a message being composed, these models predict which email addresses should not have been included as recipient (i.e., addresses accidentally included) and also which other email addresses should be included. The intuition behind these models is that different recipients

CEAS 2009 - Sixth Conference on Email and Anti-Spam July 16-17, 2009, Mountain View, California USA

are typically addressed using different word distributions (or topics).

Although the proposed techniques showed promising results in static experiments on very large email collections [3, 4], many questions are still unanswered. How can these methods be incorporated in an integrated interface? Are the proposed techniques accurate enough to be used in current email clients? Can users notice any difference in quality between the different recommendation algorithms? Can these methods really catch real email leaks? Can we estimate how often email leaks occur? Can these techniques be adopted and benefit a large number of email users?

To address these questions, we built an extension – Cut Once – that adds intelligent recipient recommendation and email leak detection capabilities to Mozilla Thunderbird[1], a popular open source email client. All aspects of the usage of the extension are logged and periodically emailed (if express permission is given by the user) to us, thus allowing us to study the impact of the algorithms on users who use email as a part of their day to day activities.

Based on Cut Once, we designed and evaluated a 4-week long user study that led to very positive results. More than 15% of the human subjects reported that Cut Once prevented real email leaks, and more than 47% of them utilized the provided recipient recommendations. It left an overall positive impression in the large majority of the users, and it was even able to change the way three of the subjects compose emails — instead of the usual address-then-compose, some users started relying on Cut Once to perform a compose-then-address procedure. More than 80% of the subjects would permanently use Cut Once in their email clients in case a few interface/optimization changes are implemented. Overall, the study showed that both leak prediction and recipient recommendation are welcome additions and can be potentially adopted by a very large number of email users.

2. CUT ONCE: A MOZILLA THUNDERBIRD EXTENSION

Selecting an email client in which these data mining techniques could be implemented depended on several factors such as the popularity of email client, whether or not the client is open source, operating system interoperability, the ease with which it could be modified to incorporate new features, and how easily these modifications can be distributed to users. The options considered were *Mozilla Thunderbird*, *GMail*, or a new standalone email client which we would have to develop from scratch.

Developing a new email client had the disadvantage that it would take a long time for it to be used widely, if at all. Moreover, considerable effort would have to be put into engineering efforts which were peripheral to the issue at hand. GMail has the advantage of being widely used especially in the academic community, however the API offered by GMail was inadequate for our needs. Mozilla Thunderbird, on the other hand, is very popular[2], has a well established mechanism to add extensions, and is open source, which makes it an excellent platform to incorporate new features.

Cut Once is a new extension to Mozilla Thunderbird that implements methods to perform recipient recommendation as well as email leak prediction. The extension was primarily written in Javascript, and the user interfaces are specified using a Mozilla specific XML-based file format called XUL. Similar to all other Thunderbird extensions, Cut Once is distributed as an *.xpi* package, which can be easily installed in any Mozilla Thunderbird client using Thunderbird's Extension Manager. A screenshot of Thunderbird's main window after installating Cut Once is displayed in Figure 1. Currently Cut Once can be downloaded from its website: `http://www.cs.cmu.edu/~vitor/cutonce/cutOnce.html`.

2.1 Recommendation Algorithms

In this section we describe the algorithms and baseline models used for the proposed tasks. In all cases, we adopted the following terminology. The symbol *ca* refers to *candidate email address* and *t* refers to *terms* in *documents* or *queries*. A document *doc* refers to *documents* in the training set, i.e., email messages previously sent by the user. A *query q* refers to an email message under composition. Both documents and queries are modeled as distributions over (lowercased) terms found in the "body" and "subject" of the associated email messages.

We also define other useful functions. The number of times a term t occurs in a query q or a document *doc* is, respectively, $n(t, q)$ or $n(t, doc)$. The *recipient function* $Recip(doc)$ returns the set of all recipients of message *doc*. The *association function* $a(doc, ca)$ returns 1 if and only if *ca* is one of the recipients (TO, CC or BCC) of message *doc*, otherwise it returns zero. $D(ca)$ is defined as the set of training documents in which *ca* is a recipient, i.e, $D(ca) = \{doc | a(doc, ca) = 1\}$.

2.1.1 TFIDF Classifier

The email recommendation task can naturally be framed as a multi-class classification problem, with each candidate email *ca* representing a class ranked by some notion of classification confidence. The first baseline method implemented in Cut Once was based on the TFIDF Rocchio algorithm [6, 9]. Essentially, each email address in the address book is represented by a TFIDF centroid vector. This vector is calculated based on the word counts from messages found in the *Sent* folder. When a new message is composed, the TFIDF vector representation[3] of the new message is compared with the stored TFIDF centroid vectors, thus producing a final ranking based on language similarity.

Specifically, for each candidate, a centroid vector-based representation is created:

$$\vec{centroid}(ca) = \frac{1}{|D(ca)|} \sum_{doc \in D(ca)} \vec{tfidf}(doc) \qquad (1)$$

where $\vec{tfidf}(doc)$ is the TFIDF vector representation. The final ranking score for each candidate *ca* is produced by computing the cosine similarity between the centroid vector and the TFIDF representation of the query, i.e., $score(ca, q) = \cos\left(\vec{tfidf}(q), \vec{centroid}(ca)\right)$.

