Threading Machine Generated Email

Nir Ailon
Technion & Yahoo! Research
Techion, Haifa 32000, Israel
nailon@yahoo-inc.com

Zohar S. Karnin
Yahoo! Research
MATAM, Haifa, 31905, Israel
zkarnin@yahoo-inc.com

Edo Liberty
Yahoo! Research
MATAM, Haifa, 31905, Israel
edo@yahoo-inc.com

Yoelle Maarek
Yahoo! Research
MATAM, Haifa, 31905, Israel
yoelle@ymail.com

ABSTRACT

Viewing email messages as parts of a sequence or a thread is a convenient way to quickly understand their context. Current threading techniques rely on purely syntactic methods, matching sender information, subject line, and reply/forward prefixes. As such, they are mostly limited to personal conversations. In contrast, machine-generated email, which amount, as per our experiments, to more than 60% of the overall email traffic, requires a different kind of threading that should reflect how a sequence of emails is caused by a few related user actions. For example, purchasing goods from an online store will result in a receipt or a confirmation message, which may be followed, possibly after a few days, by a shipment notification message from an express shipping service. In today's mail systems, they will not be a part of the same thread, while we believe they should. In this paper, we focus on this type of threading that we coin "causal threading". We demonstrate that, by analyzing recurring patterns over hundreds of millions of mail users, we can infer a causality relation between these two individual messages. In addition, by observing multiple causal relations over common messages, we can generate "causal threads" over a sequence of messages. The four key stages of our approach consist of: (1) identifying messages that are instances of the same email type or "template" (generated by the same machine process on the sender side) (2) building a causal graph, in which nodes correspond to email templates and edges indicate potential causal relations (3) learning a causal relation prediction function, and (4) automatically "threading" the incoming email stream. We present detailed experimental results obtained by analyzing the inboxes of 12.5 million Yahoo! Mail users, who voluntarily opted-in for such research. Supervised editorial judgments show that we can identify more than 70% (recall rate) of all "causal threads" at a precision level of 90%. In addition, for a search scenario we show that we achieve a precision close to 80% at 90% recall. We believe that supporting causal threads in email clients opens new grounds for improving both email search and browsing experiences.

General Terms

Information Systems

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications—*Electronic Email*

Keywords

Email Threading, User Experience, Algorithms, Models, Frequent sets and patterns

1. INTRODUCTION

Web mail has become one of the largest sources of user generated content today, significantly bigger than the Web. The volume of email going through Web mail services such as Windows Live Hotmail (recently re-branded as Outlook.com), Yahoo! Mail and Gmail, today's largest email providers [3], keep growing. Some analysts even predict that the number of email accounts will increase from 3.1 billion in 2011 to nearly 4.1 billion by 2015, [7].

Even if personal communications seem to increasingly move away from mail toward social media such as Facebook or instant messaging applications, especially with the younger generations [4, 15], email retains its value as an asynchronous method of communication with commercial entities or organizations. One key characteristics of these entities is that they regularly and automatically generate messages and send them to individual users.

Machine-generated messages dominate email traffic, as we have verified on Yahoo! mail, and detail later. It is also clear that the volume of machine-generated email will keep growing. Among these machine-generated messages, while some are of critical importance like an e-ticket, a car rental reservation, a bill to pay, or a new password, others are almost spammy, like hotel newsletters, or obsolete mailing list subscriptions. Most users do not take the time to clean up their inboxes, never define a single folder [10], and have to turn to the usual search and browse mechanisms when tracking important messages. One convenient mechanism to navigate one's inbox is "threads" or "conversations" that visually link related messages.

The task of *email threading* consists of identifying sequences (threads) of messages related to a single logical conversation, event or action. The most common type of email thread is a group of email messages that have been exchanged by a common list of senders and recipients via the mail reply/forward mechanisms. They can be extracted by purely syntactic methods, through the analysis of the senders and recipients in the mail headers and more importantly by observing the shared subject lines. Indeed, in these "syntactic threads" the subject lines are similar modulo the addition of the prefixes '`Re:`' for reply and '`Fw:`' for forward, possibly multiple times. Other threading header field based mechanisms were defined in RFCs 822 [5] and 2822 [13]. These suggest email clients should endow forwarded or replied messages with a special "`In-Reply-To`" field header and include the original message identifier. Clearly when such fields exist threading becomes straightforward.

This, however, happens almost solely for personal (individual or multi-party) conversations. Automated messages that are caused by a same action will not be identified as a thread. This is because separate email messages in these threads have different subject lines and content. In many cases, they do not even the share the same sender.

We argue here that it is critical to offer some kind of threading mechanism dedicated to machine-generated email simply because already today it represents the majority of the volume of email traffic. Indeed, we have analyzed the inbound (non-spam) traffic of Yahoo! mail over a period of one month, simply counting the number of messages originating from a same sender. We verified that more than 60% of the overall traffic originated from mass senders in multiple domains (manually classified via editors) such as social (facebook, linked in, etc.), commerce (ebay, amazon, etc.), finance, shopping, travel, etc. as illustrated in Figure 1. We also sampled[1] the bodies and subject lines of the mass senders' messages and verified they were clearly automatically generated, as expected. Indeed each of us can easily recognize in their inboxes these messages derived from a same boilerplate, where only user specific information such as name, address, date, purchase good, confirmation number, etc. differ. In the same Figure 1, the categories labeled "Personal", "Other", "Miscellaneous", and "Unknown" contain mostly personal conversations.

Given the mass volume of these machine-generate email messages and the fact that we can only expect it to grow in the future, we argue here that it is critical to help users naivgate it in a more convenient way. One first step in this direction is to define and offer a new type of threading that takes advantage of this mass volume in order to provide a new type of threading abstraction. We introduce in this paper the notion of "causal threading", which is a specific type of thread dedicated to machine-generated email messages.

