

# Preventing Information Leaks in Email

Vitor R. Carvalho  
Language Technologies Institute  
Carnegie Mellon University

William W. Cohen  
Machine Learning Department  
Carnegie Mellon University

## Abstract

The widespread use of email has raised serious privacy concerns. A critical issue is how to prevent email information leaks, i.e., when a message is accidentally addressed to non-desired recipients. This is 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.

In this paper we present the first attempt to solve this problem. We begin by redefining it as an outlier detection task, where the unintended recipients are the outliers. Then we combine real email examples (from the Enron Corpus) with carefully simulated leak-recipients to learn textual and network patterns associated with email leaks. This method was able to detect email leaks in almost 82% of the test cases, significantly outperforming all other baselines. More importantly, in a separate set of experiments we applied the proposed method to the task of finding real cases of email leaks. The result was encouraging: a variation of the proposed technique was consistently successful in finding two real cases of email leaks. Not only does this paper introduce the important problem of email leak detection, but also presents an effective solution that can be easily implemented in any email client — with no changes in the email server side.

## 1 Introduction

On July 6th 2001, the news agency Bloomberg.com published an interesting article entitled **California Power-Buying Data Disclosed in Misdirected E-Mail**<sup>1</sup>. An excerpt is reproduced below:

*“California Governor Gray Davis’s office released data on the state’s purchases in the spot electricity market — information Davis has been trying to keep secret — through a misdirected e-mail. The e-mail, containing data on California’s power purchases yesterday, was intended for members of the governor’s staff, said Davis spokesman Steve Maviglio. It was accidentally sent to some reporters on the office’s press list, he said. Davis is fighting disclosure of state power purchases, saying it would compromise negotiations for future contracts”.*

This was a famous case of information leak via email, where a message was accidentally sent to unintended recipients. This episode, however, was by no means an isolated case. In fact, most regular email users have received such misdirected email messages, often due to email clients that

are overly aggressive at completing partial email addresses.<sup>2</sup> With the widespread use of email, it is reasonable to expect that an increasing number of email users will experience similar situations — as a sender of an information leak or, more frequently, as a recipient.

As the California Power-Buying example above indicates, unintentional email leaks can be disastrous. They can lead to major negotiation setbacks, losses in market share and financial burdens. Furthermore, when related to personal or corporative privacy policies, an email leak can potentially be the cause of expensive lawsuits and irreparable brand reputation damage. Even though it is not easy to estimate the amount of loss caused by information leaks, one thing is for certain: such incidents should be avoided at all costs. In this paper we present a new technique to prevent sending email messages to unintended recipients. To the best of our knowledge, this is the first attempt to solve this critical problem.

We approach this problem using the following methodology. We start by casting it as an *outlier detection* problem: i.e., we model the messages sent to past recipients, and consider a (messages,recipient) pair to be a potential leak if the message is sufficiently different from past messages sent to that recipient. This approach has the advantage that it can be easily implemented in an email client—it does not use any information that is available to the server only.

To evaluate different approaches of this type, we require data. Since we do not have access to a considerable number of real cases of unintentional email leaks, we create artificial cases of unintended recipients in real-world email data. More specifically, we simulate email leaks in the Enron Email corpus<sup>3</sup> [2] using different plausible criteria. These criteria imitate realistic types of leaks, such as misspellings of email addresses, typos, similar first/last names, etc.

On this benchmark data, we evaluate a number of leak-detection methods. We show that a classification-based approach works best. In this method, we extract textual

<sup>2</sup>For instance, during his employment at AT&T Research, the linguist Steven Abney noted that he received a large number of misdirected emails that had been addressed to `steven@att.com`. At the time he was the alphabetically first Steven in the AT&T corporate directory of email addresses.

<sup>3</sup>A large collection of real email messages from several managers and employees of the Enron Corporation. This data was originally made public by the Federal Energy Regulatory Commission during the investigation.

<sup>1</sup>In Sep 2006, the entire article could be found at <http://www.freerepublic.com/forum/a3b4611e82dc0.htm>

and social network features from the messages, and then use supervised learning techniques to predict email leaks. Evaluations show that this technique can correctly identify the (synthetically-introduced) “leak recipient” in almost 82% of the messages.

In a separate set of experiments we apply a variation of this technique to real data. In particular, we searched the Enron data for real cases of email leaks, and evaluated the method on these. We show that a variation of our method could successfully handle two independent real cases of email leaks (unintended message recipients) in the Enron corpus. This result shows that the proposed technique is effective, and has the potential to prevent actual email leaks in realistic scenarios.

The paper is organized as follows. In Sections 2.1 and 2.2, we introduce the Enron data and describe our approach to detecting email leaks. Then in Section 2.3 we describe the criteria used to introduce artificial email leaks in the Enron data. Sections 3.1 and 3.2 explain our experiments and how the different sets of features were combined to produce a very robust and effective model for the problem. In Section 4 we demonstrate that our model was successful in detecting two real cases of leaks in the Enron data. We then present related references and a discussion of results in Section 5.

## 2 Dataset

**2.1 The Enron Collection** Although email is ubiquitous, large, public and realistic email corpora are not easy to find. The limited availability is largely due to privacy issues. For instance, in most US academic institutions, a email collection can only be distributed to researchers if all senders of the collection also provided explicit written consent.

