Automated Email Activity Management: An Unsupervised Learning Approach

Nicholas Kushmerick
University College Dublin, Ireland
nick@ucd.ie

Tessa Lau
IBM T.J. Watson Research Center, USA
tessalau@us.ibm.com

ABSTRACT

Many structured activities are managed by email. For instance, a consumer purchasing an item from an e-commerce vendor may receive a message confirming the order, a warning of a delay, and then a shipment notification. Existing email clients do not understand this structure, forcing users to manage their activities by sifting through lists of messages. As a first step to developing email applications that provide high-level support for structured activities, we consider the problem of automatically learning an activity's structure. We formalize activities as finite-state automata, where states correspond to the status of the process, and transitions represent messages sent between participants. We propose several unsupervised machine learning algorithms in this context, and evaluate them on a collection of e-commerce email.

Categories and Subject Descriptors

H.5 [**Information Interfaces and Presentation**]: Misc.

General Terms

Algorithms, Experimentation, Management

Keywords

Activity management, email, machine learning, text classification, clustering, automaton induction.

1. INTRODUCTION

Email overload is becoming a critical problem [15]. Studies have shown that email has evolved from simply a communications medium to a "habitat"—the primary interface to one's workplace, supporting tasks such as activity management, meeting scheduling, and file transfer [4]. Yet today's email applications are still oriented towards manipulating individual messages. Though email is increasingly used to communicate about tasks and activities, today's clients provide minimal support for managing those activities.

IUI'05, January 10–13, 2005, San Diego, California, USA.

One important class of email-based activity is participation in a structured processes or workflows. Many email messages are a manifestation of a user's participation in a business process. For instance, an employee in an organization with a centralized hiring process receives automatically-generated messages reminding her of an upcoming interview, requesting feedback on the candidate after the interview, and notifying her of the final decision. A manager receives a series of messages when his employee requests a new computer, after the request has been approved by the financial approver, and when the machine is ready for delivery. A consumer purchasing an item from an e-commerce vendor may receive messages that confirm the order, or notification of a delay or that the items have been shipped.

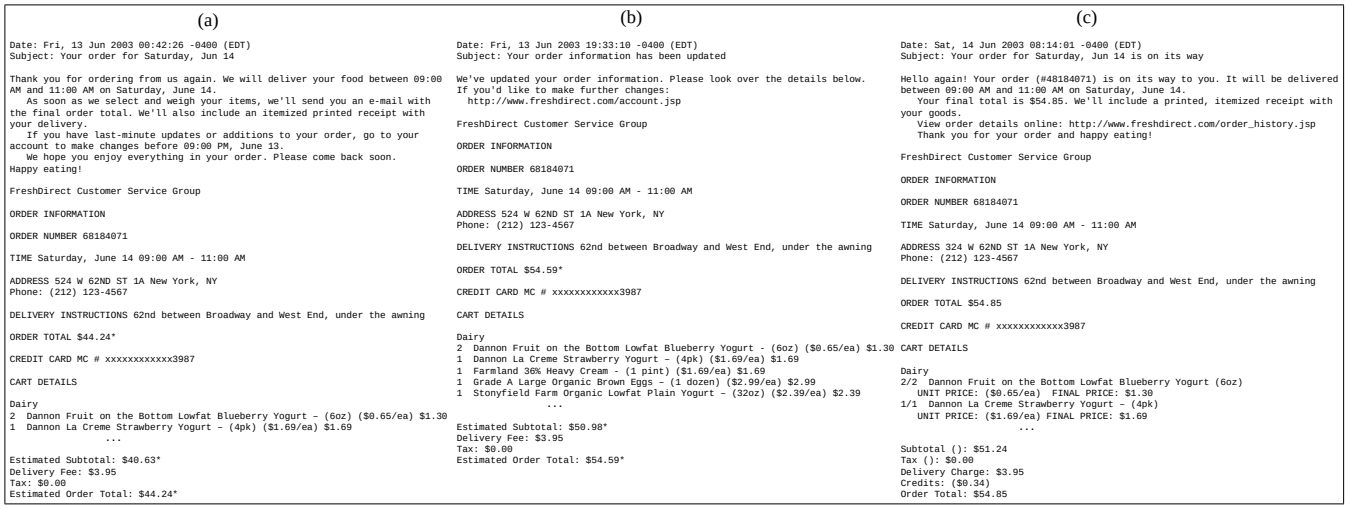
A single user may be involved with dozens of these activities simultaneously. Our goal is to provide a high-level interface to these types of structured processes, to help users manage their activities more effectively. We want to enable users to interact with their activities directly, not simply their constituent messages. For instance, a consumer should be able to quickly see how many of her e-commerce transactions are still pending. An employee should be able to easily see that no decision has been taken regarding hiring her favored candidate. Some process steps should be automated, such as sending a reply confirmation to subscribe to a mailing list.

The first step towards an activity-centric interface for email is to automatically recognize structured processes in email, and track the user's progress through these processes as new messages arrive.

We assume that a user participates in a variety of distinct classes of activities (e.g., purchases from amazon.com, auctions at ebay.com, recruitment activities with the personnel department, etc.). We formalize activities as finite-state automata called *process models.* We create a distinct process model for each type of activity (e.g., one model for amazon.com, a second model for ebay.com, a third for the personnel department, etc).

States in a process model correspond to the internal configuration or status of the process, and email messages correspond to transitions between process states. For example, an amazon.com purchase might be in an "order submitted" state; when the order is shipped, the state changes to "done" and amazon.com sends a message to the purchaser to indicate this transition.