We define a causal thread as a sequence of messages that originate from a same cause and are thus logically linked. Consider the following example of a user who clicked on a "`I forgot my password`" button on a site she is registered to. She might first get an email message entitled "`change your password`" with a link taking her to a dedicated "`change your password`" page on the site, and then a second message entitled "`your password has been changed`" containing the new password and informing her that her password has been successfully changed. Clearly these messages are strongly related, as they pertain to one action that spans several stages, yet today's threading mechanisms will not group them together as they are not seen as part of a same conversation. Moreover, most such machine-generated messages are one directional, they are inbound messaged often without a return channel[2]. Yet, we argue that these messages are part of a same logical thread as the second message is an automatic result of the action taken on the first message. In other words, as the second message can be traced back to the first message as its "cause", we want to link them into the same causal thread. Consider now this more complex example, in which a user purchased an electronic device from an online mass retailer. He will first get from the retailer a message that the order was placed, then possibly a message from the delivery service with the tracking number, and another one confirming delivery. Finally he might even get another message from the retailer thanking him and including instructions for returning the device if not satisfied. All these messages would not belong to the same syntactic thread. Indeed, not only do they not share the same subject lines (as in the previous example) but also involve different senders that were not part of a same conversation at any time. We propose here to group them in the same causal thread that will link them in a logical order to the source or cause of the thread.

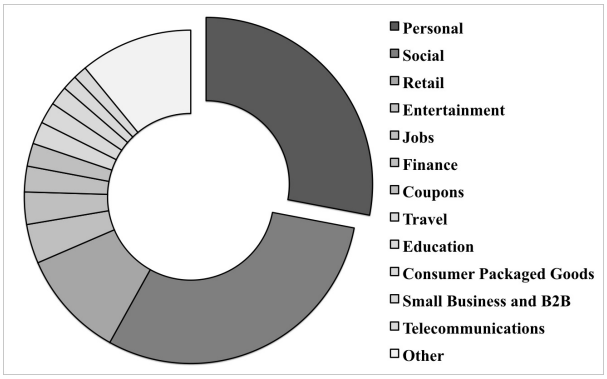

Figure 1: Distribution of email traffic per domain over a month of Yahoo! mail traffic.

Our approach is based on the intuition that since these messages are machine generated and sent over and over, modulo minor user-specific variations, to large volume of users we should be able to observe causality patterns between them.

We propose to first identify, over very large amounts of email data, these similar messages that share a common sender and variations of subject lines. We refer to these as *email templates*, and will define them more formally later on. Note that for performance we do not, at this stage, consider the body of messages.

Given these email templates that represent an abstraction of machine-generated mail, we propose to learn causality

[1]All email studies reported here that involved looking at subject lines or bodies were conducted exclusively on the inboxes of users who had voluntarily opted in for such research.

[2]In fact, many of such automated messages explicitly ask the recipient *not* to respond.

relations offline over a very large corpus of email data. Our process consists of three offline stages and one online stage:

1. We first learn email templates over a very large representative sample of email data and devise an efficient procedure that associates any new incoming message with its template, as well as return the list of variables instantiated by the message. This stage is detailed in Section 2.

2. We then learn, offline, a "causal graph" over email templates, as explained in Section 3. Given the causal graph, one can estimate the likelihood that one message follows another.

3. In addition, we learn a "causality prediction function" that takes as input a pair of email messages, produces a list of features that rely on the causal graph, and outputs a confidence score to whether the first message caused the second. This function is critical to deciding whether the second message should follow the first one in the same thread. This step relies on a supervised data set. The features produced in the step are detailed in Section 4.

4. Given the causality prediction function, we thread the incoming emails, in an online manner, by appending a newly received email to its most likely predecessor (i.e., assigning the new email to the same thread as its chosen predecessor) or decide that it is the beginning of a new thread when all candidate predecessors are given an insufficient score. We describe this process in Section 5.

The unsupervised offline process needs to be conducted on a regular basis to account for the arrival of new mass senders on the market, or template updates as service providers keep changing their infrastructures. The supervised phase of learning the confidence function need not be updated so often as it is a function of features that are independent of the exact templates. Once the templates, causal graph and confidence function are learned, actual threading can be conducted online. The causal threading process serves the inbound message stream. It associates each incoming message, at arrival time, with a thread identifier, while maximizing a utility function parametrized by the confidence score, as detailed in Section 5.

The contributions of our work are three-fold: We introduce the notion of "email template" for machine-generated email and define a new mechanism for identifying them efficiently. We propose a method for efficiently learning relations between these templates, over millions of users' inboxes, so as to infer causality. We introduce the notion of "causal threading" and propose an efficient mechanism for dynamically assembling causal threads as messages are delivered to the email system.

2. EMAIL TEMPLATES

By analyzing inbound traffic it is possible to identify these near similar messages from a same sender, and separate the boilerplate from the variables parts. The idea is to find frequent subsequences of words in email subject lines shared by many messages originating from a same sender. For example, an email sent by "`usps.com`" whose subject line is "`Your package number 2049862-56 is on its way`" will be rare if not unique. But, the regular expression "`Your package number * is on its way`" , where the wild-card character "`*`" can match any number of words or characters, will match a great number of subject lines in messages sent by "`usps.com`".

Following the above intuition, each machine-generated message can be mapped into a tuple formed by a static template and a list of variables. For example, consider the message e sent by "`usps.com`" whose subject line is "`Your package number 2049862-56 is on its way`". The template identifier $\tau(e)$ would be formed by concatenating the sender name and the regular expression matching best a great number of subject lines, namely "`usps.com: Your package number * is on its way`" and vars(e) would be a list containing a single element: "`2049862-56`''.

Given a stream of messages, it is an interesting computational task to identify the templates and extract the variables. We refer to this process as templating. Note that the proposed threading introduced in this paper is independent from the templating process, and should work in a similar manner with any other templating technique. The templating technique we apply here is rather simple yet effective. Templates are extracted for the most frequent senders by analyzing the subject lines of the messages originating from that sender. The probability that the subject line of a message from "`usps.com`" matches the regular expression "`Your package number * is on its way`" is higher than some small constant. Obtaining these regular expressions from raw data is standard practice in data mining. See [9, 2, 1] for examples of efficient techniques for association rule mining.