In all experiments of this paper we used the Enron Email Corpus, a large collection of real email messages from managers and employees of the Enron Corporation. This collection was originally made public by the Federal Energy Regulatory Commission during the investigation of the Enron accounting fraud. We used the Enron collection to create a number of simulated user email accounts and address books, as described below, on which we conducted our experiments.

As expected, real email data have several inconsistencies. To help mitigate some of these problems, we used the Enron dataset version compiled by Jitesh and Adibi [7], in which a large number of repeated messages were removed. This version contains 252,759 messages from 151 employees distributed in approximately 3000 folders.

Another particularly important type of inconsistency in the corpus is the fact that a single user may have multiple email addresses. We addressed part of these inconsistencies by mapping between 32 “raw” email address and the normalized email address for some email users. This mapping (author-normalized-author.txt) was produced by Andres

Corrada-Emmanuel, and is currently available from the Enron Email webpage [2].

For each Enron user, we considered two distinct sets of messages: messages sent by the user (*sent collection*) and messages received by the user (*received collection*). The received collection contains all messages in which the user’s email address was included in the *TO*, *C.C.* or *B.C.C.* fields. The sent collection was sorted chronologically and then split into two parts, *sent\_train* and *sent\_test*. *Sent\_train* contains 90% messages sent by the user, corresponding to the oldest ones. The most recent messages, 10% of the total sent collection, were placed in *sent\_test*. The final message counts for 20 target Enron users is illustrated in Table 1.

Enron user	received	sent_train	sent_test
rapp	408	146	17
hernandez	792	1326	15
pereira	737	179	20
dickson	1263	198	22
lavorato	1930	361	41
hyatt	1797	566	63
germany	466	729	82
white	922	441	50
whitt	836	414	46
zufferli	324	314	35
campbell	1383	531	60
geaccone	889	396	44
hyvl	1246	650	73
giron	667	999	111
horton	964	426	48
derrick	1283	686	77
kaminski	1042	1097	122
hayslett	1590	706	79
corman	2274	686	77
kitchen	5681	876	98

Table 1: Number of Email Messages in the Different Collections

This 90%/10% split was used to simulate a typical scenario in a user’s desktop — where the user already has several sent and received messages, and the goal is to predict if the next sent message will be an information leak. In order to make the received collection consistent with this, we removed from it all messages that were more recent than the most recent message in *sent\_train*. The general time frames of the different email collections is pictured in Figure 1.

We also simulated each user’s address book: for each Enron user  $u$ , we build an address book set  $AB(u)$ , which is a list with all email addresses that can be found in the *received* and *sent\_train* collections of this user. More precisely, the list was constructed using information from both *sent\_train* and *received* collections, but sent and received

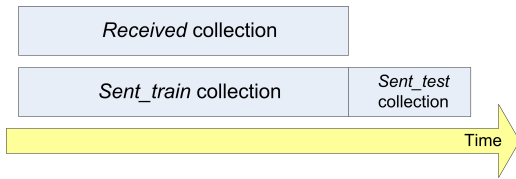


Figure 1: Time frames of different email collections.

messages are used in different ways. From `sent_train`, we consider all email addresses that were recipients of at least one message. In the `received` collection, on the other hand, we disregard all message recipients—in other words, we consider all email addresses from the senders of messages, and only the senders. The message recipients are not added to  $AB(u)$  because a received message is a communication between its sender and all its recipients, and not among recipients—i.e., a particular recipient does not necessarily know the other recipients.

In all our experiments we represented the content of the messages with a “bag of words”, where the counts of all tokens in a message were extracted and taken as feature weights. In this process, a small set of stop words<sup>4</sup> was removed from the email body. In addition, self-addressed messages with no other recipients were also disregarded.

Only the first six Enron users (`rapp`, `hernandez`, . . . , `hyatt`) were used during the development of our methods. After all development and tuning were complete, the remaining 14 Enron users were added to the test collection as an evaluation set. As we will see, performance is quite similar on the two collections of users.

**2.2 Generating Synthetic Leaks** Information leaks in email are a relatively common problem. Most email users have experienced a “strange” message in their mailboxes and probably spent some time trying to remember who the sender was, or why they were copied in the message. In spite of this, collecting sufficient data for a controlled experiment is a challenging task, largely due to privacy concerns.

In this work we addressed this limitation by using real data (the Enron Email collection) in combination with synthetic information leaks, or simulated *leak-recipients*. In outline, leaks are simulated by adding an additional recipient to emails from a user’s `sent_test` collection. This process thus allows us to evaluate performance on over 1100 simulated leaks for 20 users. There are several plausible ways to add these “leak-recipient” users, and details on this process are described in Section 2.3.

<sup>4</sup>about, all, am, an, and, are, as, at, be, been, but, by, can, cannot, did, do, does, doing, done, for, from, had, has, have, having, if, in, is, it, its, of, on, that, the, they, these, this, those, to, too, want, wants, was, what, which, will, with, would

After the simulated leak recipients are generated, we can then attempt to learn leak patterns and automate the process of leak prediction. We start by using features based on the email contents (Section 3.1). We then improve the prediction by reranking the text-based results using social network features (Section 3.2).

**2.3 Leak Criteria** Accidental email leaks can happen in various situations. A typical case is when the message is a reply to a previous message but not all previous recipients should be included. Another common situation is when one of the intended recipients has a similar first name (or surname, or email address) of another entry in the user’s contact list. The latter scenario is particularly frequent when the email client uses aggressive auto-completion of addresses and/or contact names.