This paper proposes several unsupervised machine learning algorithms related to this framework. Our empirical evaluation demonstrates that even without any user super-



(a)

```
Date: Fri, 13 Jun 2003 00:42:26 -0400 (EDT)
Subject: Your order for Saturday, Jun 14

Thank you for ordering from us again. We will deliver your food between 09:00
AM and 11:00 AM on Saturday, June 14.
   As soon as we select and weigh your items, we'll send you an e-mail with
the final order total. We'll also include an itemized printed receipt with
your delivery.
   If you have last-minute updates or additions to your order, go to your
account to make changes before 09:00 PM, June 13.
   We hope you enjoy everything in your order. Please come back soon.
Happy eating!

FreshDirect Customer Service Group

ORDER INFORMATION

ORDER NUMBER 68184071

TIME Saturday, June 14 09:00 AM - 11:00 AM

ADDRESS 524 W 62ND ST 1A New York, NY
Phone: (212) 123-4567

DELIVERY INSTRUCTIONS 62nd between Broadway and West End, under the awning

ORDER TOTAL $44.24*

CREDIT CARD MC # xxxxxxxxxxx3987

CART DETAILS

Dairy
2  Dannon Fruit on the Bottom Lowfat Blueberry Yogurt - (6oz) ($0.65/ea) $1.30
1  Dannon La Creme Strawberry Yogurt - (4pk) ($1.69/ea) $1.69
                ...

Estimated Subtotal: $40.63*
Delivery Fee: $3.95
Tax: $0.00
Estimated Order Total: $44.24*
```

(b)

```
Date: Fri, 13 Jun 2003 19:33:10 -0400 (EDT)
Subject: Your order information has been updated

We've updated your order information. Please look over the details below.
If you'd like to make further changes:
  http://www.freshdirect.com/account.jsp

FreshDirect Customer Service Group

ORDER INFORMATION

ORDER NUMBER 68184071

TIME Saturday, June 14 09:00 AM - 11:00 AM

ADDRESS 524 W 62ND ST 1A New York, NY
Phone: (212) 123-4567

DELIVERY INSTRUCTIONS 62nd between Broadway and West End, under the awning

ORDER TOTAL $54.59*

CREDIT CARD MC # xxxxxxxxxxx3987

CART DETAILS

Dairy
2  Dannon Fruit on the Bottom Lowfat Blueberry Yogurt - (6oz) ($0.65/ea) $1.30
1  Dannon La Creme Strawberry Yogurt - (4pk) ($1.69/ea) $1.69
1  Farmland 36% Heavy Cream - (1 pint) ($1.69/ea) $1.69
1  Grade A Large Organic Brown Eggs - (1 dozen) ($2.99/ea) $2.99
1  Stonyfield Farm Organic Lowfat Plain Yogurt - (32oz) ($2.39/ea) $2.39
                ...

Estimated Subtotal: $50.98*
Delivery Fee: $3.95
Tax: $0.00
Estimated Order Total: $54.59*
```

(c)

```
Date: Sat, 14 Jun 2003 08:14:01 -0400 (EDT)
Subject: Your order for Saturday, Jun 14 is on its way

Hello again! Your order (#48184071) is on its way to you. It will be delivered
between 09:00 AM and 11:00 AM on Saturday, June 14.
   Your final total is $54.85. We'll include a printed, itemized receipt with
your goods.
   View order details online: http://www.freshdirect.com/order_history.jsp
   Thank you for your order and happy eating!

FreshDirect Customer Service Group

ORDER INFORMATION

ORDER NUMBER 68184071

TIME Saturday, June 14 09:00 AM - 11:00 AM

ADDRESS 324 W 62ND ST 1A New York, NY
Phone: (212) 123-4567

DELIVERY INSTRUCTIONS 62nd between Broadway and West End, under the awning

ORDER TOTAL $54.85

CREDIT CARD MC # xxxxxxxxxxx3987

CART DETAILS

Dairy
2/2  Dannon Fruit on the Bottom Lowfat Blueberry Yogurt (6oz)
   UNIT PRICE: ($0.65/ea)  FINAL PRICE: $1.30
1/1  Dannon La Creme Strawberry Yogurt - (4pk)
   UNIT PRICE: ($1.69/ea) FINAL PRICE: $1.69
                ...

Subtotal (): $51.24
Tax (): $0.00
Delivery Charge: $3.95
Credits: ($0.34)
Order Total: $54.85
```

Figure 1: A transaction from the grocery store freshdirect.com contains messages that confirm (a) the initial order; (b) a modification to the order; and (c) delivery. (Messages have been anonymized and abbreviated.)

vision or labeled training data, our system is able to identify the correct state for 91% of the messages.

Before proceeding, we note that our approach is motivated primarily by either legacy or decentralized scenarios. If a process is controlled by a reconfigurable centralized workflow system, then clearly many of the challenges we address can be eliminated by modifying the system to, for example, embed machine-readable metadata in email headers. However, in many scenarios, the automated process components cannot be modified, either because they rely on legacy code whose modification is infeasible, or because the participants do not have access to these components.

The remainder of this paper is organized as follows. First, we describe the corpus of e-commerce transactions that we use to motivate our research and evaluate our results. We then describe four specific subproblems that form our core technical contribution. We then describe our approach and empirical results for each of the four subproblems. Finally, we discuss related work and opportunities for future work.

2. E-COMMERCE CORPUS

As a detailed case-study, we have investigated email activity management in the context of e-commerce transactions. We gathered a set of messages relating to a number of transactions with several retail e-commerce vendors. We then hand-labeled these messages to create a "gold standard" against which our algorithms are evaluated.

Our corpus contains messages from six vendors: half.com, eddiebauer.com., ebay.com, freshdirect.com, amazon.com and petfooddirect.com. Our corpus contains 111 messages relating to 39 transactions. As a concrete example, Fig. 1 shows an example transaction from freshdirect.com.