[1]Available at http://www.mozilla.com/thunderbird

[2]It is estimated that Mozilla Thunderbird has between 5 and 10 million active users.

[3]For each term t in document *doc*, the value $tfidf(t) = log(n(t, doc) + 1) log(\frac{|sent\_train|}{DF(t)})$, where $DF(t)$ is the document frequency of t.

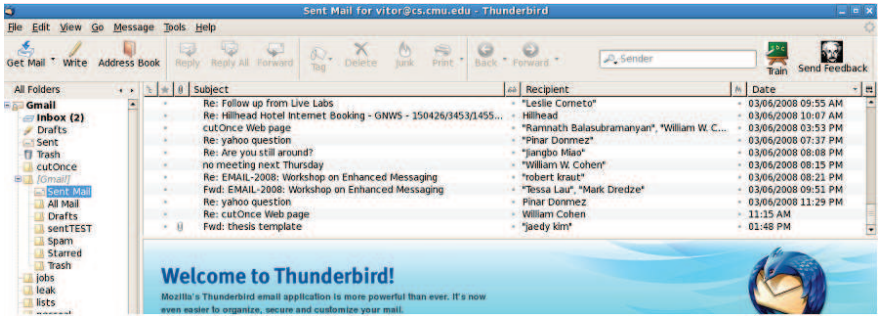

Figure 1: Thunderbird main window after installing Cut Once

2.1.2 K-Nearest Neighbors

We also adapted another multi-class classification algorithm, K-Nearest Neighbors as described by Yang & Liu [12], to these email ranking tasks. Given a query q, the algorithm finds $N(q)$, i.e., the K most similar messages (or neighbors) in the training set. The notion of similarity here is also defined as the cosine distance between the TF-IDF query vector $\vec{tfidf}(q)$ and the TFIDF document vector $\vec{tfidf}(doc)$.

The final ranking is computed as the weighted sum of the query-document similarities (in which ca was a recipient):

$$score(ca, q) = \sum_{doc \in N(q)} a(doc, ca) \cos\left(\vec{tfidf}(q), \vec{tfidf}(doc)\right) \quad (2)$$

Despite the very good results from preliminary experiments [4], the memory and computational demands of this algorithm could not be afforded by most user clients.

2.1.3 Frequency and Recency-based Recommendation

In addition to language-based counts, frequency and recency information can be used as baselines for email prediction tasks. The intuition behind it is that users tend to write more messages to people who have been recently or frequently addressed. The frequency method ranks candidates according to the number of messages in the training set in which they were a recipient. Likewise, the recency method ranks candidates in a similar way, but attributes more weight to recent messages according to the same exponential decay function.

The *Frequency* method ranks candidates according to the number of messages in the training set in which they were a recipient: in other words, for any query q the *Frequency* model will present the following ranking of candidates:

$$Frequency(ca) = \sum_{doc} a(doc, ca) \quad (3)$$

Compared to *Frequency*, the *Recency* model ranks candidates in a similar way, but attributes more weight to recent messages according to an exponential decay function. In other words, for any query q the *Recency* model will present the following ranking:

$$Recency(ca) = \sum_{doc} a(ca, doc) e^{\left(\frac{-timeRank(doc)}{\beta}\right)} \quad (4)$$

where $timeRank(doc)$ is the rank of doc in a chronologically sorted list of messages in *sent_train* (the most recent message will have rank 1).

2.1.4 Aggregating Baseline Methods with Data Fusion

The ranks obtained by the recency, frequency and TFIDF methods can be combined using data fusion techniques based on the Mean Reciprocal Ranks (MRR) of the baseline rankings [5, 8]. Previous work using the Enron email corpus [4] showed that using MRR to combine different baselines can provide better performance than one single baseline in isolation.

In Cut Once we implemented an MRR-based ranking method combining the TFIDF, Frequency and Recency baselines described above. The MRR combination score can be expressed as: $MRR(ca) = \frac{\alpha}{recency\_rank(ca)} + \frac{\beta}{frequency\_rank(ca)} + \frac{\gamma}{tfidf\_rank(ca)}$. That is, the final aggregated ranking score of a email recipient candidate ca is a function of the ranking of the same recipient obtained by the base methods (TFIDF, frequency and recency). Based on preliminary experiments from a previous reference [4], we set γ to 2.0 by default and the decay coefficient β in the Recency baseline to 100 (thus emphasizing the last 100 messages sent).

2.1.5 Random Selection of Baseline Algorithm

One of the main questions we would like to answer is whether the differences in overall ranking performance between the baseline recommendation methods are actually noticeable to email users. To investigate this, we designed Cut Once with a controlled variable affecting the ranking method used by a particular user. That is, the extension uses the TFIDF baseline for roughly half the users (chosen randomly at installation time), and the MRR baseline for the other half of the users.

From an implementation perspective, the same algorithm can be applied to both leak prediction and recipient recommendation. For leak prediction, the algorithm ranks the already specified recipient's addresses (in the TO, CC or BCC fields), where the least likely one is the one presented as the most likely leak. For the recipient recommendation, all addresses in the user's address book are ranked, and the top ones are presented to the user as a ranked list.