To simulate the latter situation, we developed the following procedure to create leak-recipients (or outliers)—i.e., the email addresses that are unintentionally included as a recipient. We will assume that for the `sent_test` messages, the recorded list of recipients were all intended recipients, and that no other recipients were intended; thus leak-recipients can be generated by simply adding some other recipient to the message. However, we elected to simulate a certain plausible process for generating email leaks; specifically, we elected to simulate the actions of an email client that provides the recipient in response to an incompletely-specified email address. The procedure we used is illustrated in Table 2 and we refer to it as *3g-address* henceforth.

For a given message with  $n$  recipient addresses (i.e., the set of recipient addresses  $A = \{a_1..a_n\}$ ), we randomly select one of the addresses  $a_i$ . We then consider the addresses  $AB(u)$  in the address book of the user, discard addresses in  $A$ , and search for other addresses that start with the same three initial characters as  $a_i$ . For instance, if  $a_i = \text{marina.carvalho@enron.com}$ , we would return all email addresses in  $AB(u) - A$  starting with the sequence of characters “*mar*”<sup>5</sup>. If the returned list is not empty, we randomly select one of the addresses as the leak-recipient and finish the procedure; otherwise, we find all addresses in  $AB(u)$  that cannot be found in  $A$  and start with the same two initial characters as  $a_i$  (i.e., the characters “*ma*”<sup>6</sup>). If this list is not empty, we randomly choose one of the entries as the leak-recipient and end the procedure; otherwise, we find all addresses in  $AB(u)$  that and cannot be found in  $A$  and start exactly the same initial character of  $a_i$  (i.e., the character “*m*”<sup>7</sup>). If this list is not empty, we randomly select one of the entries as leak-recipient and finish the procedure; otherwise, we randomly select any address from  $AB(u)$  (that cannot be found in  $A$ ) and return it.

<sup>5</sup>For instance, `mary...`, `marco...`, `margaret...`, `marcia...`, etc.

<sup>6</sup>For instance, `matthew...`, `may...`, `manuel...`, `madaline...`, etc.

<sup>7</sup>For instance, `melyssa...`, `michael...`, `monika...`, `morgan...`, etc.

Table 2: 3g-address, an Information Leak Heuristic

- 
1. Input: User  $u$  and set of user’s messages  $M = \{m_1..m_j\}$
  2. Build user’s address book set  $AB(u)$
  3. For each message  $m_j$  in  $M$ :
    - (a) Randomly select  $a_i$  from set of recipients addresses  $A$  in  $m_j$ .
    - (b) Find set  $L3$  (i.e., all addresses in  $AB(u) - A$  with the same three initial characters of  $a_i$ )
    - (c) If  $L3 \neq \emptyset$ , randomly select leak-recipient from  $L3$
    - (d) Else
      - Find set  $L2$  (same as  $L3$  but using the two first characters instead)
      - If  $L2 \neq \emptyset$ , randomly select leak-recipient from  $L2$
      - Else
        - Find set  $L1$  (same as  $L1$  but using only the first character only)
        - If  $L1 \neq \emptyset$ , randomly select leak-recipient from  $L1$
        - Else, randomly select leak-recipient from  $AB(u) - A$
    - (e) Return the selected leak recipient
- 

Even though the *3g-address* is a reasonable criterion to simulate email information leaks, several other leak criteria could have been used. For instance, we could use a similar 3g-address criterion for first names and/or last names; or even some string distance similarity metric [3]. Unfortunately the Enron dataset does not include contact information (or address books) of most users; thus only a small percentage of the email addresses could have the first and last names extracted. Because of this limitation, we initially decided to apply only the 3g-address criterion when evaluating leaks in the Enron dataset. Later we will consider a variation of this process as well.

Using a particular leak criterion, we are able to simulate artificial leaks on real data. The idea is to add a single leak-recipient to the list of recipients of the message.

With large quantities of email messages with simulated email leaks, the problem now becomes finding the most effective way to predict the unintended recipients.

### 3 Methods for Email Leak Prediction

**3.1 Baselines: Using Textual Content** In this Section we develop different techniques for the leak prediction problem based on the textual contents of the messages. The main idea here is to model the “recipient-message” pairs, and then to predict the least likely pair (the worst outlier of the model) to be a leak-recipient. Predicting exactly one pair to be a leak is a reasonable choice, since in our simulated data, each message contains exactly one leak-recipient; however, all of the methods we describe actually produce a ranking of all message recipients. We start by using only the previously sent messages (sent\_train collection) as training set.

The first method was based on cosine similarity between

two vector-based representations of email messages. Given a message from user  $u$  to a set of recipients  $A$ , we derived the message’s TF-IDF (Term Frequency-Inverse Document Frequency) vector representation from its textual contents and then normalized the vector to length 1.0. The second representation was built from a concatenation of all previous messages sent from user  $u$  to a particular recipient  $a_i$  in  $A$ . In other words, we concatenated all previous messages sent from  $u$  to  $a_i$  and considered it to be one single large document. Analogously, we derived TF-IDF weights in the concatenated model, and normalized the TF-IDF vector to 1.0. We then computed the cosine similarities between the current message vector and the  $|A|$  concatenated vectors. The recipient associated with smallest similarity value is then predicted as leak-recipients. We refer to this method as *Cosine*.

