BuzzTrack: Topic Detection and Tracking in Email

Gabor Cselle
ETH Zurich
Distributed Computing Group
8092 Zurich, Switzerland
mail@gaborcselle.com

Keno Albrecht
ETH Zurich
Distributed Computing Group
8092 Zurich, Switzerland
kenoa@tik.ee.ethz.ch

Roger Wattenhofer
ETH Zurich
Distributed Computing Group
8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

ABSTRACT

We present BuzzTrack, an email client extension that helps users deal with email overload. This plugin enhances the interface to present messages grouped by topic, instead of the traditional approach of organizing email in folders. We discuss a clustering algorithm that creates the topic-based grouping, and a heuristic for labeling the resulting clusters to summarize their contents. Lastly, we evaluate the clustering scheme in the context of existing work on topic detection and tracking (TDT) for news articles: Our algorithm exhibits similar performance on emails as current work on news text. We believe that BuzzTrack's organization structure, which can be obtained at no cost to the end user, will be helpful for managing the massive amounts of email that land in the inbox every day.

ACM Classification: H5.2.f. [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces. H4.3.c. [Information technology and systems applications]: Communications applications. - Electronic mail. H3.3.a. [Information storage and retrieval]: Information Search and Retrieval. - Clustering.

General terms: Algorithms, Measurement, Design

Keywords: email, organization, clustering, topic detection, topic tracking, sender behavior

INTRODUCTION

To help users deal with the growing amounts of email they receive, new structures of organization are needed. A paradigm shift is taking place: Users are now turning to fast full-text search functionality when looking for old emails in their archive. However, sensibly organizing the unstructured inbox is still a challenge. Typically, inbox data is viewed in a list sorted by arrival time: There is no sense of importance, coherence, or content.

In this paper, we address this challenge by grouping emails into topics. A topic is a cohesive stream of information that is relevant to the user – here, it consists of a number of emails which discuss or relate to the same idea, action, event, task, or question, among others. Examples for topics are sequences of emails in which a meeting is organized and the results are discussed, all emails in a newsletter, or an email exchange with a coworker about a research idea. Topics are *not* equivalent to threads. A thread consists of emails in the same reply sequence. A topic may span several threads and a thread may be distributed over several topics.

Several comparable approaches, such as task-, activity-, or priority-oriented organization schemes, exist. Our method is more general in that it covers all kinds of email-based discussions instead of just business processes or task-related items. Another difference is that we seek to complement, not replace existing email client functionality: The user's inbox stays untouched – we simply provide a view on the data, which we integrate into the email client as a plugin, shown in Figure 1.

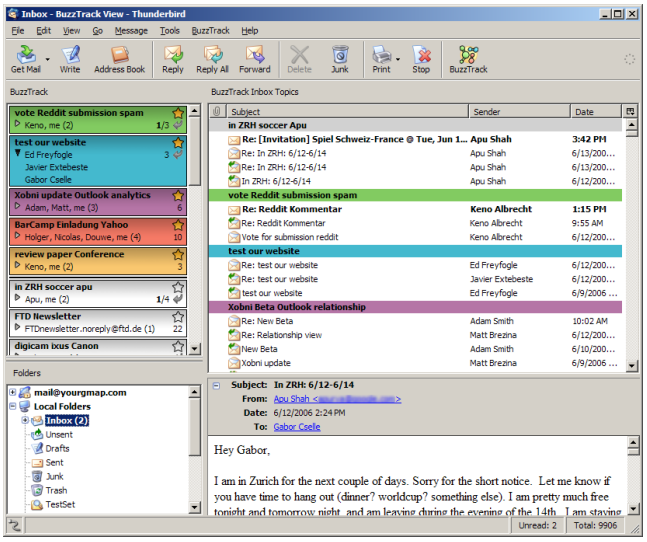

Figure 1: Inbox view with BuzzTrack plugin; topic sidebar on the left, grouped inbox view on the right.

Email organization has been stuck in the file-system-like folder paradigm for a long time. Standard email clients, such as Microsoft Outlook or Mozilla Thunderbird, store and display email in a folder hierarchy. Users have to set up these folders and move emails into them manually or via user-defined filter rules. While there has been much work on automatic foldering [4, 7, 14, 20], these methods are not widely used. One reason for this is the distrust of users in the under-

IUI'07, January 28–31, 2007, Honolulu, Hawaii, USA..

lying classification algorithms: They fear that a misplaced message may never receive the attention it deserves. In contrast, we never move messages out of the inbox, but provide a view on the data.