2.2 Training

Because the algorithms were implemented in Javascript, scalability and computation time are significant factors. The

memory available to the extension was also limited since computation occurs on client machines. Keeping this in mind, steps were taken to keep the training time in check to limit the impact on user experience.

Firstly, all words with a document frequency lower than a fixed threshold (set at 5) were eliminated from the TFIDF representation. Secondly, centroids for recipients to whom the number of messages sent was below a threshold (set at 5), were not calculated. After the model is trained, the parameter values are stored in a file on the user's computer. When the client is restarted, this model file is read thus preventing the need to retrain the system each time the client is started. The model file created by the training process stores the centroid vectors, the document frequencies and the recency/frequency ranks.

From the user's perspective, training is achieved by clicking on a "Sent" folder and hitting the "Train" button on Thunderbird's toolbar. The time taken for training depends on the number of messages in the sent folder, the speed of the processor, among other factors. A rough estimate is 150 messages per minute. A weekly reminder encourages users to retrain on a regular basis.

2.3 Prediction

After training is completed, Cut Once is ready to make predictions. The runtime predictions are triggered when the user hits the "Send" button for a message under composition. In this case, a dialog box pops up (see Figure 2), highlighting possible email leaks, and also listing other recommended recipients for the particular message just composed. Clicking on any of the predicted leak addresses will remove the address from the recipient list of the original message. Analogously, clicking on a recommended address will automatically add this address to the message recipient list. This dialog box has a countdown timer that sends the message after 10 seconds if the user does not take any action — thus ensuring that no additional action is needed to send a message. A screenshot of this dialog box can be seen in Figure 2.

2.4 Logging

CutOnce logs information about many aspects of the extension usage. This includes information such as the rank of an address that the user clicks on, the time taken by the user to click on that address, and the rank and prediction score of the address clicked by the user. The complete list of attributes logged by Cut Once are shown in Table 1.

1	whether the use used the explicit Send button or let the timer expire
2	whether the user deleted a recipient (possibly due to a potential leak)
3	rank of the deleted recipient in the leak list
4	confidence score of the recipient deleted
5	time elapsed before the recipient was deleted
6	rank of the added recipient in the recommendation list
7	time elapsed before recipient was added
8	confidence score of recipient added
9	number of messages in the user's Sent folder
10	number of recipients addressed in the Sent folder
11	Cut Once software version
12	baseline ranking method (TFIDF or MRR)

Table 1: Set of attributes logged by Cut Once

Every week the user is reminded to send the logged information via email to the user study researchers. If the user acquiesces, a new email compose window is opened up with the log information prefilled in the content section. The logging message does not contain any personal or private information from the user (such as email content or recipients), nor from any of the user's contacts. Users are also encouraged to send in comments in a designated area in this email.

In addition to the weekly reminders, at any time the user can also send this logging message by clicking on the "Mail Statistics" button (Einstein button) of the main Thunderbird window (see Figure 1).

3. EXPERIMENTS

Several users were recruited using web forums and newsgroups messages for a four-week long user study. These participants were told that the goal was to study how to improve the way people address email messages based on intelligent addressing techniques [3, 4]. Participants were required to be Thunderbird users, to write email using Thunderbird on a daily basis, and to be at least 18 years-old. The recruitment message also indicated that the task would be simple, with minimum or no interruptions at all.

After contacting the study researchers indicating their interest, participants were instructed on how to install and train Cut Once. After successfully installing and training the extension using the procedures described at Cut Once's website[4], participants received a message explaining exactly what Cut Once could do. They were also instructed to keep on using Thunderbird as usual, and that in one week Thunderbird would request them to send an initial logging message to the user study researchers.

After this logging message was received and analyzed, qualified participants were partially compensated (20% of total compensation) and invited to participate in the second phase of this user study. Qualification was based on frequency of email use during this first week, number of addresses in the Sent folder, and the number of message previously sent using Thunderbird. The main purpose of this procedure was to avoid selecting users who rarely used Thunderbird, or users who used Thunderbird to email a few people only — for obvious reasons, these cases would not add value to our experiments.

In the second phase of the study, participants were compensated with the remaining 80% of the total compensation after three more weeks using Cut Once[5]. They also had to complete an initial questionnaire with general questions, as well as a final questionnaire exclusively about Cut Once. The final questionnaire was about the general Cut Once experience, quality of predictions, interface issues, and usability, as well as suggestions for improvement. Results are reported below.

4. RESULTS

4.1 Adoption

[4] `http://www.cs.cmu.edu/~vitor/cutonce/cutOnce.html`.
[5] Due to scheduling conflicts to arrange the final questionnaire interview, many participants ended up using Cut Once for more than than 3 weeks.