The second method was based on the K-Nearest Neighbors algorithm described by Yang & Liu [8]. Given a message from user  $u$  addressed to a set of recipients  $A = \{a_1..a_n\}$ , we found its 30 most similar messages in the training set. The notion of similarity here is also defined as the cosine distance between the text of two normalized TF-IDF vectors. With the top 30 most similar messages selected from the training set, we then computed the weight of each recipient  $a_i$  according to the sum of similarity scores of the messages in which  $a_i$  was one of the recipients. After ranking all  $n$  recipients in the given message according to this method, we selected the one with lowest score as the predicted leak-recipient. We refer to this method as *Knn-30 (sent)*.

Both methods above can handle received messages using a very simple assumption: to treat received messages as sent messages with a single recipient — the sender. In

fact, this is consistent to what we did to extract the address books  $AB(u)$  in Section 2.3, where we only added to the address book the message senders from the received collection. We use the symbols (*sent*) or (*sent+rcvd*) to identify, respectively, the smaller (*sent\_train*) and the larger(*sent\_train* + *received*) training sets. (The *Cosine* method is shown only with *sent\_train* messages due to space constraints.)

The overall results in this section are shown in Table 3. This Table shows the experimental results for each Enron user. The results are expressed in terms of Precision at rank 1 (or *Prec@1*), i.e., the average number of times (in  $N$  trials) that the predicted leak-recipient is the actual leak-recipient. We used  $N = 10$  trials. On each trial, a completely new set of leak-recipients is generated for the training and test sets, and the experiment is completely repeated. The *Random* column shows the *Prec@1* values when the leak is chosen randomly from the recipient list.

From Table 3 we observe that, in average, the *Cosine* method had approximately the same level of performance of the *Knn-30* method. Another interesting point is that, compared to the baseline *Random*, the gain obtained by using textual information is obvious, but relatively modest. As we shall see in Section 3.2, much larger improvements in performance can be obtained by using social network features. Also from Table 3, it does not seem to make a lot of difference to add the received messages to the training set, since the average performance barely changed.

Enron user	Random	Cosine (sent)	Knn-30	
			(sent)	(s+r)
rapp	0.236	0.470	0.547	0.459
hernandez	0.349	0.226	0.247	0.353
pereira	0.459	0.490	0.450	0.465
dickson	0.462	0.627	0.641	0.659
lavorato	0.463	0.697	0.668	0.637
hyatt	0.400	0.488	0.533	0.586
germany	0.352	0.570	0.620	0.588
white	0.389	0.648	0.626	0.616
whitt	0.426	0.478	0.522	0.563
zufferli	0.479	0.628	0.654	0.697
campbell	0.385	0.454	0.422	0.451
geaccone	0.367	0.413	0.423	0.420
hyvl	0.455	0.523	0.467	0.436
giron	0.444	0.551	0.588	0.616
horton	0.460	0.646	0.604	0.615
derrick	0.454	0.784	0.758	0.668
kaminski	0.471	0.711	0.753	0.739
hayslett	0.304	0.547	0.561	0.551
corman	0.466	0.782	0.728	0.695
Kitchen	0.300	0.424	0.379	0.415
Average	0.406	0.558	0.560	0.561

Table 3: Email Leak Prediction Results: *Prec@1* in 10 trials.

### 3.2 Classification-Based Method: Using Social Network

**Information** So far we have considered only the textual contents of emails in the task of leak prediction. Yet, it is reasonable to consider social network features for this problem, such as the number of received messages, number of sent messages, number of times two recipients were copied in the same message, etc. In this Section we describe how these network features can be exploited to considerably improve performance on this problem.

In order to combine textual and social network features, we used a classification-based scheme. The idea is to perform the leak prediction in two steps. In the first step we calculate the textual similarity scores using a cross-validation procedure in the training set. In the second step, we extract the network features and then we learn a function that combines those with textual scores.

The textual scores are calculated in the following way. We split the training set (*received* + *sent\_train* collections) in 10 parts. Using a 10-fold cross-validation procedure, we compute the score for the *knn-30* on 90% of the training data and use it to make predictions in the remaining 10%. In the end of this process, each training set examples will have, associated with it, a list of email addresses (from the top 30 messages selected by *Knn-30*) and their predicted scores. Now we have an “outlier score” associated with each message recipient in the training set. These scores will be used as features in the second step of the classification procedure.

In addition to the textual scores, we used three different sets of social network features. The first set is based on the relative frequency of a recipient’s email address in the training set. For each recipient we extracted the normalized sent frequency (i.e., the number of messages sent to this recipient divided by the total number of messages sent by this particular Enron user) and the normalized received frequency (i.e., the number of messages received from this recipient divided by the total number of messages received by this particular Enron user). In addition, we used two binary features to indicate if no messages were sent to a particular user, and if no messages were received from a particular user. We refer to these features as *Frequency* features.