A large proportion of users live with a flat inbox structure, with just a few folders and filter rules for newsletters [16]. Our aim is to complement this flat structure with a view that provides a fully automatic grouping of emails based on topics, with no change to the inbox data and no additional cost to the user.

How do we group emails into topics? Our techniques are inspired by "topic detection and tracking" (TDT), a series of NIST competitions [18] aimed at organizing news articles into coherent topics. As in TDT for news articles, the core of our clustering algorithm is a simple text similarity measure. Although email text is often of lower quality than newspaper content, it is richer in non-textual information: We can make use of sender-receiver relationships and behavioral measures such as the percentage of replied emails, contact rankings based on email volume, past behavior, and reply timing. The focus of our work was designing a clustering algorithm which organizes email well, according to standardized NIST measures. We investigated different clustering methods and features, and present our findings in the section on clustering.

Another problem we address is that of topic labeling: After we identified topic groups, we label them so the user understands their contents at a glance. We briefly present a heuristic which uses the subject line for single-thread topics and common subject or content words for multi-thread topics or cases where the quality of the subject line is insufficient.

The rest of this paper is split into two parts: After reviewing related work, a first section explains our email client plugin and its functionality, while the rest discusses our clustering and labeling algorithms.

RELATED WORK

The problem of email overload [23] is now widely acknowledged and has found a lot of attention even in the popular media. A variety of solutions have been proposed, including task-based [22], activity-based [10], priority-based, [12], and sender-based [3] organization schemes. For example, Dredze et al. [10] provide successful algorithms to recognize emails that belong to particular activities, such as organizing a conference, reviewing papers, or purchasing equipment. Our approach is more general as it covers all kinds of discussions, but comes at the price of slightly decreased accuracy.

Most similar to our work are the "personal topics" proposed by Surendran et al. [21]. However, the mechanisms presented there provide a retrospective view of past emails, whereas our processing is an on-line scheme that is updated as new emails come in. We had to apply simpler but less computationally intensive schemes of topic clustering and labeling.

Automatic foldering [4, 7, 14, 20] also offers help in organizing email, but the user needs to manually create folders and seed them with example data – a laborious task if many fine-grained topics need to be differentiated. Many users distrust these schemes because they might move emails out of sight, never to be seen again [3].

While other models of text-based topic identification have been proposed [6], the work presented here uses the same techniques as topic detection and tracking for news articles [18]. Allan et al. investigated making this technology more accessible for normal users [2], but we are not aware of any work that applied TDT to email.

There exists a considerable body of work on redesigning email user interfaces, and grouping emails by various attributes. Two examples are TaskMaster [5], which helps users group emails, attachments, links, and other information by outstanding task, and Bifrost [3], based on the simple but ingenious idea of grouping emails based on the importance that the user manually assigned to contacts. In contrast, we automatically display topic groups without requiring any manual input. The user can still make manual corrections, if desired.

APPLICATION OVERVIEW

This section presents the functionality of BuzzTrack, its implementation, and usage.

BuzzTrack is implemented as an extension to Mozilla Thunderbird 1.5, but topic clustering and labeling is performed in Python. We will make this software available for download at `www.buzztrack.net`. The implementation contains about 3,000 lines of JavaScript / XUL code and 14,000 lines of Python code. Both components are platform-independent.

Figure 1 shows a typical screenshot of using BuzzTrack: A topic list was added as a sidebar on the left, and the email list on the right side groups emails by topic. When a new email arrives, it is processed by the clustering component, which returns either the decision to create a new topic for the email or adds it to one of the existing topics. Depending on application settings, each email can also be added to multiple closest topics.

Each item in the sidebar shows additional information about a topic: The topic label, a count and the names of all people involved in the topic, and the number of unread and total messages. In an expanded view, the full names of all topic participants are shown. Figure 2 provides an example. When the user clicks on a sidebar entry, the email list scrolls to the topic.

Topics in the sidebar are generally sorted by last incoming email. The user can "star" important topics to pull them to the top of the list: This solves a common user problem that important emails are quickly forgotten once they drop out of the first few screens of the inbox [23]. The email list ignores starring to mirror the traditional organization scheme of many users.