In our system, we found that removing long numbers, unique identifiers, proper names, and all other non frequent words (those whose probability is below some small constant) almost always results in the exact template. There is some subtlety in identifying either overspecified or underspecified templates but this goes beyond the scope of this paper.

In what follows, we assume that given any machine-generated mail e, one can efficiently compute both a template identifier $\tau(e)$, and a list of variable values[3] vars(e).

3. LEARNING THE CAUSALITY GRAPH

The input to the system is a set of N anonymized users $\{1, \ldots, N\}$. For each user, i, we are given a set of n_i inbound messages $\{e^i_1, \ldots, e^i_{n_i}\}$, collected between time t_{begin} and t_{end}. For clarity, e^i_j denotes the j'th email in user i's inbox. The inbound email streams are chronologically sorted, namely, $t(e^i_{j_1}) < t(e^i_{j_2})$ for all i and $j_1 < j_2$. We denote by $\tau(e^i_j)$, vars(e^i_j) and $t(e^i_j)$ the template, variables, and arrival time of email j to user i, respectively. We use $\mathcal{T}$ to denote the space of templates occuring in our data (see Section 2 for details on discovering the templates). Many messages do not fit into any template, for example, personal email. These can be ignored since they are irrelevant in the context of automated mail threading. From this point on, our notation assumes that all messages in the system match a valid template.

[3]We note that in practice, given a set $\mathcal{T}$ of templates, one can construct in linear time, a *trie*-like database that would support these types of queries in constant time.

Let $\Delta = t_{\text{end}} - t_{\text{begin}}$ denote the length of the time window in which data is collected. For each[4] $\tau \in \mathcal{T}$, define $\lambda(\tau)$ to be the average number of times template τ was observed in a single time unit:

$$\lambda(\tau) = \frac{\left|\left\{(i,j):\ i \in [N],\ j \in [n_i],\ \tau(e_j^i) = \tau\right\}\right|}{N\Delta}.$$

We posit that the number of appearances of a template τ per time unit in a stream is distributed Poisson with parameter $\lambda(\tau)$. This means that the probability of observing template τ at least k times within an interval of δ units is estimated by $\text{Poiss}(\lambda(\tau)\delta, k) = e^{-\lambda(\tau)\delta}(\lambda(\tau)\delta)^k/k!$.

In order to identify statistical relations between the appearance of two distinct templates $\tau, \tau_{\text{caus}} \in \mathcal{T}$, we define a window size parameter δ. We then count the conditional frequency of τ_{caus} given τ as follows:

$$C(\tau, \tau_{\text{caus}}) = \Big|\{(i, j_1, j_2) \quad : \quad i \in [N],\ j_1 < j_2,\ t(e_{j_2}^i) < t(e_{j_1}^i) + \delta \le t_{\text{end}},\ \tau(e_{j_1}^i) = \tau_{\text{caus}},\ \tau(e_{j_2}^i) = \tau\Big\}\Big|.$$

In words, $C(\tau, \tau_{\text{caus}})$ counts the number of times templates τ and τ_{caus} appeared within δ time units in one user's stream, where τ_{caus} appears before τ. In order to infer a potential causal connection $\tau \to \tau_{\text{caus}}$ (read: τ was caused by τ_{caus}), we compare the prior probability of observing τ in an arbitrary window of length δ to the probability of observing τ in a window of length at most δ following an appearance of τ_{caus}.

Our directed, weighted causal graph $G_{\mathcal{T}} = (V_{\mathcal{T}}, E_{\mathcal{T}}, W_{\mathcal{T}} : E_{\mathcal{T}} \mapsto \mathcal{R}^+)$ is constructed as follows. Its nodes correspond to templates $V_{\mathcal{T}} = \mathcal{T}$. The weight function is in fact defined for any pair of vertices:

$$W_{\mathcal{T}}(\tau, \tau_{\text{caus}}) = \frac{\Pr[\tau \text{ appears in the } \delta\text{−window after } \tau_{\text{caus}}]}{\Pr[\tau \text{ appears in a } \delta\text{−window}]}$$

$$= \frac{C(\tau, \tau_{\text{caus}})/C(\tau)}{1 - e^{-\lambda(\tau)\delta}},$$

where

$$C(\tau) = \left|\left\{(i, j_1)\ :\ i \in [N],\ t(e_{j_1}^i) + \delta \le t_{\text{end}},\ \tau(e_{j_1}^i) = \tau\right\}\right|.$$

In words, $W_{\mathcal{T}}(\tau, \tau_{\text{caus}})$ is the ratio between the number of times the pair of templates τ and τ_{caus} co-appeared in a window of length δ and the expected number of times τ would appear after τ_{caus} assuming an independence between the appearance of τ and of τ_{caus} (the *null* hypothesis). We kept only arcs $\tau \to \tau_{\text{caus}}$ for which the weight $W_{\mathcal{T}}(\tau, \tau_{\text{caus}}) \ge 1$. For scalability reasons, we also restricted the out degree of each vertex to be at most 20.

3.1 Causality Graph: Construction and Interpretation

We implemented the algorithm described above using email streams from 12.5 million users, over a time span of $\Delta = 2$ months. We took the window interval parameter δ to be 17 days. The set of templates, $\mathcal{T}$, was limited to a set of 11646 templates.

[4]We abuse notations and refer to τ both as a function and a template.

To remove noise factors for infrequent templates, we used a smoothing factor for the weight ratio calculation. Specifically, in the calculation of $W_{\mathcal{T}}(\tau, \tau_{\text{caus}})$, we have artificially added a constant number of appearances of τ_{caus} followed by a time window of length δ in which τ appeared exactly $\mathbb{E}$[number of appearnaces of τ in a δ−window] many times. See Figure 2 for an excerpt from the resulting causal graph.