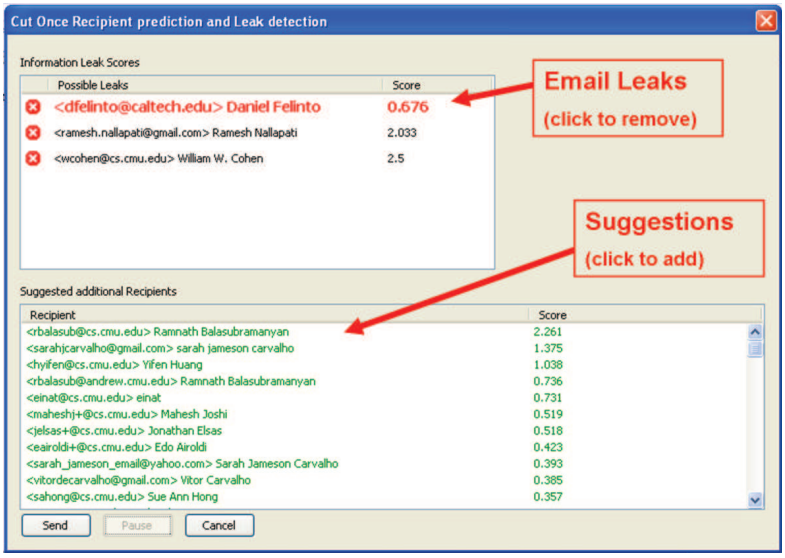

Figure 2: The information leak predictions (top part) and recipient suggestions (bottom part) dialog window displayed after the user clicks on the Send button. Clicking on any of the displayed email addresses will either remove (leaks) or add (suggestions) it from/to the message recipient list.

A total number of 26 participants completed the study: 4 female and 22 male. Ages ranged from 18 to 49 years-old, with an average of 31.7 and median of 28.5 years. From the 26 subjects, 13 were graduate students, mostly from Carnegie Mellon University or from the University of Pittsburgh. Other reported occupations were software engineers, system administrators, undergraduate students, one staff member and one faculty.

Subjects used Thunderbird on a daily basis, composing messages largely in English. During the user study, subjects composed 2315 messages using Mozilla Thunderbird, with an average of 11 messages sent per week. According to statistics collected from Sent directories, on average, subjects had written 2399 messages to 113 different recipients before the beginning of the user study. An average of 2.4 devices (computer, cell phone, etc.) per person were used to compose emails.

Another 17 users started but did not finish the study. They installed and successfully trained Cut Once, sent out at least one logging message, but stopped sending these messages not long after that. Either these users did not qualify to the second phase of the study, or voluntarily stopped sending logging messages.

In addition to these, 11 users showed initial interest and contacted the researchers, but were never able to send a single logging message. In these cases it is hard to know exactly the reasons for the discontinuation. Perhaps these users found Cut Once uninteresting or annoying after installation, or became unmotivated by the low compensation and lengthy nature of the study. We speculate that one of the main reasons is the slow training process.

Installation of Cut Once was smooth for all participants, but training frequently was not. Many users complained that training took too long or got "stuck" in a few messages. It was indeed a problem — Javascript is a slow interpreted language, not suited to large amounts of textual data processing. As expected, this issue affected more severely users with large number of messages, or users having a few very large messages.

Mozilla provides a portal for developers and practitioners of their open source softwares. We submitted Cut Once to Mozilla Thunderbird Sandbox, and it is currently available at `https://addons.mozilla.org/en-US/thunderbird/statistics/addon/6392`. According to their statistics, there has been 116 downloads of Cut Once from their site so far. Mozilla may have helped advertise the extension and the associated user study.

4.2 Usage and Predictions

As previously explained, Cut Once provided an interface in which the predicted email leaks could be automatically removed from the addressee list with a click. Eighteen out of the 26 subjects used it at least once. Overall, these 18 subjects used the leak deletion functionality in approximately 2.75% of their sent messages.

The final interviews revealed two main reasons why subjects utilized the leak deletion interface. First, some subjects clicked on these suggested leaks to play with the extension, particularly right after installation and training. Other subjects, as revealed in their final interviews, utilized the leak deletion button to "clean up" the addressee list — to remove unwanted people after hitting the reply-all button, or to remove themselves as recipients (some clients are configured to automatically include the sender as a CC'ed recipient).

Unfortunately, none of the subjects reported using the delete leak functionality to actually remove a real case of email leak. However, it does not mean that they did not occur among the 2315 messages sent throughout the user study. In fact, four different subjects reported that Cut Once correctly caught real email leaks. After noticing the mistake, all four subjects rushed to click on the cancel button, immediately closing Cut Once's dialog window and con-

sequently not reporting the real leak case in next logging message. Instead of deleting the leaks using Cut Once's interface, the reasons why these users canceled the dialog window were because subjects were uncomfortable or unfamiliar with the interface features, or because subjects were feeling under pressure due to 10-second timer, or a combination of both.

The first of these subjects was a network administrator, who addresses several users everyday by their aliases (user IDs). He reported that he confused two students with very similar alias, and Cut Once alerted him to the mistake. A similar case happened to a systems administrator, who frequently uses auto-completion to select recipients. He reported that in two or three different messages, one of the addresses selected by auto-completion was wrong, and that Cut Once correctly warned him of the potential email leak. An undergraduate student reported that he confused the email addresses of two acquaintances with very similar names, and Cut Once helped prevent that email leak. A graduate student reported that he used the reply-all button when he should not have, and Cut Once caught one of the unintended addresses as a leak.