Unlike with conventional folder-based systems, users do not need to manually create or edit topic contents. However, they can still manually fix mistakes made by the clustering algorithm, either by drag-and-dropping items in the familiar way or through context menus. Similarly, users can manually rename topic clusters.

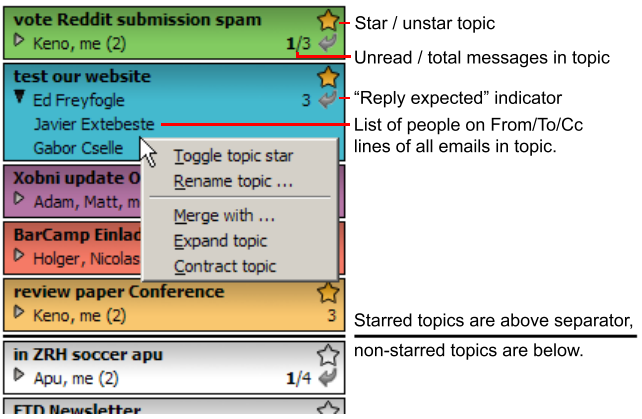

Figure 2: Components and supported operations in the topic sidebar.

We are also testing two experimental features: A useful "reply expected" indicator marks topics in which the newest email was addressed to the user and has not yet been replied to. The other feature is "expand topic" / "contract topic" – these two options pull less or more email into a certain topic if the user thinks that the topic is too large or that important messages have been left out. These operations retroactively modify the clustering threshold for a given topic.

At present, we concern ourselves solely with incoming email in the inbox. We assume that all spam has already been filtered out, either by the spam filter integrated in the email client or a filtering solution such as Spamato [1].

Note that the user can exit the BuzzTrack view and return to the traditional three-pane setup by clicking a button in the toolbar.

CLUSTERING

This section explains our clustering algorithm for grouping email into topics. It is similar to methods used for clustering news messages. Email is much richer in information content than the text of news articles, and we are able to use a large number of features for matching together emails, which we will discuss in detail.

The basic algorithm is single-link clustering with a distance metric which consists of a $tf \cdot idf$ text similarity measure and several non-textual attributes. We use an on-line clustering algorithm, as we are interested in immediately handling incoming emails.

The output of our clustering algorithm is a decision score that describes whether an email should be matched to a topic or not. When this score is below a clustering threshold for all existing clusters, the email is mapped to a new topic. Else, it is mapped to any number of closest topics. We have experimented with different methods for generating decision scores from feature values, including brute-force guessing, linear regression models, and linear support vector machines.

This section describes the preprocessing steps for emails and gives details of the features used in generating decision scores.

Preprocessing

In a first step, we parse each email's headers, email body, and attachments. We remember header information but only keep the file names of attachments. For the text similarity metric at the core of our algorithm, we need to clean and tokenize the email body and subject into terms of one word each and perform the following operations:

- We convert foreign characters into canonical form.
- We remove words with special characteristics. This includes HTML tags, style sheets, punctuation, and numbers. These will often contain information not relevant to topic matching.
- Identify parts-of-speech. For topic labeling, which occurs later on, we apply a part-of-speech tagger from the NLTK Lite toolkit for Python [15].
- We run the full text of subject and body through TextCat [8], a language guesser. If the language in which the email was written is known, we apply the Porter stemming algorithm to stem all words into their basic forms. We currently handle English and German. These two languages make up 99.6% of our corpus. We do not use the information from the language guesser for any other purpose.

Clustering Features

We now review the features we have constructed from the email data. We denote the N emails in the inbox with $m_1, ..., m_N$ and the M existing clusters, which represent one topic each and contain one or more emails, with $C_1, ..., C_M$. There exists a set of all n stemmed terms that appear in all emails. We refer to these terms as $t_1, ..., t_n$.