Our corpus does not contain a large number of transactions, but this as an asset, not a liability. Our goal is to develop technology that can be deployed to ordinary people's desktops, so we want to ensure that our learning algorithms are effective with only modest amounts of training data.

In addition to the messages, the annotation procedure involved creating a process model for each vendor. We precisely define such a model below, but for now, note that a process model consists of a set of states and a set of transitions between them, where messages are associated with state transitions. On average, each hand-crafted model contains 3.3 states and 4.3 edges. Fig. 2 shows freshdirect.com's hand-crafted process model. For example, message (a) from Fig. 1 corresponds to the edge from "start" to "orderplaced".

3. PROBLEM FORMULATION

As shown in Fig. 3, our work on addressing the general problem of automated activity management of e-mail workflows is focused on four distinct subproblems.

- **Task 1: Activity identification** is the task of partitioning a set of messages according to the activities with which they are associated. For example, in our experiments, activities corresponds to e-commerce transactions; from the dozens of messages received from freshdirect.com, the three messages in Fig. 1 would be identified as relating to the same transaction. Note that our unsupervised algorithm is not provided training data such as a sample message from each activity, or the total number of activities to be discovered.

- **Task 2: Transition identification** is the task of partitioning a set of messages according to which process model transition they correspond. For example, the algorithm would partition the freshdirect.com messages into those relating to order confirmation, order modification, etc. As with activity identification, our algorithm is given no training data.

- **Task 3: Automaton induction** is the task of automatically generating the process model. For example, given a few activities such as Fig. 1, the task is to derive the model in Fig. 2.

- **Task 4: Message classification** is the task of assigning a incoming message to its transition. For example, given the model in Fig. 2, the classification task is to assign message Fig. 1(a) to the edge from "start" to "orderplaced", etc.

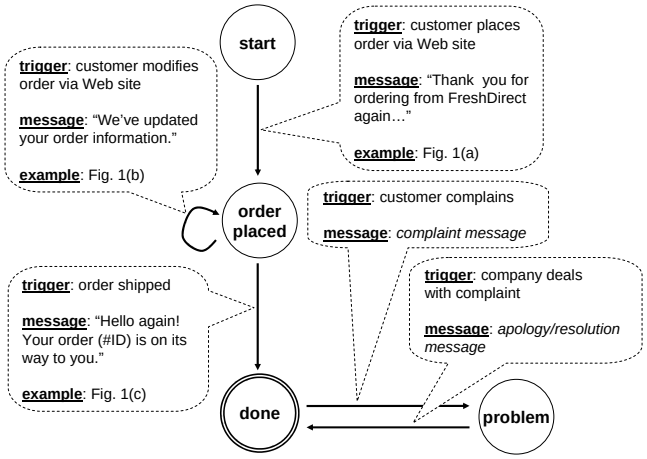


Figure 2: The process model for freshdirect.com.

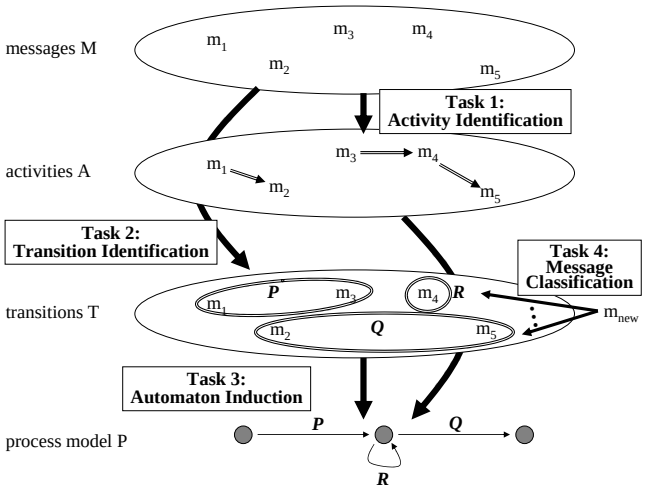


Figure 3: The four subproblems.

4. TASK 1: ACTIVITY IDENTIFICATION

The first task is to partition a set of messages relating to a single vendor into subsets, each of which represents a distinct activity. In our e-commerce scenario, the task would be to partition the messages according to their transaction. This task is complicated because transactions with a given vendor may overlap chronologically, and because activities (unlike normal email threads) usually do not have related Subject or Message-ID headers.

§ **Approach.** Fortunately, we find that most transactions are uniquely identified by some kind of alphanumeric code. For example, every Amazon purchase is assigned a distinct identifier such as "058-8847140-7311537". There are two challenges to exploiting this regularity. First, most messages contain a large number of alphanumeric tokens in addition to the actual unique identifier, but to scale to a large number of vendors we can't afford a hand-crafted pattern for recognizing each vendor's identifiers. Second, we have observed that some messages relating to a given activity do not in fact contain its unique identifier.

We address these challenges as follows. We use a rather generic regular expression for identifying plausible unique identifiers in each message; essentially we look for all sequences of alphanumeric characters, and then discard obvious mistakes such as dates and telephone or credit card numbers. The notation $U(m)$ indicates the result of applying this regular expression to m and filtering out these mistakes. The intent is that $U(m)$ will contain m's correct unique identifier, possibly with additional false positives.

To identify the activities, we first sort the messages chronologically, and then segment them into activities by repeatedly selecting the next unclaimed message, and combining it with other messages sharing a unique identifier. We allow for activities to overlap by at most K messages. Decreasing K might split some activities, while increasing K might merge some distinct activities. Our experimental results below confirm this predicted tradeoff.

The final step is to deal with the fact that some messages do not contain their activity's transaction identifier. The result is that we often get a set of activities that are correct except that these "orphan" messages are not clustered with their activity.