The second set of social network information is based on co-occurrence of recipients on other messages in the training set. The intuition behind this feature is that we expect leak-recipients to co-occur less frequently with the other recipients. Given a message with three recipients  $a_1, a_2$  and  $a_3$ , let the frequency of co-occurrence between recipients  $a_1$  and  $a_2$  be  $F(a_1, a_2)$  (i.e., the number of messages in the training set that had  $a_1$  as well as  $a_2$  as recipients). Then the relative co-occurrence frequency of users  $a_1, a_2$  and  $a_3$  will be proportional to, respectively,  $F(a_1, a_2) + F(a_1, a_3)$ ,  $F(a_2, a_3) + F(a_2, a_1)$  and  $F(a_3, a_1) + F(a_3, a_2)$ : i.e., the relative co-occurrence frequency of each recipient  $a_i =$

$\sum_{j \neq i} F(a_i, a_j)$ . These values are then divided by their sum and normalized to one. In case of two recipients only, the value of this feature is obviously 0.5 to each. No features will be extracted if the message has only one recipient. We refer to this feature in as *Coocurr* features.

We will call the third set of network features the *Max3g* features. To explain this feature set, we need to refer to Table 2 in the Leak Criteria Section. For each recipient  $a_j$  in a message, we return the *L3* set. And from the *L3* set we select the candidate  $a_m$  with the highest score (score from the cross-validation procedure). We then use this highest score minus the score of  $a_j$  as a feature. Since the scores are between 0 and 1, the final value of this feature can be normalized as  $\frac{\text{score}(a_j) - \text{score}(a_m) + 1}{2}$ . The intuition behind it is that leak-recipients are likely to have lower values for this feature, since their own scores are likely to be lower than their *L3* highest score. Obviously, if *L3* is empty, the *L2* set is used; and if the latter is empty, *L1* is used.

After the three sets of features are extracted, their values were discretized according to the following thresholds: 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000005 and 0.000001. The feature value is then represented by all thresholds that are smaller than it. (For example, if a feature B had a value 0.0003, its representation after being discretized would be “B-0001, B-00005, B-00001, B-000005, B-000001”. If the value of B were smaller than 0.000001 then an extra feature would be generated (B-000001L). This discretization process was used to increase the robustness of the learning algorithm.)

We used the Voted Perceptron [4] as learning algorithm, as an example of a learning method which is robust and effective, but efficient enough to be plausibly embedded in an email client. It was trained using five passes through the same training data, and training examples for each user’s leak-detection method were generated from the entire training collection (sent\_train + received) for the user. The learning proceeded in the following way. For each message with  $J$  recipients (where one of them is the leak-recipient), we created  $J$  examples: 1 negative example with the features associated with the leak-recipient and  $J - 1$  positive examples associated with the true recipients. The leak-recipient detection thus becomes a binary classification problem.

Experimental results using textual and network features are illustrated in Table 4. For comparison, the second column is the best text-only method from Table 3, i.e., the Knn-30 using both sent and received messages. The third column shows the Prec@1 values of our method using the cross-validation score in addition to the Frequency features. As we can see, results are surprisingly good, with very large performance improvements. On average, more than 80% of the test messages had their leak-recipients correctly predicted.

The fourth column reveals the performance of the cross-validation score in addition to the Coocurr features. Again, a general improvement compared to the textual-only methods can be observed, and for some users results were even better than the “+Frequency” column. However, in average results were not as good as using only the first set of network features.

The fifth column shows results associated to the Max3g features. Compared to the two previous feature sets, this is the least effective one, but still performing better than the best textual-only baseline.

The sixth column illustrates the performance results when all three feature sets are used in addition to the cross-validation scores. Again we observe very good results, better on average than all other feature sets taken in isolation and obviously considerably better than the best textual-only method. In average, this technique was able to detect the leak-recipients in almost 82% of the messages — a very good result in itself. The last column shows the relative gain in performance between the “All” column and the Knn-30 column. Gains for all users were observed, include all of the 14 evaluation-set users. (Recall that the method was fully developed and debugged on the first 6 users.) On average, the relative gain was nearly 49%.

Overall, Table 4 is a clear indication that the proposed method is very effective and robust in detecting email leaks, significantly outperforming all baselines for 20 different Enron users.

## 4 Finding Real Leaks on Real Data

**4.1 Finding Real Email Leaks** In previous sections we have presented promising results for the task of leak detection, but they were all based on artificially constructed data. It is not clear if the technique will in fact work for a real case of an email information leak.

First of all, we needed to find real leak cases and, as expected, this is not a trivial task. We approached the problem by performing some message filtering<sup>8</sup>, and then manually screening the results. Messages containing these terms tend to occur in the emails following a leak (typically in the same message thread), after someone realized the mistake. We discovered several cases of real email leak in the corpus. Unfortunately, most of these cases were originated by non-Enron email addresses or by an Enron email address that is not one of the 151 Enron users whose messages were collected — two situations in which our technique would not work since it requires the collection of sent and received messages of the same user. Eventually, we were still able to find two distinct email leaks associated with two different

<sup>8</sup>Selecting messages containing the terms *sorry*, *accident* or *mistake*. We were looking for sentences similar to “Sorry. Sent this to you by mistake. Please disregard.”, “I accidentally send you this reminder”, etc.