Since one of the subjects reported Cut Once catching leaks in "two or three" different messages, henceforth we assume that five leaks were caught by the extension during the user study. This is a likely lower bound on the real number of leaks for that population, given that in some cases users do not even realize their addressing mistakes. Three out of these five real leaks came from subjects using the TFIDF baseline ranking method, and the remaining two leaks had subjects using the MRR baseline. Data from these four subjects did not reveal any strong correlation with the number of sent messages, nor with the number of observed leak deletions using Cut Once. Likewise, no correlation was observed with the number of entries in the subject's address book.

Overall there were five real email leaks in 2315 sent messages. A sample average of approximately 0.00215982 email leaks per sent message, or one email leak occurrence per 463 sent messages. Assuming email leak occurrences follow a binomial distribution with probability of success $p = \frac{5}{2315}$, it would be necessary at least 321 messages for having a 50% chance to experience at least one email leak, and 1066 messages for a 90% chance.

Three out of the five leaks caught by Cut Once came from subjects whose occupations require a lot of email message handling (a systems administrator and a network administrator), even though only 5 out of the 26 subjects had professions demanding substantial email handling. A binomial test on this data indicates that, with approximately 95% confidence, users whose professions require lots of message handling have a higher probability of generating leaks than other professions. Indeed, this agrees with subject's final questionnaire answers, where it was reported that the most likely users to benefit from the functionalities provided by Cut Once are persons who work with many different people, send a lot of messages or manage several different projects (e.g., secretaries, administrators, executives).

The other functionality provided by Cut Once was recipient recommendation. With a click on the suggested addresses, users could add recipients to messages under composition. A total of seventeen of the subjects used the functionality at least once. Overall, these 17 subjects utilized the email suggestions functionality in approximately 5.28% of their sent messages.

Considering all subjects in the study, there were 95 accepted suggestions in 2315 sent messages. A sample average of approximately 0.041036 accepted suggestions per sent message, or one accepted suggestion occurrence per 24.37 sent messages. There are a few reasons behind these low numbers. Some users did not seem interested in the functionality, others claimed that they simply "did not need it", while others did not even know that recipients could be added by clicking on the suggested email addresses. Another issue was the fact that the pop-up window with recommendations was triggered on all sent messages, regardless whether it was a new composition or a reply, and many subjects claimed that the proposed functionalities were not necessary in case of replies, particularly to a single recipient only[6]. Another consequence of triggering predictions on all sent messages is that the leak detection false positive rate (or false alarm rate) was high: $\frac{2315-5}{2315} = 0.99784$.

Ideally Cut Once should only provide predictions if models are reasonably confident of a leak or a missing recipient. However, learning a user-based confidence threshold can be challenging, particularly for users with a small number of messages. Also, if it adopted a fixed arbitrary threshold, not all real leaks would be displayed to the user, potentially causing the number of reported leaks (a very rare event) to be even lower. Because of these issues, we left the implementation of confidence-based triggered predictions as future work.

Cut Once presented recipient recommendations in a scrollable window that could fit up to 9 addresses in a ranked list. The distribution of the ranks of the accepted recommendations (or clicked ranks) can be found in Figure 3. Figure Figure 3(a) shows the data in a histogram, Figure 3(b) displays the same data in a boxplot. The median clicked rank was 2, and first and third quartiles were, respectively, 1 and 7. This plot indicates that users typically clicked on the first 7 recommended addresses, and only rarely had to scroll down to higher positions of the ranked list.

The boxplots in Figure 4 show distributions of likert scores from the answers to the many questions in the final questionnaire. Final questionnaire's results were reported in such a way that higher scores corresponded to more positive answers. The usual likert scale used was 5(excellent), 4(good), 3(neutral), 2(bad) and 1(very bad).

As previously explained, Figure 3 can be seen as an indication of the reasonably good quality of Cut Once's suggestions. In fact, one of the questions in the final questionnaire was exactly about the quality of the suggested rank ("In your opinion, what was the quality of the suggested rank?"). The reported mean of this distribution was 3.46. The other likert questions were about the interface of Cut Once, how annoying the extension was, how helpful the extension was, how often the user used the suggestions, the overall experience of the user, and the user's general impression of Cut Once. All questions were supposed to be answered in a likert scale, although some subjects insisted in providing non-integer scores. Higher values reflect better impressions of Cut Once for all questions.

Overall, subjects were not annoyed by Cut Once interruptions — mean value was 4.18, between "never" and "rarely" annoying, and all reported scores were positive. Figure 4 also

[6]Unfortunately Cut Once could not distinguish between a reply and a compose action.

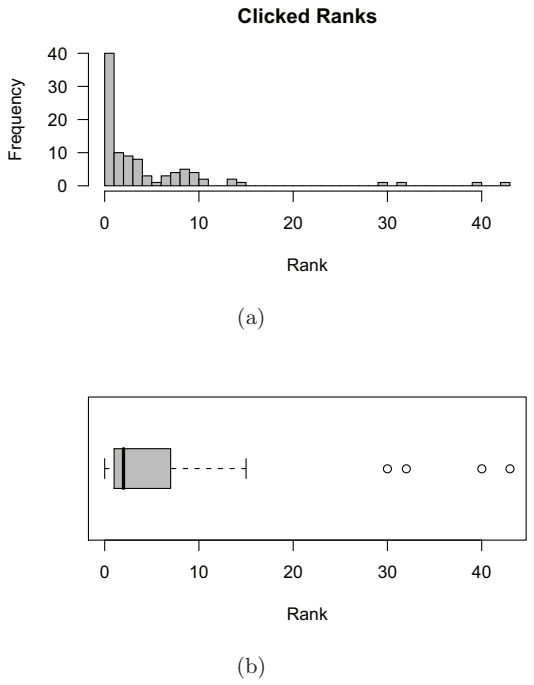

Figure 3: (a) Histogram with ranks of the recommendations clicked by the users. (b) The same data in a boxplot: median of distribution is 2.00, first quartile is 1.00 and 3rd quartile is 7.00. Whiskers mark the most extreme data point within a distance of 1.5 of the Interquartile range. Empty points indicate outliers.