We solve this problem as follows. After generating a preliminary set of activities, we compute the mean μ and standard deviation σ of the interval between messages across these preliminary activities. We then merge an orphan with the previous activity if it is "close enough", which is defined to mean that the interval between the orphan and the activity is at most β standard deviations above the mean (i.e., at most $\mu + \beta\sigma$). In our experiments, we set $\beta = 1$.

The algorithm is described in detail in Fig. 4.

§ **Evaluation.** We evaluate the accuracy of IdentifyActivities by comparing its predicted partitions to the hand-crafted partitions described earlier.

To compare two partitions, we use the definition of precision and recall proposed by [7]. Each pair of messages in the predicted partition is allocated to one of four categories: a, clustered together (and should have been clustered together); b, not clustered together (but should have been clustered together); c, incorrectly clustered together; and d, correctly not clustered together. Precision is then computed as $p = a/(a+c)$, recall is $r = a/(a+b)$, and $F_1 = 2pr/(p+r)$

```
algorithm IdentifyActivities
input:      set M of messages for a given vendor
output:     set A of activities (a partition of M)
◇ step 1: incremental merge
sort M chronologically
A ← {}                                   ◇ list of discovered activities
a ← {}                                   ◇ current activity
while M ≠ {}
    if a = {} then                       ◇ start of a new activity
        remove the first message m from M
        a ← {m}
    else
        if ∃ m in the 1st K entries of M such that
                (U(m) ∩ ⋂_{m'∈a} U(m')) ≠ {} then
            remove m from M              ◇ extend current activity
            add m to a
        else
            add a to A; a ← {}           ◇ end of current activity
add a to A                               ◇ don't forget final activity
◇ step 2 (optional): merge singletons
calculate μ and σ from the intervals between messages in A
for each a ∈ A such that |a| = 1
    let m be a's message, a' be the activity prior to a, and i be
        the interval between m and the closest message in a'
    if i < μ + ασ then
        merge a and a', and remove a from A
return A
```

Figure 4: IdentifyActivities: The algorithm for partitioning a vendor's messages into activities.

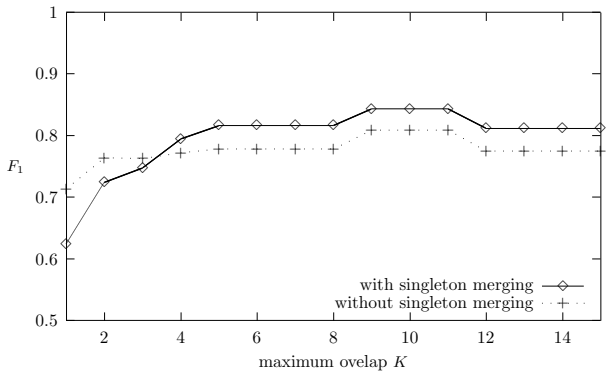


Figure 5: Evaluation of IdentifyActivities: F_1 as a function of the maximum overlap parameter K, with and without singleton merging.

is the harmonic mean of precision and recall. F_1 is always between 0 and 1, with 1 indicating a perfect clustering.

Fig. 5 shows the results for the e-commerce corpus. We plot F_1 as a function of the maximum overlap parameter K, with and without singleton merging. We see that singleton merging generally improves accuracy. For the remaining experiments we set $K = 5$. (While $K = 5$ doesn't maximize the mean F_1, it does maximize the median.)

For several of the vendors, the activity identification process involved at most one mistake. For example, one of the problematic activities involved a response from the vendor regarding a user's complaint. Neither the complaint nor the response contained the transaction identifier. The interval between the complaint and the response was 92 hours while the threshold for merging singletons was just $\mu + \alpha\sigma = 23$ hours, so the algorithm decided it was implausible that the response belonged to the preceding activity.

5. TASK 2: TRANSITION IDENTIFICATION

The second task is to partition a set of messages relating to a given vendor according to the transitions between states of the underlying process model to which they correspond. In our e-commerce setting, we need to separate messages in which the vendor acknowledges the order, from messages announcing that the order has shipped, etc. Note that the number of state transitions is not provided as input, but must be automatically discovered.

§ Approach. We treat transition identification as a clustering problem. We first define the distance between every pair of messages, and then employ a standard clustering algorithm to partition the messages.

Consider the three example messages in Fig. 6: (a) and (b) correspond to the same state transitions while (c) represents a different transition. The intent is that the distance between (a) and (b) should be much smaller than the distance between (a) and (c).

We measure distance as the negative of the length of the longest common subsequence (LCSS) between pairs of messages. For example, the LCSS between (a) and (b) is [*Subject, Thanks, for, ordering, Thank, you, for, your, recent, order, of, We, ll, notify, you, when, the, order, has, shipped*], which is much longer than the LCSS between (a) and (c).

(a)	(b)	(c)
Subject: Thanks for your order Thank you for your recent order (#123-45Q) of "Life of Pi". We'll notify you when the order has shipped.	Subject: Thanks for your order Thank you for your recent order (#129-66T) of "Vamped". We'll notify you when the order has shipped.	Subject: Your order has shipped We're writing to confirm that order #123-45Q has shipped. We hope you enjoy "Life of Pi". Thanks again!

Figure 6: Messages (a) and (b) correspond to the same process state transition and have a long LCSS, while messages (a) and (c) from the same activity have a short LCSS.

In effect, the LCSS between messages from the same transition captures the template used to generate these messages, while the LCSS between messages from the same activity will capture details of the specific transaction. Under the assumption that the template is generally longer than the activity-specific content, we can use the length of the LCSS between two messages to estimate the likelihood that messages are associated with the same transition.