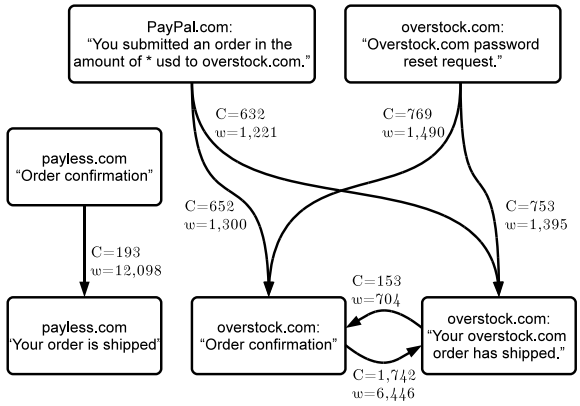

Figure 2: A snippet of the learned causal graph. Note that, for example, shipping notices tend to follow order confirmations. An interesting insight is that users who intend to purchase online often forget their password and thus a password reset precedes the purchase. This is a common structure across many vendors.

Although the causal graph is not the only ingredient in the final threading algorithm (see Section 4 below), some interesting facts can already be observed from it. If we consider the semantic meaning of templates we can identify template types which repeat across many senders. These include all password change confirmations, payment due notices, shipping alerts, service begin/end confirmation, etc. By aggregating statistical behavior on the type level we gain a lot of insights into the global flow of machine generated messages. See Figure 3 for examples for such insights.

In what follows, we assume the learned graph is fixed, and denote it by $G_{\mathcal{T}} = (V_{\mathcal{T}}, E_{\mathcal{T}}, W_{\mathcal{T}})$, where $V_{\mathcal{T}} \subseteq \mathcal{T}$, $W_{\mathcal{T}} : E_{\mathcal{T}} \mapsto \mathcal{R}^+$ denotes the arc weights, as described above.

4. PREDICTING CAUSALITY

While the end goal is to thread a sequence of email messages, a crucial building block is the ability to evaluate the likelihood that one email is caused by another. The causality relation of a pair of messages cannot be derived solely from the causality relation between their templates. Clearly, a purchase confirmation and a shipment notice a year apart are not related. We assert that the likelihood that a message e_i is caused by the message e_j is a function F of a list of features of the (ordered) tuple (e_j, e_i). In what follows we describe these features along with some insights. The function F is learned in a supervised manner; the exact learning process is explained in Section 6.

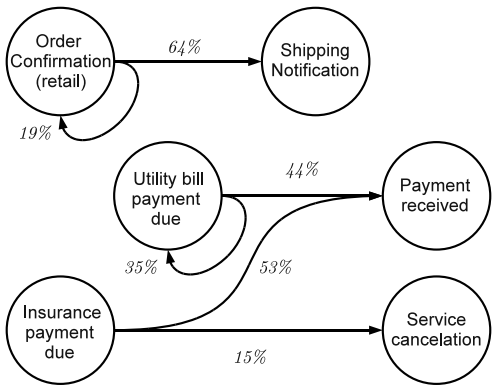

Figure 3: We present here the transition probability graph between types. The figure shows a very small portion of this graph in a classic automaton-like layout. Taking utility bill payments as an example, we see that 44% of the times they are followed by a confirmation of the payment being received. Yet, in 35% of the cases, a second payment notification is followed.

We note that for every feature described below, two features are actually created; one for an ordered pair (e_i, e_j) and another for the same pair reversed (e_j, e_i). The motivation is that a causal connection between a pair of email types is anti-symmetric while a spurious relation is in many cases symmetric.

4.1 Time Difference Features

Every arc a in the causality graph corresponds to an ordered pair of templates that tend to appear in temporal proximity. We endow each arc with the empirical mean $\mu_{\text{timediff}}(a)$ and standard deviation $\sigma_{\text{timediff}}(a)$ of the time difference between the appearance of the first and the second in the pair. When considering threading e_i and e_j with the corresponding arc $a = (\tau(e_i), \tau(e_j))$, we include the feature

$$\frac{|t(e_i) - t(e_j) - \mu_{\text{timediff}}(a)|}{\sigma_{\text{timediff}}(a)}$$

This measures the deviation of the arrival time difference between e_i and e_j from its expected value.

To further sharpen this feature, we created another cleaner version thereof in which the calculation of $\mu_{\text{timediff}}(a)$ and $\sigma_{\text{timediff}}(a)$ excluded the top and bottom 10 percentiles of the observed time differences. This cleanup was designed to reduce the effect of the following scenario occurring in the causality graph creation step. Consider a user who changed her password twice within two weeks. Changing a password usually includes a thread of two messages with templates in the spirit of 'how to change your password' and 'your password has been changed'. These two messages tend to arrive at very close time intervals. Forgetful users who change their password, say, twice in two weeks, are considered outliers for the purpose of computing the time difference. Our cleanup method is likely to avoid such noise.

In addition to the time difference information, we also computed edge weights as described in Section 3, using 4 different values of the time interval parameter δ. The different values were 4 minutes, 1 hour, 1 day and 17 days. The rationale for computing the different weight variants was that causality relations between templates happens at varying time resolutions. Note that we do not create a new graph for the different values of δ. The set of arcs is still determined by $\delta = 17$ days (as explained in Sections 3 and 3.1). We simply endow the arcs with multiple weights.

4.2 Variable Match Features

A variable match may be very significant when determining the causality relation between two emails. For example, in template pairs of the type: '`order #number# confirmation`' and '`shipment for order #number#`', a match in the variable '`#number#`' is very significant. On the other hand, there are cases where a variable match is not as significant. Consider the two templates: '`the itinerary of your flight from #location1# to #location2#`' and '`changes in your flight from #location1# to #location2#`'. A match in only one of the location variables does not mean that the two emails are connected. It is not unlikely for the two emails to discuss different flights while '`#location1#`' is simply the user's city of residence.

We introduce variable match features into threading decision making as follows. Consider an arc $a = (\tau_1, \tau_2)$ in the causality graph, and assume that templates τ_1 and τ_2 are endowed with non-empty variable lists.

In the causality graph learning step, given an instance of τ_1 and τ_2 appearing within time interval δ, the corresponding *variable match pattern* is a bipartite graph with the variables of τ_1 on the left, the variables of τ_2 on the right, and an edge between two variables if their value is identical in the two corresponding emails. For each possible variable match pattern M, we compute a weight that is defined as $W_{\mathcal{T}}(a)$, except that only occurrences of τ_1, τ_2 with variable match pattern M are counted.