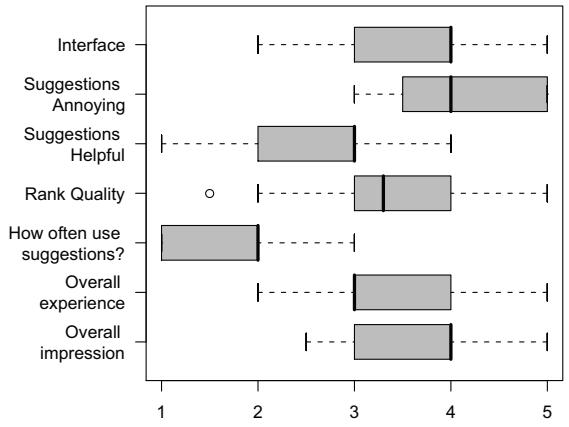

Figure 4: Distributions of likert scores (1 to 5) given as answers to the "yes or no" questions in the final user study questionnaire (higher=better)

indicates that Cut Once's interface was also well received, with mean value of 3.63 and median of 4.

Responses to question "How often did you use the suggestions", were largely negative, with a median of 2 and mean of 1.75 (between "never" and "rarely"). This reflects the fact that most of the time users were replying to messages, and not composing new messages. As previously noted, users accepted Cut Once's suggestions in approximately 6.17% of their sent messages. This fact is also linked to slightly negative responses on the question "Were the *suggestions helpful*?"), with median of 3 and mean 2.5 (between "kind of" and "marginally" helpful). The overall impression of the extension was positive — with median value of 4 and mean value of 3.6 (between "good" and "neutral"). A slightly positive judgment was seen on the *overall experience* using the extension — with a mean value of 3.36 and median of 3 (between "good" and "neutral").

Results from the binary questions in the final questionnaire are summarized in Table 2. The 15.38% affirmative answers to question 3 are exactly the four cases of successful leak detection described above. Three subjects reported changing the way they compose emails, as in question 8 of the questionnaire. They reported sometimes performing a *compose-then-address* procedure to send messages (i.e., writing the text of the message first, and then selecting recipients), instead of the traditional *address-then-compose*. In other words, these subjects became used to the the extension to a point that they were often relying on Cut Once to suggest the right recipients for the message they just composed. In fact, because clicking is faster than using auto-completion or typing complete addresses, users reported that this procedure was typically faster than the usual compose-then-address.

Also supporting the overall positive impression of the extension, question 11 revealed that 50% of the subjects would recommend Cut Once to their friends. The second part of question 11 was "who do you think would consider this extension helpful?". The most frequent answers were: people who work with many different persons, people who send a lot of messages or people who manage several different projects. Typical examples were secretaries, managers, executives and lawyers. Subjects also stressed that Cut Once should be much more helpful in the workplace than in handling personal messages.

Question 9 of final questionnaire asked if subjects would continue using the extension after the user study. Approximately 42% of them responded affirmatively. After this question, subjects were asked about problems, annoyances, software bugs, and how Cut Once could be improved. A summary with the most frequent limitations reported by the user subjects can be seen in Table 3.

After collecting user's complaints and ideas for improvement, Question 14 then asked "if your suggestions and ideas were implemented, would you consider using Cut Once permanently?". More than 80% of the subjects reported that they would — a clear indication that recipient prediction and leak detection were considered welcome additions to the subject's email clients, in spite of Cut Once's limitations.

4.3 Baseline Comparison

Previously we described Cut Once as having a mechanism to randomly assign a different ranking baseline (either MRR

Question	Affirmative response
Q.3) "Did Cut Once catch any leak?"	15.38% (4 users)
Q.8) "Did Cut Once change the way you compose emails?'	11.53% (3 users)
Q.9) "Would you keep on using Cut Once after this study?"	42.30% (11 users)
Q.11) "Would you recommend Cut Once to your friends?"	50.00% (13 users)
Q.14) "If your suggestions and ideas were implemented, would you consider using Cut Once permanently?"	80.77% (21 users)

Table 2: Percentage of the 26 subjects giving affirmative answers on four questions of final questionnaire.

Slow training procedure It needs incremental training (instead of batch training) The reminder to retrain every week was annoying Cannot use (train) multiple email accounts
Dialog box pops up even when message is being replied It should prompt a leak only if highly confident Place suggestions on the side, not in a separate pop-up It needs more configuration parameters
Timer countdown made people nervous. Remove it. Unclear indications of what happens when we click Confusing confidence scores

Table 3: Frequent issues and complaints about Cut Once reported by the subjects. Most frequent one are placed on the top.

or TFIDF) to different users. From the 26 subjects, sixteen were assigned TFIDF ranking, while the remaining ten used MRR-based ranking.