Based on this message distance, we use a standard hierarchical agglomerative clustering algorithm. Initially, we compute the average distance μ across all pairs of messages, and create a distinct cluster for each message. The distance between transition clusters is defined to be the average pairwise distance between their messages. We repeatedly merge the two nearest transitions, stopping when the distance between the nearest pair exceeds μ.

In fact, this simple approach is inadequate. Recall we assume that the transition's message templates are generally longer than the activity-specific content of each message. In fact, in some cases the activity-specific content can be very large, so that the LCSS between two messages from the same activity but different transitions is longer than the LCSS between messages from the same transition. For instance, the order confirmation and shipment messages from an e-commerce grocery store may list the dozens of items purchased.

To solve this problem, we start by never clustering messages from the same activity. However, this heuristic is too conservative: it is quite common for an activity to transition between the same states multiple times. Therefore, we modify this initial clustering using a revised distance between messages. We assign scores to terms, and the revised distance between messages m_1 and m_2 is the sum of the score of every term in their LCSS. When m_1 and m_2 belong to different activities, this term score is 1—i.e., the distance for messages in different activities is unchanged.

On the other hand, if m_1 and m_2 belong to the same activity, then the term scores are calculated as follows. First, a TFIDF-like weight is computed for each term. These weights are then linearly scaled so that the term with the highest TFIDF weight is assigned a score of 0, and the term with the lowest TFIDF weight gets a score of 1. Thus, the revised distance between messages in the same activity is a weighted sum of the terms in their LCSS, where terms that usually occur in the activity get a small weight, while terms that usually occur outside the activity get a high weight.

Given this revised distance metric, we merge the G most similar pairs of clusters, where G is a user-specified parameter. Increasing G makes it less likely that transitions encountered multiple times per activity are split, but it becomes more likely that distinct transitions are merged. We

algorithm IdentifyTransitions
input: set M of messages for a given vendor, set A of activities
output: set T of process states transitions (a partition of M)
◇ *step 1: initial clustering*
let $d(m_1, m_2) = -|\text{LCSS}(m_1, m_2)|$ if messages m_1 and m_2 are from different activities in A, and $d(m_1, m_2) = \infty$ if m_1 and m_2 are from the same activity
let μ be the average value of d between all pairs of messages, excluding pairs from the same activity
let T be the result of applying average-link HAC to M with distance metric d; clustering is terminated when the distance between merged states exceeds μ
◇ *step 2: merge states with messages from the same activity*
let $w(t, a) = \text{TF}(t, a) \log(|A|/\text{DF}(t))$ be the weight of term t in activity a, where $\text{TF}(t, a)$ is the number of times that t occurs in the messages in a, and $\text{DF}(t)$ is the number of activities that contain t
let $w'(t, a) = (Z - w(t, a))/(Z - z)$, where $Z = \max_{t,a} w(t, a)$ and $z = \min_{t,a} w(t, a)$
let $d'(m_1, m_2) = -\sum_{t \in \text{LCSS}(m_1, m_2)} s(t)$, where $s(t) = 1$ if m_1 and m_2 are in different activities, and $s(t) = w'(t, a)$ if m_1 and m_2 are in the same activity a
repeat G times
  let t_1 and t_2 be the pair of transitions most similar under d'
  if $d'(t_1, t_2) < \mu$ then merge t_1 and t_2
return T

Figure 7: IdentifyTransitions: The algorithm for clustering partitioning a vendor's message into state transitions.

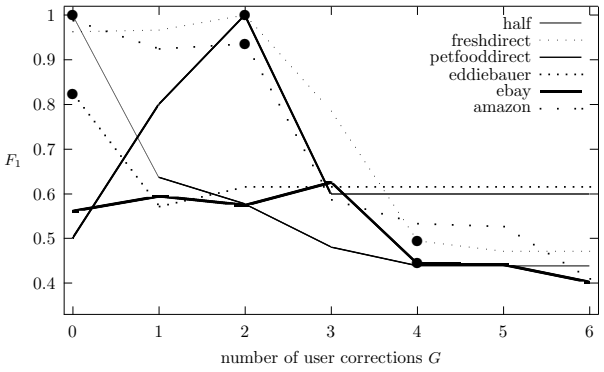


Figure 8: Evaluation of IdentifyTransitions: F_1 as function of the merge parameter G, for each vendor. The black circles indicate the automatically tuned values of G.

do not expect that a user will be able to specify G in advance. Rather, we envision a user interface that shows the set of transitions for $G = 0$, and allows the user to click on "Merge more/less!" buttons to increase/decrease G.

Fig. 7 shows the transition identification algorithm.

§ Evaluation. We evaluate the accuracy of IdentifyTransitions by comparing its predicted partitions to the hand-labeled messages described earlier. Recall that the algorithm requires as input the set A of activities. Note that we do *not* make the supervised assumption that the correct activities are known, but rather we use the possibly noisy activities discovered by IdentifyActivities.

Fig. 8 shows the F_1 score of IdentifyTransitions as a function of the merge count parameter G. The first point to note is that for five of the six vendors, G can be tuned to give nearly perfect accuracy (i.e., $F_1 \geq 0.8$, and in several cases F_1 approaches 1). Next, recall that G roughly corresponds to the amount of user intervention that is required. For half the vendors, accuracy is maximized for $G = 0$ (i.e., with no user intervention). In the remaining experiments, we fix $G = 0$.

We also investigated a technique for automatically adjusting the G parameter. We modified the algorithm to ignore G, and instead use the same termination criteria as in the initial clustering. Specifically, the algorithm continues to merge states until the revised distance d' between the merged states exceeds the average distance μ. The black circles in Fig. 8 shows value for G at which the algorithm terminated in this configuration. In half the cases, this technique was able to find the optimal value for G.

6. TASK 3: AUTOMATON INDUCTION

The final task is to discover a vendor's underlying process model. This model takes the form of a finite-state automaton, where each state corresponds to the vendor's internal state, and transitions corresponds to messages sent by the vendor to indicate state changes.