The feature we output for messages e_1, e_2 with templates τ_1, τ_2 is the weight of the corresponding variable match pattern. Additionally, we provide a binary feature indicating whether the variable match pattern contains at least one variable match.

4.2.1 Matching Variables to Sender Domain Names

An additional type of match can occur between the variables of the template corresponding to one email and the domain of the sender of the other. Here, an example where this connection is meaningful for our purpose. One message from '`racingbuy.com`' with subject '`Your order confirmation`', the other email from '`paypal.com`' with subject '`Your purchase from racing buy`'. The corresponding feature we introduced in the system is a textual measure of similarity between the sender domain name of one email and a variable of the other. In case more than one variable exists, the maximal similarity measure is chosen as the feature.

4.3 Periodicity Features

We say that a template is periodic if its instances appear periodically in users' inbound mailboxes. Common examples include: daily quotes, weekly newsletter or monthly bills. In such cases, a false causality arc might be created due to *spurious relationships*. In statistics, two events have a *spurious relationship* when neither causes the other, but both are caused by a third event. In our case, the third event is e.g.

the beginning of a month. We elaborate on spurious relationships and the problem they present in Section A. We add features indicating periodicity as follows. For each template τ we compute both the average and the standard deviation of the *time difference* between its consecutive appearances in users' email streams. Denote these statistics by period_av(τ) and period_std(τ). For each causality graph arc (τ_1, τ_2), we add log(period_av(τ_i)) and log(period_std(τ_i)) for $i = 1, 2$ as features, totaling 4 numerical features. Note that we use logarithms because the important information is the ratio between the standard deviation and the mean, which is captured by difference of logarithms, which in turn, can be captured by a linear classifier.

5. SERVING AN INBOUND MAIL STREAM: THREADING

After a template graph is created offline and the causality prediction function is learned, the system is ready to provide online threading service to the email stream. The task is to process each of the user's incoming emails one at a time and decide whether it continues an existing thread or starts a new one. Fix a user, and let $\{e_1, e_2, \dots\}$ denote the stream of messages arriving into her inbox[5].

We now think of the stream $\{e_1, e_2, \dots\}$ of inbound messages as a vertex set $V_{\mathcal{E}}$ of a sequentially revealed graph. There is also a special vertex $e_0 \in V_{\mathcal{E}}$ which corresponds to no email. In this notation, the goal is to endow $V_{\mathcal{E}}$ with a set of arcs $E_{\mathcal{E}}$. When node e_i is revealed the algorithm must output a single arc (e_i, e_j) for some $0 \le j < i$. We think of e_j as being the parent of e_i and denote it by $e_j = \text{par}(e_i)$. The choice must meet the following conditions. Either $j = 0$ in which case we say that e_i starts a new thread. If $j > 0$ then it must be the case where $(\tau(e_i), \tau(e_j)) \in E_{\mathcal{T}}$ and $0 < t(e_i) - t(e_j) \le \delta$ where δ is the same window size parameter used in Section 3. In this case we say that e_i continues a thread by being appended to $e_j = \text{par}(e_i)$.

We refer to the process of selecting the arc $(e_i, \text{par}(e_i))$ upon arrival of e_i as *threading*. Note that there may be cases where $\text{par}(e_{i_1}) = \text{par}(e_{i_2}) > 0$ for some $i_1 \neq i_2$. This means that we are allowing threads to *split*. There is a good reason for allowing splitting, an example being that of *piecemeal shopping cart delivery*: A user purchases a cartload of goods from an aggregate vendor. The vendor sends a confirmation and receipt message for the entire purchase. Then the products are processed and shipped by separate sub-vendors. Each of those will result in a separate thread of email notifications. Naturally, those threads continue (or are caused by) the original online purchase.

5.1 Optimal Log-Likelihood Threading

Let (e_i, e_j) be a pair of email messages. We abuse notations and denote by $F(e_i, e_j)$ the score given by the function F (as defined in Section 4) when given as input the features extracted from (e_i, e_j). We consider $F(e_i, e_j)$ as an estimated probability for e_j being the parent of e_i. Define the score of a threading output at step n, i.e. after handling n emails, as

$$\sum_{i \in [n]: \text{par}(e_i) \neq e_0} -\log\left(F\left(e_i, \text{par}(e_i)\right)\right) \ .$$

[5]Since in this stage we deal with only one user we omit the email superscript indicating the user index.

By our construction, maximizing the score is equivalent to maximizing the log-likelihood of the chosen threading given a pairwise independence statistical model on email messages. Moreover, the following greedy threading algorithm always ensures a maximal score. Upon arrival of e_i, choose $\text{par}(e_i)$ to be

$$\text{par}(e_i) = \text{argmax}_{0 \le j < i} F(e_i, e_j) \tag{5.1}$$

We define $F(e_i, e_0)$ as the probability of e_i being the first member in its thread, which is equal to

$$\prod_{0 < j < i} (1 - F(e_i, e_j))$$

assuming pairwise independence. Note that had we not allowed threads to split, this greedy online step would not have ensured optimality. Another important note is that eventually our threading process does not completely follow the maximum log-likelihood model, but rather appends the newly arrived message to an existing thread only when the probability corresponding to the best candidate predecessor exceeds some threshold (a parameter of the model). The main reason for using a threshold-based threading rather than the maximum log-likelihood is that the regime of the problem dictates a need for very large precision rates; a requirement that is not necessarily provided by the log-likelihood model yet is provided by the latter model.

6. EXPERIMENTS AND RESULTS

6.1 Labeled Data

We obtained editorially annotated thread information for a dataset consisting of 10,606 examples. An example is an inbound email e along with a set of email messages $e_1, \dots, e_m$ received prior to e in the same mailbox which consists of candidate *causes* of e. We refer to e as the *target* email message, and to $e_1, \dots, e_m$ as *candidate email messages*.