Enron user	Knn-30 (s+r)	CV-Scores				$\Delta$ (%) (to Knn-30)
		+Frequencies	+Cooccur	+Max3g	+All	
rapp	0.459	0.706	0.747	0.6352	0.788	71.796
hernandez	0.353	0.693	0.7466	0.6533	0.720	103.793
pereira	0.465	0.795	0.78	0.74	0.850	82.796
dickson	0.659	0.814	0.7909	0.7727	0.786	19.317
lavorato	0.637	0.898	0.7731	0.7536	0.910	42.922
hyatt	0.586	0.827	0.8222	0.7634	0.824	40.652
germany	0.588	0.659	0.6209	0.5938	0.665	13.240
white	0.616	0.832	0.776	0.6719	0.812	31.823
whitt	0.563	0.867	0.7826	0.7413	0.889	57.922
zufferli	0.697	0.806	0.7714	0.7971	0.809	15.980
campbell	0.451	0.703	0.7677	0.7457	0.739	63.909
geaccone	0.420	0.782	0.609	0.6613	0.789	87.583
hyvl	0.436	0.826	0.8205	0.7684	0.822	88.682
giron	0.616	0.831	0.7441	0.6729	0.858	39.176
horton	0.615	0.840	0.752	0.7479	0.856	39.333
derrick	0.668	0.942	0.8662	0.8207	0.934	39.880
kaminski	0.739	0.902	0.9213	0.9377	0.902	22.068
hayslett	0.551	0.778	0.5658	0.5556	0.747	35.634
corman	0.695	0.910	0.7792	0.7883	0.912	31.203
Kitchen	0.415	0.680	0.5173	0.5459	0.662	59.451
Average	0.561	0.804	0.748	0.718	0.814	49.358

Table 4: Email Leak Prediction Results: Prec@1 in 10 trials.

users in the original 151 Enron user set.

The first case happened in message germany-c/sent/930; which can be inferred from message germany-c/all\_documents/1489. In this case, the email leak contains 20 recipients and the leak corresponds to the address alex.perkins@enron.com. The second case is located in the message kitchen-l/sent\_items/497, and message kitchen-l/sent\_items/495 can confirm it. Message kitchen-l/sent\_items/497 contains 44 recipients, and in this case the leak address is rita.wynne@enron.com.

In order to detect these two leaks, we prepared the datasets in the same way as described in Section 2. We assured that these two email leak messages were placed in the sent\_test collection of the two users and then we applied the best classification-based method on them. For this test, simulated leak-recipients were added to the training set, but not to the two test messages. In the two test messages, we obviously considered, respectively, alex.perkins@enron.com and rita.wynne@enron.com as the leak-recipients. The training method is non-deterministic, since it includes cross-validation to compute the textual similarity, so we ran 100 trails and report the average performance.

The results are indicated in second column (Original) of Table 5. In addition to Prec@1, we also report Average Rank (AvgRank) as an evaluation metric. AvgRank is defined as the average value of the rank in which the true leak-recipient was listed. The minimum value of AvgRank is 1.0 (when all

predictions are correctly ranked in position 1). Larger values of AvgRank indicate worse predictions.

Leak case	Classification-based (Original)	Classification-based (Variation $\alpha = 0.2$ )
Germany-c	[0.0%, 3.7]	[0.89%, 1.11]
Kitchen-l	[0.0%, 10.9]	[0.25%, 2.50]

Table 5: Performance when Detecting Real Leak Cases. [Prec@1, Average Rank]

Performance was rather disappointing. Not only were the average ranks far from what we would hope for in a practical system, and also the Precisions@1 were 0.0 in both cases. In other words, the algorithm could not predict leaks correctly even once in 100 attempts.

This disappointing performance, when analyzed in detail, has a very simple explanation. In both cases, the two real leaks (alex.perkins@enron.com and rita.wynne@enron.com) were to recipients that had never been encountered in the previous messages, either in the sent\_train collection nor in the received collection. In contrast, recall that the simulated leak-recipients in the training set are selected from the procedure in Table 2, i.e., only email addresses from the Address Book can be selected as leak-recipients. Since email addresses that were never observed before will never be selected as leak-recipients,

it is not surprising that the learning method used cannot detect them. Clearly these email leaks did not occur as a result of incorrect selection of an address-book value from an abbreviation, as we assumed in our synthetic-data experiments.

Therefore, even though we believe the classification-based method proposed in Section 3.2 works well for predicting leaks associated with the plausible leak criteria explained in Section 2.3, it is not suited to predict leaks of the sort illustrated by *germany-c* and *kitchen-l*—i.e., leaks to email addresses not in a user’s address book. However, we will describe below, a simple variation in the leak criteria can make the classification-based method considerably robust to these types of leaks.

**4.2 Sampling from Seen and Unseen Recipients** In order to make the classification-based algorithm handle unseen leak-recipients, we applied a very simple modification to the process of selecting artificial leak-recipients.

The idea can be stated in the following way: with probability  $1 - \alpha$  the leak-recipient will be selected according to the *3g-address* leak criteria in Table 2; while with probability  $\alpha$  it will be randomly selected from a distribution of random email addresses not in the Address Book (i.e., sampling randomly from unseen email addresses).

With this small change, we created a variation of the original classification-based algorithm that should be able to learn patterns associated with seen and unseen leak-recipients. Larger values of  $\alpha$  are expected to predict unseen leak-recipients more frequently, whereas smaller values of  $\alpha$  have the opposite effect (when  $\alpha = 0$ , we have the original classification-based algorithm).

This effect can be observed in Figure 2. There, precision@1 and average rank curves are illustrated as a function of  $\alpha$  for the two real cases of leak. For *Germany-c*, values of  $\alpha$  around 10% indicate Precision@1 around 50%. When  $\alpha = 0$ , we return to the original performance values (first column of Table 5). As  $\alpha$  increases, the performance is consistently improved — for instance, Prec@1 is around 90% and Average Rank is about 1.11 for  $\alpha$  close to 20%.