In more detail, $P = (X, s_0, F, L, T)$ is a process model, where X is the set of states, $s_0 \in X$ is the initial state, $F \subseteq X$ is the set of final states, L is the set of edge labels, and $T \subseteq X \times L \times X$ is the set of transitions. Note that the learning algorithm is responsible for discovering the correct number of states.

§ Approach. We treat the task of discovering a process model as that of learning a regular grammar from positive examples. Given a set of messages M, which are partitioned into a set of activities A and a set of transitions T, we first construct a set of positive examples E. There is one string s_a in E for each activity in $a \in A$. s_a contains one symbol for each message $m \in a$: $s_a = (t(m_1), \ldots, t(m_{|a|}))$, where $t(m)$ is a symbol indicating the transition from T with which m is associated.

In the example shown in Fig. 3, we have

$$\begin{aligned} M &= \{m_1, m_2, m_3, m_4, m_5\}, \\ A &= \{\{m_1, m_2\}, \{m_3, m_4, m_5\}\}, \text{ and} \\ T &= \{\{m_1, m_3\}, \{m_2, m_5\}, \{m_4\}\}. \end{aligned}$$

Therefore,

$$E = \{t(m_1)t(m_2), t(m_3)t(m_4)t(m_5)\} = \{PQ, PRQ\},$$

where $t(m_1) = t(m_3) = P$ is a symbol representing the messages associated with T's first transition, $t(m_2) = t(m_5) = Q$, and $t(m_4) = R$.

Following Gold's seminal work on learning regular languages [5], the problem has received substantial attention. In our experiments, we use Thollard et al's MDI algorithm [13] for learning an automaton from positive examples. MDI has been shown to be effective on a wide range of tasks, but any grammar inference algorithm could be substituted. Note that MDI learns a stochastic automaton while our process models are deterministic. It would be interesting to consider stochastic process models, but currently we convert the stochastic automaton into a deterministic one.

§ Evaluation. To measure the effectiveness of our approach, we must quantify the quality of the learned process model. To do so, we measure the agreement $\mathcal{A}$ between the learned model and the vendor's hand-crafted model. Let P be the learned model and P' be the hand-crafted model. The agreement $\mathcal{A}(P, P')$ combines of four quantities: a precision-like measure of the number of predicted transitions that are

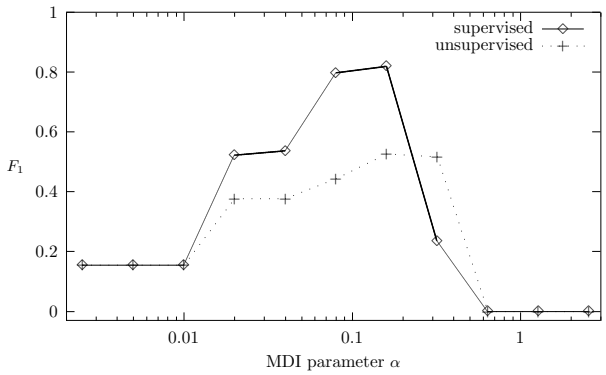


Figure 9: Agreement $\mathcal{A}$ of the learned process models, as a function of MDI's parameter α, in both supervised and unsupervised scenarios.

correct, a recall-like measure of the number of correct transitions that are predicted, and two measures of whether the initial and final states coincide. See the appendix for details.

Fig. 9 shows the agreement $\mathcal{A}$ between the learned and hand-crafted models, as a function of MDI's precision parameter α. The curve labeled *supervised* indicates that the automaton was learned from the correct activities and transitions. The *unsupervised* curve indicates that the automaton was learned from the set of activities created by IdentifyActivities and the set of transitions generated by IdentifyTransitions.

Fig. 9 indicates that the learned models have $F_1 \approx 0.8$ in the supervised setting and $F_1 \approx 0.5$ in the unsupervised setting. In fact, this is overly pessimistic. With the fixed parameter values, the algorithm does substantially better for five of the vendors but is highly inaccurate for ebay.com. By tuning the parameters, it is possible to increase performance for ebay.com at the expense of performance for the other vendors. In Fig. 9 we use the fixed set of parameters that maximizes performance over all six vendors.

As a concrete example, when trying to learn the model in Fig. 2, the algorithm generates a model with agreement $\mathcal{A} = 0.95$. The learned model is perfect (no missing states or edges) except for one spurious edge from "start" to "done".

7. TASK 4: MESSAGE CLASSIFICATION

Given a process model and an incoming message, the message classification task is to decide the transition with which the message should be associated. We solve this problem using standard supervised learning techniques.

Specifically, based on a set of state transitions T over some set of messages M, we train a classifier on M by labeling each message with its transition from T. While we use a supervised learning algorithm, our approach is actually unsupervised: the class labels (i.e., transitions) associated with the training messages are assigned automatically by the IdentifyTransitions algorithm.

We adopt a straightforward approach to text classification: We use a binary feature for every distinct term in the training corpus; the feature value is 1 for a given message if it contains the term and 0 otherwise. We do not employ any term transformation techniques such as stemming, nor any feature selection. As a learning algorithm, we use

	(a) Next state		(b) End of activity		(c) Message overlap	
	sup	unsup	sup	unsup	sup	unsup
repair	92%	92%	94%	94%	92%	86%
ignore	91%	91%	97%	97%	94%	89%

Figure 10: Ability to track each activity, under four conditions: supervised *vs.* unsupervised learning, and scenarios in which the user repairs *vs.* ignores incorrect predictions. We report accuracy in predicting (a) the next state; (b) the end of the activity; and (c) overlap between with the predicted and correct transitions' messages.