The editorial examples were randomly collected from the Yahoo! inbound mail stream only from users who voluntarily opted-in specifically to allow such research to be performed. From each such user, only a handful of machine-generated messages that matched our templates were picked. More specifically, for a randomly chosen message e, the set of candidate messages taken from that user are those suspected of causing e, namely, those for which $(\tau(e), \tau(e_i)) \in E_{\mathcal{T}}$. For each message (either target or candidate), only the following information was made available to Yahoo! employed editors: sender, obfuscated subject line, and delivery time. Subject line obfuscation garbled identifiable information such as: names, numbers, dates, places, etc. (appearing as template variables). For each target message, the editors were asked to indicate which (if any) of the candidate messages was the cause of the target message as previously defined. The instructions that were given to editors are listed below.

We collected a total of 10,606 such examples, with a total of 13,701 candidate email messages. In other words, each target email message had an average number of roughly 1.3 candidates. Since "No preceding email" was also an option, each instance gave rise to an average of roughly 2.3 options to choose from. The small number of candidates is critical for modern email services to be able to make this choice at delivery time, for each and every inbound message. Out of the 10,606 instances 4,504 labels indicated a thread contin-

Instructions given to editors to label data:
You are given a target (input) email message and a handful of other email messages that arrived shortly before it. Please mark *at most one message* that is causally related to the target (input) email message or naturally precedes it and was caused by a common action.

1. If more than one preceding email exists mark the most recent one.
2. If no causally related preceding email exists mark 'No preceding email'.
3. If there is no way to make a valid judgment, e.g., foreign language, missing information, etc. mark 'No Judgment' (NJ).

target		
From	**Subject line**	**Time Stamp**
chegg.com	your returned chegg book has been processed	6pm 5/24
candidates		
From	**Subject line**	**Time Difference**
chegg.com	returning your chegg textbook	8 hours prior
	No preceding email	
	NJ	

Table 1: A simple editorial example. The editors were asked to indicate whether the target email should or should not be appended to the candidate email to create (or continue) a thread.

target		
From	**Subject line**	**Time Stamp**
ebay.com	your invoice for ebay purchases apple iphone 3gs 8gb black smartphone 170730882108	7pm 4/9
candidates		
From	**Subject line**	**Time Difference**
ebay.com	updates for your purchase from thegeex apr 07 12	1 hour prior
ebay.com	your ebay item sold apple iphone 3gs 8gb black smartphone 170730882108	1 day prior
ebay.com	ebay invoice notification for saturday march 31 2012	4 days prior
members .ebay.com	re details about item jgdsne02 sent a message about comcast arris tg262 docsis 3 modem wireless 210038603616	5 days prior
	No preceding email	
	NJ	

Table 2: An editorial example where there are a number of possible candidate emails. Here, the target email should continue the thread ending with the second candidate email. This example also shows that simply threading emails by sender and time difference cannot succeed.

uation, 4800 were indicated to be the first in their thread and 1302 were not identified (the option "NJ" was chosen).

6.2 Editorial Data Creation

The straightforward way to create the editorial data is to sample a set of email messages and for each sampled message e, output the following instance: The email message e is the target email message and the candidates $c_1, \ldots, c_n$ are all of the email messages in the same inbox, received in the appropriate time window, for which the arc $(\tau(e), \tau(c_i))$ appears in the causality graph.

This sample is problematic due to the following reason. The number of candidates per email averages at 6.05 meaning that even if all of the instances were positive, the number of positive pairs is only a sixth of the data. We sampled 200 instances without the mentioned filtering and found that 39 (19.5%) were positive meaning that the estimated percentage of positive examples is slightly more than 3%. Due to this skewness, it is not practical to obtain sufficiently many editorially labeled positive examples and we must use some unsupervised initial filtering. An added bonus of the quick filtering process is that the drop in the average number of instances (from 6.05 to 1.3) translates into a more efficient classifier, which is very important in a large scale system such as an email service.

Our unsupervised filtering is done as follows: A possible candidate c for an email message e is removed if either the weight of the corresponding arc $(\tau(e), \tau(c))$ is smaller than 12 or the ratio between its weight and that of the inverse arc $(\tau(c), \tau(e))$ is smaller than 3. To assess the cost of this processing to the recall, we sampled 200 instances without the mentioned filtering and found (via manual labeling of editors) that in 24 of these instances, the correct answer was removed in the unsupervised filtering stage. It follows that our preprocessing stage yields an approximate multiplicative penalty of 0.88 to the recall rate. Notice that the precision can also be affected by this filtering. This may occur when the true candidate was filtered yet the classifier predicted a different parent. Though theoretically possible, none of the 200 labeled examples emitted such an error, meaning that the penalty for the precision is negligible.

6.3 The Learning Model

We divided our labeled data into training, validation and test portions. Their respective sizes were set to 60, 20 and 20 percent (of instances). The trained model is a real function mapping between pairs of email messages (a target e and a candidate e') and a confidence score. The higher the confidence is, the stronger our belief that e' is the actual parent of e in the same causal thread. Specifically, given a new email message e and a candidate parent e', we compute the vector valued feature extraction function described in Section 4 $\bar{f}(e, e') = (f_1, f_2, \ldots)$, and compose the result with a classifier given by a learning system (described below).

Notice that the editors did not provide labels per a pair

of target e and candidate e', but rather a choice of one or no candidate from a list. To use this information for learning, each instance of target e and candidates $e_1, \ldots, e_m$ was converted into m pairs $(e, e_1), \ldots, (e, e_m)$. If an editor chose e_i as the actual parent of e, then (e, e_i) was taken as a positive instance, and $(e, e_1), \ldots, (e, e_{i-1}), (e, e_{i+1}), \ldots, (e, e_m)$ were taken as negative ones. If the editor marked no parent (e.g. that e begins a new thread) then all m pairs were taken as negative instances. We trained an AD tree classifier [6]. We also considered other classifiers such as logistic regression; while their results are comparable, AD trees performed slightly better.