The *Kitchen-l* curves in Figure 2 present a similar behavior — weaker performance numbers for small  $\alpha$  values and better performance for larger values of  $\alpha$ . It is interesting to notice that the maximum value of Precision@1 here is 0.25 and the maximum value of Average Rank is 2.5. This happened because this particular message has 4 different unseen email addresses (out of 44 recipients) and only one of these is the true leak. Therefore, the best possible result for an algorithm which relies only on past email is to choose randomly among the the four unseen addresses, i.e., to classify them as leaks with the same confidence. This is exactly what happens case when  $\alpha \geq 0.1$ , where the precision at 1 reaches 25%.

For comparison, performance results of the  $\alpha = 0.2$  variation are also illustrated in Table 5. Now we have a general method for email leak prediction that handles well both seen and unseen types of leak-recipients.

**4.3 Overall Comparison** From Table 5 and Figure 2, it is clear that the proposed variation of the classification-based method can handle unseen leak-recipients much better than the original algorithm. However, it is not obvious how this modification affects the overall performance for the task, i.e., the overall leak prediction performance in all 20 enron users.

We compare the original classification-based method ( $\alpha = 0.0$ ) to two of its variations ( $\alpha = 0.1$  and  $\alpha = 0.2$ ) in Table 6. Generally speaking, the original method presents better overall performance than its variations. As expected, it is easier to make leak predictions when unseen recipients are never considered leak-recipients. Also, as  $\alpha$  values are increased, the performance is slightly deteriorated. Notice, however, that even the results of the  $\alpha = 0.2$  variation are still better than all other baselines from Table 3.

## 5 Discussion and Related Work

Detecting email leaks is a new problem. To solve it, we proposed a new technique that has shown promising results in various tests. In our experiments, we applied leak-detection methods only to messages that actually contain a leak, and evaluated only the ability to distinguish the intended recipients of a message from the unintended recipients.

In reality, of course, most messages do not contain leaks. Thus in a real email client implementation, it would be necessary to extend our method to also determine if messages do or do not contain leaks. For instance, we could use the prediction confidence of the learning algorithm to decide whether or not the user should be warned of a potential leak, or use a secondary classifier to decide whether or a message contains a leak. We have not yet explored this issue. We note that user studies will probably be necessary to determine what level of “false positive” predictions users will tolerate. Also, from a user’s point of view, the number of false positive predictions might also be reduced not by machine learning methods, but by applying additional heuristics to estimate the severity of a possible leak—e.g., in corporate settings, the potential consequences might be worse for an email sent outside the company than an email sent within the company.

The literature overlapping privacy and email is very limited. Generally speaking, in this paper the leak prediction task was approached as a supervised outlier detection problem [5], where the normality distribution was estimated from real data but the abnormality distribution was simulated.

Boufaden et al. [1] proposed a privacy enforcement system in which information extraction techniques and domain knowledge were combined to monitor specific privacy

Enron User	$\alpha = 0.0$		$\alpha = 0.1$		$\alpha = 0.2$	
	Prec@1	AvgRank	Prec@1	AvgRank	Prec@1	AvgRank
RappB	0.788	1.471	0.753	1.458	0.747	1.459
Hernandez	0.720	1.900	0.653	2.053	0.613	2.407
Pereira	0.850	1.235	0.790	1.430	0.765	1.360
Dickson	0.786	1.214	0.700	1.300	0.718	1.282
Lavorato	0.910	1.220	0.861	1.253	0.861	1.202
Hyatt	0.824	1.202	0.792	1.244	0.770	1.265
Germany	0.665	1.601	0.679	1.598	0.669	1.542
white	0.812	1.274	0.790	1.310	0.758	1.354
whitt	0.889	1.124	0.872	1.145	0.822	1.200
zufferli	0.809	1.194	0.797	1.211	0.769	1.249
campbell	0.739	1.385	0.678	1.549	0.671	1.536
geaccone	0.789	1.411	0.755	1.525	0.755	1.509
hyvl	0.822	1.196	0.795	1.223	0.773	1.245
giron	0.858	1.188	0.806	1.254	0.782	1.313
horton	0.856	1.265	0.785	1.456	0.767	1.565
derrick	0.934	1.074	0.921	1.112	0.896	1.170
kaminski	0.902	1.129	0.880	1.160	0.886	1.152
hayslett	0.747	1.794	0.719	1.832	0.725	1.834
corman	0.912	1.095	0.866	1.146	0.839	1.177
Kitchen	0.662	3.156	0.584	3.305	0.621	2.911
Average	0.814	1.406	0.774	1.478	0.760	1.487

Table 6: Email Leak Prediction Results for Different  $\alpha$  Values

breaches via email in a university environment. They were particularly concerned with the following types of entity breaches: student names, student grades and student IDs. Using 205 manually labeled emails and tailored ontologies, they were able to correctly predict breaches with an F-score of 69.3%. Similar techniques could be used in conjunction with the methods described here to detect email leaks that are particularly harmful from a privacy point of view.