Weka's SMO support vector machine implementation with the default parameter settings.

8. PUTTING IT ALL TOGETHER

So far, we have described our approaches to the four subproblems described earlier. We now describe an experiment that measures the performance of all four algorithms in an integrated manner. Recall from the introduction that our main motivation is to provide a high-level activity-based view over a set of messages. To estimate the utility of such a view, we evaluate our ability to track activities as they unfold.

We use a leave-one-activity-out-at-a-time methodology. To measure the performance for a given vendor, we select each of the vendor's activities in turn, train our algorithms on the remaining activities, and then measure performance on the held-out activity. Our performance metrics for the vendor are averaged over each activity.

For the held-out activity, we measure our ability to predict the correct state transitions over the course of the activity, and predict that the activity has completed. Specifically, we classify each message in turn using the message classification algorithm. After each message, we compare the state predicted by our learned process model with the correct state. After the final message, we determine whether the predicted state is final in the learned model.

We evaluate our algorithms in four configurations. First, we compare a *supervised* scenario in which the correct activities and transitions are provided as input, and an *unsupervised* scenario in which the activities are created automatically by IdentifyActivities and the transitions are created by IdentifyTransitions. Second, we compare an interactive *repair* scenario in which the user intervenes to correct any mistaken state predictions over the course of the activity, and an autonomous *ignore* scenario in which prediction errors are allowed to accumulate.

Fig. 10 shows three measures of our ability to track the messages within each activity: (a) the accuracy of predicting the next state; (b) the fraction of states predicted after the last message that are in fact final states; and (c) the overlap between the messages associated with each predicted transitions compared to the messages associated with the correct transition. Measures (a) and (b) indicate how well the learned process model captures the overall structure of the activities. We envision a user interface in which a new message is described by showing messages belonging to the same transition; (c) essentially measures how accurately we can retrieve these messages.

Interestingly, the supervised scenario is not substantially more accurate than the unsupervised scenario. This sug-

gests that even though our learned models might disagree with the hand-crafted models, they are apparently good enough for making predictions about the activities. This helps to explain a second somewhat paradoxical observation, that user intervention appears to decrease accuracy. If the user (in this case, the hand-crafted model) structures the activity in a different way than the learned model, then it may do more harm than good for the user to intervene.

9. RELATED WORK

Little work has been done in the area of automatically inducing process models from sequences of email messages. Related work on improving email management falls into several categories: better visualization of email structure, task-centric interfaces on top of email, and machine learning for email classification.

It has long been recognized that people use email to manage ongoing tasks, to-do lists, and reminders, even though it was originally designed as a communications application [15, 6]. One approach to help people manage email more effectively is the ReMail system [10], which explores better visualization techniques for displaying message threads, and uses simple text analysis to extract important dates and message summaries. These visualization techniques are complementary to the automated structure induction we describe in this paper; ideas such as Thread Maps could be used to display the process models learned by our system.

Others have proposed task-centric user interfaces, such as Taskmaster [2] and TaskVista [1], which help people organize email and other online information into task-specific groupings. However, while these systems group messages together, they do so only using standard message headers. Not only does our system infer groupings that are not necessarily part of the same conversational thread, but it also automatically infers the structure of the underlying task, which gives a user more information about the relationships between messages in a task.

Machine learning has been applied to email messages, primarily for the purposes of detecting spam (e.g., [11]), predicting where to file a message (e.g., [12]), identifying related messages [9], and processing incoming messages (e.g., [8]). These efforts treat email as independent messages and ignore the larger context of email as part of ongoing activities. In contrast, we have applied machine learning to induce the structure of the activities with which a user is engaged.

There has been substantial work on the problem of learning workflow or process models from example execution logs (e.g., [14]). This work focuses mainly on our third task, inducing the process model. The inputs to our algorithms are raw messages such as those shown in Fig. 1, and we use text classification and clustering to attach meaningful labels to these messages. In contrast, a typical workflow mining system assumes these meaningful labels are available directly from the execution log. On the other hand, the workflow mining community has focused on learning much more expressive classes of process models. An important direction of future work is to replace our simple automatic induction algorithm with a more sophisticated learner.

10. CONCLUSIONS

Many structured activities are managed by email. Existing email clients have no understanding of this structure, forcing users to manage their activities by manually sifting through lists of messages. As an alternative, we envision email clients that provide high-level support for activity management. The key idea is that activities should be identified and managed as entities in their own right. To this end, we have presented an approach to the problem of automatically identifying structured activities in email, and validated our results on a corpus of email messages from an e-commerce domain.

Specifically, we make the following contributions: (1) We formalize email-based activities as finite state automata, where messages represent state transitions; (2) We specify and describe solutions to several unsupervised learning tasks in this context: activity identification, transition identification, automaton induction, and message classification; and (3) We provide empirical evidence demonstrating that our algorithms can learn process models given a small amount of unlabeled training data, and accurately update a user's state in the model as new messages arrive.

There are many directions for future work. We plan to integrate our algorithms into an existing mail client, and conduct user evaluations to determine the effectiveness of our approach. This integration will also reveal opportunities to incorporate user feedback into our unsupervised process in order to improve our system's usefulness. We also plan to investigate ways of extending the algorithms to learn process models that generalize to multiple vendors, in order to reduce the amount of training data required for the system to make useful predictions.

Acknowledgements. This research was sponsored by the IBM Dublin Software Laboratory's Center for Advanced Studies. We thank Catalina Davis, Mark Dredze, Wendy Kellog, Brian O'Donovan and Jeff Stylos for helpful discussions.

11. REFERENCES

[1] V. Bellotti, B. Dalal, N. Good, P. Flynn, D. Bobrow, and N. Ducheneaut. What a to-do: studies of task management towards the design of a personal task list manager. In *Proc. Conf. Human Factors in Computing Systems*, 2004.