Since our conversion method produced a skewed data set of significantly more negative examples than positive ones, we introduced a parameter $\alpha_1 > 0$ and gave each positive instance a weight of $\alpha_1 \cdot m$ where m is the number of candidate for that instance. The intuition for the m factor is two-fold. First, it results in a (roughly) balanced weight of positive and negative examples. Second, a candidate that tends to have a lot of competition (i.e. many other candidates) would intuitively require a higher score to be chosen by the classifier than a candidate that tends to have zero or little competition. We also distinguished between two types of negative instances. The first type results from an instance in which the new email began a new thread (none of the candidates was chosen as the actual parent). The second type is a pair (e, e') of target e and a candidate e' distinct from the actual candidate e'' the editor marked. The latter example should intuitively have a smaller weight as it could be the case that a candidate e' is a moderately fine choice for a parent of e but a better candidate was found. Negative examples of the latter type received a weight of α_2, where $\alpha_2 > 0$ is another tuning parameter, while the first type was given a unit weight.

The testing and validation were performed as follows. Denote the learned function mapping a feature vector to a confidence by h. Given a target e and candidates $e_1, \ldots, e_m$ we computed $h(\bar{f}(e, e_1)), \ldots, h(\bar{f}(e, e_m))$ and identified $e^* = \text{argmax}_{e' \in \{e_1, \ldots, e_m\}} h(\bar{f}(e, e'))$. We also introduced another threshold parameter $\beta > 0$. If the confidence $h(\bar{f}(e, e^*))$ was greater than β, we predicted a causal relation, i.e. that e^* was the actual parent of e. Otherwise we predicted that e began a new thread. A correct prediction for an instance is one where we predict the same parent option as the editor. We measured the performance by computing the precision-recall plot for the actual positive examples, i.e. the examples in which the editorials marked that e did not begin a new thread.

The parameters $\alpha_1, \alpha_2, \beta$ where chosen to optimize the $F_{0.5}$ measure w.r.t. the stated precision-recall over the validation set. We used $F_{0.5}$ rather than the F_1 score since in our setting, low recall is more acceptable than low precision. Once we determined the parameters, we ran the test on the test portion of the data with various values of β to obtain the precision-recall plots in figure 4.

6.4 Baseline: Causal Graph Independent Features

To measure the amount of information hiding in the causal graph, we compared our results to the following baseline. For each pair of messages, we omit all features that depended on the causal graph in any way. Specifically, we left only two features; the domain similarity and time difference. We mention that this baseline still somewhat depends on the graph because the graph edge set was used in the first place for selecting editorial instances. Nevertheless, the algorithm using the complete set of features is significantly superior to the baseline as seen in figure 4. Specifically, by using the additional graph related features we increase (absolutely, not relatively) the recall rate by 15%-20%. Moreover, this is achieved in the range of the precision values, which is large enough to be presented to users.

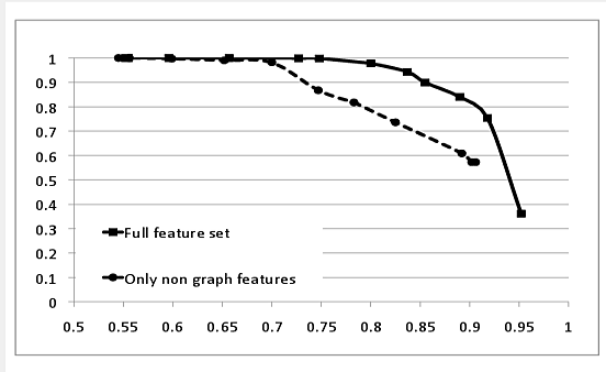

Figure 4: Precision (on the x-axis) versus recall (y-axis) curves for the threading process and baseline. As can be seen from the plot, features extracted from the template causality graph are extremely helpful. From a practical standpoint, such results can already be presented to users with $\sim 90\%$ precision at $\sim 80\%$ recall. Notice that due to the preliminary filtering of candidates, the true recall is a product of 0.88

7. RELATED WORK

Our work intersects several lines of research. The first typically known as *association rule mining* (see for example [1, 8]) refers to the problem of identifying statistical dependence between sets of items appearing in logical shopping carts. Such rules allow the vendor to apply sophisticated pricing and discounting strategies. Our work differs from this line of work in that no logical shopping cart exists here. The stream of inbound emails for a user, for our purpose, is an infinite object revealed over time. The association rules we mine have a temporal nature in that we search for repetitive patterns over relatively small time windows. Additionally, the rules are not symmetric in the sense that the relation 'caused by' is anti-symmetric as opposed to 'correlated with', which is the relation of interest in the case of logical shopping carts.

Another relevant line of research is that of identifying semantic threads based on statistical text similarity [16, 11, 12, 17]. This problem has been studied both in the context of email and newsgroups, where threading can be offered to the end user as a valuable feature for navigating through a large corpus.

Yeh et. al [17], in addition to using semantic tools to identify threads, report their study on identifying threads using payloads planted in a protocol used by a propietary mail client. Such information allows very precise thread identification, but the approach is limited to identification of

threads in which all parties use this proprietary protocol.

Our work is different in multiple ways. First, our threads define logical connections between inbound messages *only*, whereas semantic threading uses bidirectional information towards identifying so called conversations between two (or more) parties. Outbound messages are not used in our scenario, and often do not exist. Additionally, we seek to predict association rules to within almost the same precision as the trivial tag based thread prediction, while maintaining a reasonable recall rate.

To the best of our knowledge, this is the first work, dealing with causal threading of automated email messages. The only previous work we can compare ourselves to is that on threading of personal email messages, in other words, of conversations. The comparison is difficult because the problems are almost completely different. We still make the following note of comparison. In both [11] and [12], a single message e was given to a threading algorithm as "query", and the algorithm outputted a ranked list of messages suspected as belonging to the same thread as e. Both [11] and [12] evaluate their system over a set of query messages known a priori to belong to a thread of size *more than one*. Hence it was known (by experiment design) that e was associated to another message, the question was only *which*. Assuming the first message on the retrieved list was taken as the suggested related message, the algorithm in [12] succeeded in around 35% of the times. The algorithm in [11] claimed precision of 71% using a similar evaluation method (note that the datasets were different). It is difficult to compare these result with ours, because our algorithm was measured also against messages that were not associated with any other message by way of threading. Quoted from [11]: *"Of course, these figures are for messages that are known to have a parent message. An operational system would need not only to distinguish among potential parents, but also to detect whether or not the message had a parent at all"*. Such detection is performed by neither [11] nor [12]. Nevertheless, our proposed system clearly outperforms these benchmarks.