Table 4 compares results from these two populations. Average values and standard variations of several metrics are compared, and larger values are indicated in bold. The first three variables were extracted from the logging messages: the user-averaged number of clicked address suggestion per sent message, the user averaged number of removed leaks per message, and the average rank clicked by the user. The other variables in Table 4 were extracted from the final questionnaire.

A non-paired t-test applied to these populations indicated that none of the metric differences observed in Table 4 are statistically significant. The same observation was confirmed by a non-parametric Mann-Whitney U Test, as well as by a Heckman Sample Selection test, indicating that there was no perceived difference between the two baseline ranking methods.

A closer look in the ranks of clicked suggestions can be seen in Figure 5. This figure shows two boxplots with distributions of the ranks of the suggestions accepted (clicked) by the study subjects. On the top it shows the distribution of clicked ranks for subjects having MRR as baseline method, while in the bottom for subjects having TFIDF as baseline method. After removing outliers, the average ranks are 3.69 and 3.147 for, respectively, TFIDF and MRR. Although MRR shows better average ranking than the TFIDF baseline, the difference is not a statistically significant (p-value=0.394 in a non-parametric Mann-Whitney U Test).

The observation that these two baseline ranking methods did not produce statistically significant differences, although

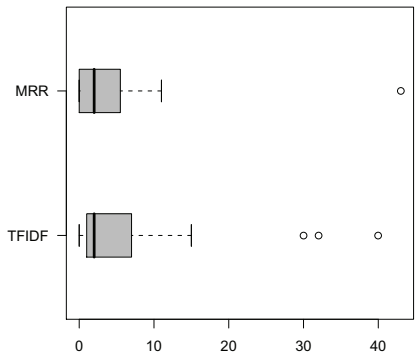

Figure 5: Distributions of ranks of clicked suggestions for both baseline methods.

somewhat limited because of the small number of subjects in the study, was not entirely surprising. There have been a few studies in the Information Retrieval (IR) literature also suggesting that users often cannot perceive much difference in the results provided by ranking systems having different performance levels. For instance, [11] described a web search task in which controlled levels of ranking performance (Mean Average Precision, or MAP, from 55% to 95%) were presented to subjects. They found that different MAP levels had no significant correlation with a precision-based user performance metric, while there was a weak correlation with a recall-based user performance metric. More recently, Scholer et at. [10] investigated how web search clickthrough data was related to the quality of search results. Their experiments showed that user click behavior did not vary significantly for different levels of MAP in displayed results, although there was a significant variation among different users.

However, email recommendation and web search are fairly different tasks, and further investigation will be necessary to adequately address to which extent traditional IR performance metrics correlate with user evaluation on the proposed email-based tasks. Another interesting question for future research is how to derive new automated evaluation metrics that can closely approximate user satisfaction.

5. DISCUSSION AND LESSONS LEARNED

Ideally this study would have benefited from a larger pool of user subjects, but unfortunately recruiting more people was not possible due to financial constraints. As previously explained, many subjects showed initial interest but discontinued using Cut Once in a short period of time. Among the reasons for this discontinuation, one can list the slow training process, the relatively small compensation (25 dollars) for a 4-week long study and the annoyance of the interruptions.

However, in case Cut Once's functionalities are implemented in a real large-scale email server (such as Gmail or Hotmail), adoption would be primarily decided by two factors: the cost of the interruptions versus the benefit of the provided predictions. In principle, interruption costs

METRIC	Mean		St.Dev.	
	TFIDF	MRR	TFIDF	MRR
Num. Clicked Suggestions per Sent Message	**0.089**	0.037	0.142	0.066
Num. Clicked Leaks per Sent Message	**0.033**	**0.033**	0.035	0.044
Average Rank Clicked by User (lower=better)	4.928	**4.505**	5.289	4.587
Overall Impression (1 to 5)	3.468	**3.850**	0.531	0.579
Overall Experience (1 to 5)	**3.406**	3.350	0.612	0.818
How Often Used Suggestions (1 to 5)	**1.843**	1.600	0.569	0.699
Rank Quality (1 to 5)	3.437	**3.510**	0.928	0.966
Suggestions Helpful (1 to 5)	2.437	**2.600**	1.014	1.074
Suggestions Annoying (1 to 5)	4.031	**4.430**	0.784	0.748
Interface (1 to 5)	**3.718**	3.500	0.657	1.054

Table 4: Comparison of different metrics for the two baseline methods (higher = better).

can be lowered with carefully designed interfaces and well-tuned confidence-based decisions, and prediction models can be made more accurate as more data is collected. As long as users perceive the system as having a good cost/benefit, widespread adoption of these functionalities can be reached.