The first feature is a *text similarity* metric. We regard emails as weighted word occurrence vectors. For a preprocessed email m_j, we refer to the term frequency of term t_i as $tf_{i,j}$, and to the document frequency of t_i as df_i. We now define the term weight $w_{i,j}$ as follows:

$$w_{i,j} = \begin{cases} (1 + \log(tf_{i,j})) \log \frac{N}{df_i} & \text{if } tf_{i,j} \geq 1 \\ 0 & \text{if } tf_{i,j} = 0 \end{cases}$$

To determine text similarity, we use a standard cosine measure. Given two emails m_i and m_j, we define the text similarity measure as follows:

$$sim_{text}(m_i, m_j) = \frac{\sum_{k=1}^{n} w_{k,i} \cdot w_{k,j}}{\sqrt{\sum_{k=1}^{n} (w_{k,i})^2} \cdot \sqrt{\sum_{k=1}^{n} (w_{k,j})^2}}$$

A second measure is also text-based and measures *subject similarity*: It calculates the overlap between the set of words S_i, S_j in the subject lines of two emails:

$$sim_{subject}(m_i, m_j) = \frac{2|S_i \cap S_j|}{|S_i| + |S_j|}$$

Next, we use two *people similarity* metrics that compare the set of people participating in a topic with the set of people to which the email is addressed. For an email m_i, we derive a set $ppl(m_i)$ with all email addresses in the From, To, and Cc headers. Similarly, for each topic cluster C_k, there is a set of senders $ppl(C_k)$ which contains all email addresses from all

emails in the cluster. We define two people-based similarity measures as follows:

$$sim_{people,subset}(m_i, C_k) = \frac{|ppl(m_i) \cap ppl(C_k)|}{|ppl(m_i)|}$$

$$sim_{people,overlap}(m_i, C_k) = \frac{2|ppl(m_i) \cap ppl(C_k)|}{|ppl(m_i)| + |ppl(C_k)|}$$

These are equivalent to the $SimSubset$ and $SimOverlap$ metrics introduced in [10]. We also remove the user from the ppl sets, as he or she is by definition present on the receiver list of every message.[1] In addition, we introduce two variants of these indicators that operate on the domain name parts of sender addresses only, $sim_{domains,subset}$ and $sim_{domains,overlap}$. They help in recognizing emails coming from different people in the same organization or company.

The *thread similarity* measure is based on threading information contained in the "References" and "In-Reply-To" email headers. Almost all contacts in the corpus used modern email clients that employed these headers. The sim_{thread} measure gives the percentage of emails in the cluster which are in the same thread as the new email:

$$sim_{thread}(m_i, C_k) = |T|/|C_k|$$

The set T contains all $m_j \in C_k$ which are in the same thread as m_i.

We also add a number of simpler features, partly taken from existing literature [10, 17], and enriched with additions:

- *Sender rank*: A ranking of contacts by the number of emails received from them. We derive two additional features by multiplying the sender rank with $sim_{subject}$ and sim_{text}.
- *Sender percentage*: Fraction of total emails in inbox which came from the same sender in the past.
- *Sender answers*: Percentage of emails from the same sender which have been answered by the user in the past.
- *Time*: A linear time decay measure indicating how much time has passed since the last email in the topic.
- *People count*: Total number of people in the To/Cc headers of an email.
- *Reference count*: Number of references to previous emails in the header fields.
- *Known people*: Number and percentage of email addresses in the To/Cc headers which have been seen before.
- *Known references*: Number and percentage of emails in the references field whose ID matches an email we have sent or received.
- *Cluster size*: Cluster size of the topic being compared.
- *Has attachment*: 1 if the email has an attachment, 0 else.

Generating Decision Scores

We use two methods to construct decision scores from features. Both employ a linear combination of the feature values, which we normalize into a range of $[0, 1]$. In the "manual" method, we take four features with high information gain, guess appropriate ranges for their weights, and run a brute-force evaluator over the development set. For the second method, we determined feature values for twelve high-rank features in the development corpus, and trained a linear support vector machine on the development set by sequential minimal optimization [19] ("SMO"), as implemented in the Weka toolbox [24].

[1]The drawback of this choice is that we need a list of all email addresses of the user – this is not immediately available from the email client, as there may exist many aliases or forwarding addresses which the email client does not know about.

Time Window

During single-link clustering, we only consider topic clusters which have been active in the last 60 days. Statistics generated from the development corpus and existing research [13] show that replies to emails typically arrive just days after the previous email or never. However, we still need to be able to catch topics with long inter-arrival times: For example, reminder emails or newsletters sent only once per month. As we are not interested in organizing the user's entire email archive, topics older than than 60 days are dropped. A great benefit of this design choice is that processing time per newly arriving email is significantly reduced.