8. CONCLUSION

In this paper, we introduced a method for building a new type of thread, coined "causal thread", specially adapted to machine-generated email. Our approach leverages the fact that machine-generate messages are typically sent by mass senders to a very large number of users. By automatically learning causality patterns, offline, across millions of users, we could develop mechanisms that allow to identify causal threads at delivery time.

It is important to stress that our solution is fully scalable and can be adopted by large scale email service providers. First, our threading procedure is very efficient and can be integrated into the email delivery pipeline without causing delays or bottlenecks. Second, the entire process is language independent. This is crucial for large email providers since they typically support numerous different languages. The only seemingly language-dependent portion of our process is the editorial judgment. Bear in mind, however, that the features used to predict causality are language independent. Moreover, the exact same kind of training data is naturally generated by users when they, for example, indicate a threading mistake.

Furthermore, our process identifies non trivial threads spanning different senders, possibly over long periods of time. This, while still correctly dealing with *interleaving* threads, which occur, for example, when a user makes multiple independent purchases almost at the same time, possibly from the same vendor.

Finally, we achieve high threading precision, while delivering a reasonable recall rate. This is crucial for a positive user experience while browsing email. For the email search scenario, our system allows extremely high recall rate (almost 90%) while maintaining a precision of close to 80%.

9. REFERENCES

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216, 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Mark Brownlow. Email and web mail statistics. *Email Marketing Reports*, October 2011. `http://www.email-marketing-reports.com/metrics/email-statistics.htm`.

[4] comScore Inc. The 2010 US digital year in review. comScore Whitepaper, February 2011. `http://www.comscore.com/Press_Events/Presentations_Whitepapers/2011/2010_US_Digital_Year_in_Review`.

[5] D. Crocker. Standard for the format off arpa internet text messages, August 1982.

[6] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *ICML*, pages 124–133, 1999.

[7] The Radicati Group. Email statistics report, 2011-2015, May 2011. `http://www.radicati.com/?p=7269`.

[8] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[9] Ruoming Jin and Gagan Agrawal. An algorithm for in-core frequent itemset mining on streaming data. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 210–217, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Yehuda Koren, Edo Liberty, Yoelle Maarek, and Roman Sandler. Automatically tagging email by leveraging other users' folders. In *KDD*, pages 913–921, 2011.

[11] David D. Lewis and Kimberly A. Knowles. Threading electronic mail: a preliminary study. *Inf. Process. Manage.*, 33:209–217, March 1997.

[12] Einat Minkov, William W. Cohen, and Andrew Y. Ng. Contextual search and name disambiguation in email using graphs. In *Proceedings of the 29th annual international ACM SIGIR conference*, SIGIR '06, pages 27–34. ACM, 2006.

[13] Ed. P. Resnick. Internet message format, 2001.

[14] George Rebane and Judea Pearl. The recovery of causal poly-trees from statistical data. *Int. J. Approx. Reasoning*, pages 1–1, 1988.

[15] Alexia Tsotsis. Comscore says you don't got mail: Web email usage declines, 59% among teens. *TechCrunch*, Feb 2011.

[16] Yi-Chia Wang, Mahesh Joshi, William Cohen, and Carolyn Rosé. Recovering implicit thread structure in newsgroup style conversations. In *Proceedings of the 2nd International Conference on Weblogs and Social Media (ICWSM II)*, 2008.

[17] Jen-Yuan Yeh and Aaron Harnly. Email thread reassembly using similarity matching. In *Proceedings of Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2006.

APPENDIX

A. SPURIOUS RELATIONSHIPS

The goal we set out to achieve in this reseasrch was to identify *strong causality* between emails. By this we mean that we are interested in identifying threads of inbound automatically generated messages, emanating from a single event driven by the user. The method used for identifying these causal relations are purely statistical. A well known problem with this approach [14] is the following. Consider three events X, Y and Z and the following two cases: (i), event X causes Z (ii), event Y causes both X and Z. In the second case X and Z are not related by a *causal relationship* but rather by a *spurious relationship*. If Y is not observed, then both cases are indistinguishable, because the pair (X, Z) appears to be statistically dependent. Thus, a false causality relation might be inferred.

The main source of false causality in our data are *monthly recurring emails*. Credit card companies usually send a report to the cardholder on the same day monthly (e.g. every first of the month). Many users hold cards from more than one credit card company, and many hold cards from the same two major credit card companies. Hence, it may appear that one credit card statement causes the other. In reality these emails are both caused by an unobserved event, namely, the beginning of the month. It is easy to eliminate these false effects by identifying monthly recurring emails, and forcing them *not* to continue threads. (Starting threads is still allowed and useful: Often a "*thank you for your payment*" template appears shortly after the bill arrives and continues the thread.)

A slightly different type of statistical phenomenon one should be aware of is known as frequent item sets. This problem is well known among data miners who analyze purchase data at supermarkets (either traditional or online). The idea there is to identify pairs (or tuples) of items that are bought together in the same cart with some noticeable statistical significance. This allows the store to apply sophisticated pricing and discounting strategies. We refer the reader to the survey [1, 8]. For the purpose of this work, we do not view pairs of frequent actions as threads. Consider the following example: Users who order streaming movies from online providers are more likely to place an online order for pizza (from a different vendor) shortly thereafter. Everyone would agree that the relationship between the action of ordering the movie and ordering the pizza should not be considered causal for the purpose of threading the corresponding emails. We have noticed that the weight assigned to such connections in our data, albeit high with strong statistical significance, is well below that of arcs corresponding to actual thread connections. Careful thresholding of arc weights in $G_{\mathcal{T}}$ typically eliminates this problem.