Pal & McCallum [6] addressed the “CC prediction problem”, i.e., the problem of suggesting recipients for an already composed email messages. The authors proposed different types of graphical models for the problem and provided some experimental results on a personal email collection. In some sense, the leak prediction problem can be seen as the negative counterpart of the CC prediction problem: in the latter, we want to find intended recipients, and in the former, we want to find unintended recipients. Email leak detection is a somewhat harder problem to study, since leaks are infrequent. Our main motivation for studying the email-leak detection problem, rather than CC prediction, is that the potential cost of email leaks is quite large.

## 6 Conclusions

In this work we introduced the problem of information leak prediction in email communication, in which the goal is predicting unintended message recipients. With the widespread

use of email, the accidental inclusion of unintended recipients in emails has become increasingly common. In many cases these mistakes can reveal sensitive or private information — which in turn can potentially lead to terrible consequences such as financial losses, brand damage and expensive law suits. In spite of its critical importance, this problem has received very limited attention from the research community.

We addressed this critical problem as an outlier detection task, where the unintended email addresses considered the outliers. Using simulated leak-recipients in combination with real world email data (the Enron Email corpus), we were able to create large amounts of labeled data — which in turn was used to learn typical outlier patterns. The simulated leak-recipients were created by imitating typical cases of mistakes such as misspellings of email addresses, typos, similar first/last names, etc. Using a combination of textual and social network features, the model correctly predicted leak-recipients in almost 82% of the test messages, a very promising result. Additionally, we tested the effectiveness of our approach in real cases of information leak — where a variation of the proposed method was successful in predicting two independent real information leaks from the Enron corpus.

There are several possibilities for future research from the ideas introduced in this paper. Depending on the particular email environment or email client, different leak cri-

teria can be utilized. Also, it might be possible to improve the current results using different features, or even a better learning model. Another possibility of future research lies in addressing the problem from the email server perspective: notice that a server-based method would be able to derive additional social network features.

## 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 *Conference on Email and Anti-Spam (CEAS'2005)*, 2005.
- [2] W. W. Cohen, *Enron Email Dataset Webpage*, <http://www.cs.cmu.edu/enron/>.
- [3] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks." in *IIWeb*, 2003, pp. 73–78.
- [4] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Machine Learning*, vol. 37, no. 3, pp. 277–296, 1999.
- [5] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, 2004.
- [6] C. Pal and A. McCallum, "Cc prediction with graphical models," in *Conference on Email and Anti-Spam*, 2006.
- [7] J. Shetty and J. Adibi, "Enron email dataset," USC Information Sciences Institute, Tech. Rep., 2004, available from <http://www.isi.edu/adibi/Enron/Enron.htm>.
- [8] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *22nd Annual International SIGIR*, August 1999, pp. 42–49.

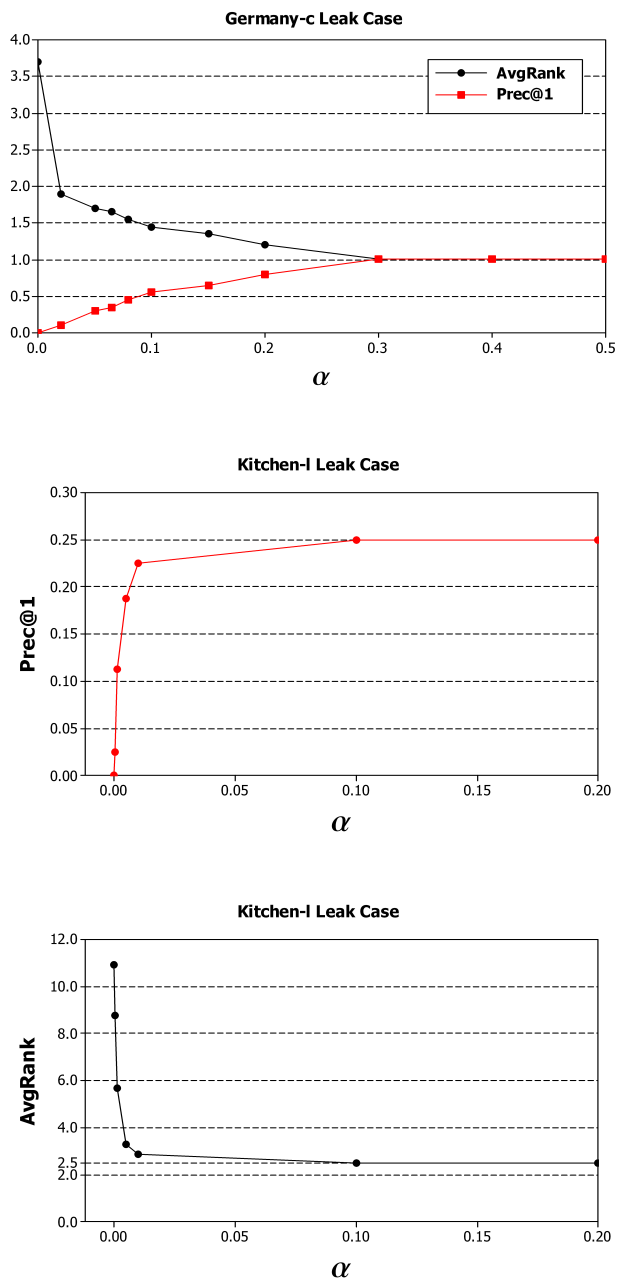


Figure 2: Performance of Real Leak Cases For Different Probabilities  $\alpha$ .