To help design these functionalities in large systems, below we present a few guidelines based on the results of this user study and final questionnaires. First, training should not be noticeable by the user. Also, training should also be incremental, that is, prediction models should be immediately updated as new messages are sent. Second, leak detection and recipient recommendation should be independent functionalities, potentially with independent models and interfaces. Third, interfaces should be as unobtrusive as possible. They should also, if possible, provide leak detection alarms and recipient recommendations in the same window in which messages are composed. Fourth, interruptions should be triggered by confidence-based decisions. Fifth, ideally, predictions should be available anytime during the message composition process, and not only after the user hits the "send" button. Predictions could be provided, for instance, at the end of each composed sentence. Additionally, prediction models should account for different user "send" actions (reply, reply-all or compose). And finally, users should be allowed to control a few parameters, such as timer period, number of suggested addresses displayed to the user, interruption confidence threshold, etc.

6. CONCLUSIONS AND RELATED WORK

In this paper we presented the deployment of data mining algorithms for email leak and suggestion predictions in Thunderbird, a popular open-source email client. The project was written in Javascript, thus requiring careful design decisions to optimize memory and processing resources on client machines.

We then presented results from a 4-week long study of this deployment that led to very encouraging results. Cut Once prevented five real cases of email leaks, and provided predictions with reasonable rank quality and little user annoyance. It was able to change the way three subjects send email, and left an overall positive impression in the large majority of the users. More than 80% of the subjects would permanently use Cut Once in their email clients if a few interface/optimization improvements are implemented. The most likely users to benefit from these functionalities are persons who work with many different people, send a lot of messages or manage several different projects (e.g., secretaries, managers, executives). In fact, three out of the five leaks caught by Cut Once came from subjects whose occupations require a lot of email message handling (a systems administrator and a network administrator).

Regarding related work, previous researchers [1, 2] described a privacy enforcement system in which information extraction techniques and domain knowledge were combined to monitor specific privacy breaches via email. They were particularly concerned with entity breaches in a university environment, such as student names, student grades or student IDs. Although related, this system had a different goal and can only be applied to the situations in which domain knowledge is available. The most related reference to Cut Once's deployment is Facemail [7], an extension to a webmail system developed to prevent misdirected email by showing faces of recipients in a peripheral display while the message is under composition. Preliminary results from this user study suggested that showing faces could significantly improve users' ability to detect misdirected emails with only a brief glance. In principle, many of the ideas in Facemail can be combined with the algorithms provided by Cut Once, potentially leading to a much better leak detection mail system.

7. REFERENCES

[1] N. Boufaden, W. Elazmeh, Y. Ma, S. Matwin, N. El-Kadri, and N. Japkowicz. Peep— an information extraction base approach for privacy protection in email. In *CEAS*, 2005.

[2] N. Boufaden, W. Elazmeh, Y. Ma, S. Matwin, N. El-Kadri, and N. Japkowicz. Privacy enforcement in email project. In *Proc. Of the Privacy, Security and Trust Conference*, 2005.

[3] V. R. Carvalho and W. W. Cohen. Preventing information leaks in email. In *SDM*, Minneapolis, MN, 2007.

[4] V. R. Carvalho and W. W. Cohen. Ranking users for intelligent message addressing. In *ECIR*, 2008.

[5] I. O. Craig Macdonald. Voting for candidates: Adapting data fusion techniques for an expert search task. In *CIKM*, Arlington, USA, 2006.

[6] T. Joachims. A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization. In *Proceedings of the ICML-97*, 1997.

[7] E. Lieberman and R. C. Miller. Facemail: Showing faces of recipients to prevent misdirected email. In *SOUPS*, 2007.

[8] P. Ogilvie and J. P. Callan. Combining document representation for known item search. In *ACM SIGIR*, 2003.

[9] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[10] F. Scholer, M. Shokouhi, B. Billerbeck, and A. Turpin. Using clicks as implicit judgments: Expectations versus observations. In *ECIR*, 2008.

[11] A. Turpin and F. Scholer. User performance versus precision measures for simple search tasks. In *SIGIR*, 2006.

[12] Y. Yang and X. Liu. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49, August 1999.

Thunderbird, developed by Mozilla Corporation is a free, open-source app that allows managing emails, chats, news feeds, and newsgroups quite efficiently. It is a desktop based application that gives complete control and extensive ownership over the email messages. If you use Thunderbird, there are a number of add-ons available that you can install and utilize to customize and enhance the emailing experience. The best thing is that you can download, install, and can use the program for free under defined terms and conditions. Different versions of the application are available that supports Wi Why should I use it? Thunderbird is the email client portion of Mozilla, an open-source suite of applications that can be downloaded for free over the Internet. Using a full-featured email client like Thunderbird offers several advantages that web-based mail interfaces like Hotmail, Gmail, Yahoo mail, and UT Webmail cannot. Thunderbird allows a user to access and manage multiple email accounts simultaneously.Â Downloading news and information directly to your inbox is quite easy; Thunderbird can access multiple websites that are updated on a frequent basis, including news sites and blogs (weblogs). Another facet of Thunderbird service encompasses the realm of Usenet (or Newsgroups), as Thunderbird can also access and manage multiple Newsgroup accounts. How can I send an e-mail using Mozilla Thunderbird from vb.net? Does Thunderbird has an API or something like Microsoft.Office.Interop.Outlook for Outlook? I've tried using command line '-compose' but it just opens the send new e-mail window in Thunderbird. I want to send it without any user interaction. Is that possible? Thanks! email thunderbird. share|improve this question. asked Feb 4 '15 at 11:45.