[2] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: The design and evaluation of a task management centered email tool. In *Proc. Conf. Human Factors in Computing Systems*, 2003.

[3] R. Carrasco. Accurate computation of the relative entropy between stochastic regular grammars. *Theoretical Informatics and Applications*, 31(5), 1997.

[4] N. Ducheneaut and V. Bellotti. Email as habitat: An exploration of embedded personal information management. *ACM Interactions*, 8(1), 2001.

[5] E. Gold. Grammar identification in the limit. *Information and Control*, 10(5), 1967.

[6] J. Gwizdka. Reinventing the inbox – supporting the management of pending tasks in email. In *Proc. Conf. Human Factors in Computing Systems*, 2002.

[7] N. Kushmerick and A. Heß. Learning to attach semantic metadata to web services. In *Proc. Int. Semantic Web Conf.*, 2003.

[8] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7), 1994.

[9] K. Mock. An experimental framework for email categorization and management. In *Proc. Int. Conf. Research and Development in Information Retrieval*, 2001.

[10] S. Rohall, D. Gruen, P. Moody, M. Wattenberg, M. Stern, B. Kerr, B. Stachel, D. Kushal, R. Armes, and E. Wilcox. Remail: A reinvented email prototype. In *Proc. Conf. Human Factors in Computing Systems*, 2004.

[11] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *Proc. AAI-98 Workshop on Learning for Text Categorization*, 1998.

[12] R. Segal and J. Kephart. Incremental learning in SwiftFile. In *Proc. Int. Conf. Machine Learning*, 2000.

[13] F. Thollard, P. Dupont, and C. de le Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. Int. Conf. Machine Learning*, 2000.

[14] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

[15] S. Whittaker and C. Sidner. Email overloading: Exploring personal information management of email. In *Proc. Conf. Human Factors in Computing Systems*, 1996.

APPENDIX: MODEL AGREEMENT

The purpose of the agreement metric $\mathcal{A}(P, P')$ is to measure the extent to which P and P' model a given set of activities in the same way. Since a stochastic automaton induces a probability distribution over the set of all strings, it might seem that one could emply Carrasco's technique of measuring the Kullback-Leibler divergence between the learned model's distribution and that of the hand-crafted model [3]. However, this is not feasible, because the alphabets of the two models are different: there is no prior correspondence between a learned model's "gensym-ed" symbols $t(m)$ and the symbols in the hand-crafted model.

We adopt the following approach, which is based on the idea of considering all possible mappings between states and labels of the two models, and then counting the number of common transitions.

Let $P = (X, s_0, F, L, T)$ and $P' = (X', s'_0, F', L', T')$ be two process models. Without loss of generality, we assume that the models have the same number of edges and labels. (If this is not true we pad the models with dummy edges and/or labels.)

Let ϕ be a mapping from X to X', and let γ be a mapping from L to L'. In order to count how many transitions are shared by P and P', we use these mappings to convert an "P" edge into the corresponding "P'" edge. Specifically, we enumerate every possible transition $(s_i, \ell, s_j) \in X \times L \times X$, and count the frequency of the following four conditions:

$$\begin{aligned}
a &= |\{(s_i, \ell, s_j) : (s_i, \ell, s_j) \in T \wedge (\phi(s_i), \gamma(\ell), \phi(s_j)) \in T'\}| \\
b &= |\{(s_i, \ell, s_j) : (s_i, \ell, s_j) \notin T \wedge (\phi(s_i), \gamma(\ell), \phi(s_j)) \in T'\}| \\
c &= |\{(s_i, \ell, s_j) : (s_i, \ell, s_j) \in T \wedge (\phi(s_i), \gamma(\ell), \phi(s_j)) \notin T'\}| \\
d &= |\{(s_i, \ell, s_j) : (s_i, \ell, s_j) \notin T \wedge (\phi(s_i), \gamma(\ell), \phi(s_j)) \notin T'\}|
\end{aligned}$$

Note that these counts depend on ϕ and γ but for brevity we do not explicitly indicate this dependency.

The a edges are those that are shared by P and P' under the mappings (ϕ, γ). Of course, these edges agree only to the extent that the same messages are associated with these edges. For label $\ell \in L$, let $M(\ell)$ be the set of messages associated with ℓ, and let

$$a^* = \sum_{(s_i, \ell, s_j)} \frac{|M(\ell) \cap M(\gamma(\ell)))|}{|M(\ell) \cup M(\gamma(\ell))|},$$

where the sum is over the a edges shared by P and P'. The intent is that $a^* \leq a$ is the number of shared transitions, each weighted by the extent to which it is associated with the same messages in both models.

Let $p_{\phi,\gamma} = a^*/(a^* + c)$ and $r_{\phi,\gamma} = a^*/(a^* + b)$. Finally, let $I_\phi = 1$ if $\phi(s_0) = s'_0$ and 0 otherwise, and let $F_\phi = |F \cap \phi(F')|/|F \cup \phi(F')|$. Our agreement score with respect to (ϕ, γ) is the harmonic mean of these four quantities.

Since the states and labels of the learned models are just arbitrary symbols, we measure the agreement by examining each possible mapping $\phi : X \to X'$ and $\gamma : L \to L'$:

$$\mathcal{A}(P, P') = \max_{\phi,\gamma} H(p_{\phi,\gamma}, r_{\phi,\gamma}, I_\phi, F_\phi),$$

where $H(\cdot)$ indicates the harmonic mean.

Note that $\mathcal{A}(P, P') = \mathcal{A}(P', P)$. Furthermore, this function has its maximum value of 1 when the two models are identical under ϕ and γ, and has its minimum value of 0 when the two models have different initial or final states, or don't share any transitions.