LABELING CLUSTERS

We use a simple heuristic for labeling clusters. In short, we use significant common words from email subject lines, if available, and resort to content words if the subject words are not descriptive. While more elaborate schemes have been presented [21], we cannot afford to spend much processing power on deriving cluster labels on-line. A topic's label is recalculated every time a new email is added to it.

For this task, we use information from several sources: In the preprocessing step, we use a tagger to identify nouns. During clustering, we derive a $tf \cdot idf$ value for each stemmed word term in each email. For each of these terms, we keep track of its most popular non-stemmed version and the most popular capitalized version.

Capitalization is an important factor: Users are well aware of the favored capitalization of certain terms, which are often very descriptive names or identifiers – "TDT," "Sarah," and the like. These words also tend to have high $tf \cdot idf$ values, and likely to appear in the topic label. In previous work [2, 21], only the lower-case version was shown.

Our algorithm distinguishes three cases:

1. If the topic consists of emails from just one thread with at least two words in the common subject, and these words have sufficiently high $tf \cdot idf$ weights, set the label to the common subject (with "Re," "Fwd," and similar prefixes removed). The first constraint ensures that we cover all emails in a topic, while the last two constraints seek to guarantee sufficient descriptiveness for the topic.

2. If there is more than one thread, try to find a subset of at least two subject words which occur in every email's subject, and have sufficiently high $tf \cdot idf$ weights. If such words are found, set them as the topic label. This method is very useful for finding newsletter labels, as they have subjects of the form "FTD Newsletter - 28 Aug 2006," "FTD Newsletter - 4 Sep 2006," and so on. In this example, the

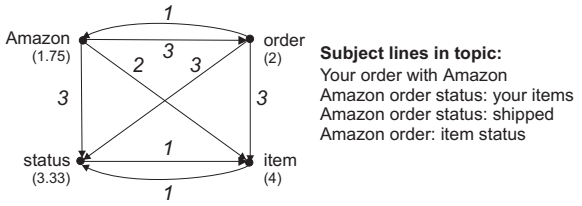

Figure 3: Example graph of occurs-after relationships between high-$tf \cdot idf$ terms. The numbers in parentheses give the average first position. Here, "your," "with," and "shipped" are not nouns and not high-$tf \cdot idf$ words. Due to stemming, "item" and "items" map to the same term. The resulting label would be "Amazon order status items".

topic label will be "FTD Newsletter," a good description of the topic contents.

3. If the first two methods fail, we take the 3 highest-$tf \cdot idf$ noun words for the cluster.

After having selected words for the label, how do we order them? We go back to the tokenized mail structure and look up the first occurrences of each word in each document's subject plus body. Depending on their relative order, we construct a directed, weighted graph of "occurs-after" relations. Each edge in the graph has an associated count with the number of emails in which the source word occurs before the target word.

As the first word, we choose the one with the highest value for total outgoing weight minus total incoming weight. We then traverse the graph by choosing the highest-weight link each. If two links share the same highest weight, we choose the word with the lowest average first position of the word. Figure 3 provides an example graph and result.

Note that we need to keep an extra $tf \cdot idf$ value for labeling: We regard the emails of the entire cluster as one document as compared to the entire document collection. This is useful if many emails inside the cluster contain the same word, but the word never occurs outside the cluster. This word is very descriptive for the cluster, but with a per-document $tf \cdot idf$ measure, its value may be too low.

We found that this method produces labels of sufficient quality. We did not specifically cover label quality in our evaluations. In case the generated label is bad, the user can still manually rename the topic as a last resort.

EVALUATION METHODOLOGY

For evaluating our clustering scheme, we follow the guidelines provided by NIST for the evaluation of topic detection and tracking on news articles [18]. We give a short overview here.

The TDT evaluation scheme defines two tasks: New Topic Detection and Topic Tracking. In effect, the clustering evaluation is recast into two detection tasks.

The *New Topic Detection Task* (NTD) is defined to be the task of detecting, in a chronologically ordered stream of emails, the first email that discusses a new topic. For each email in the stream, the output can be "yes" (the email is a new topic) or "no" (the email is not a new topic).

The *Topic Tracking Task* (TT) is defined to be the task of associating incoming emails with topics that are known to the system. A topic is "known" by its association with emails that discuss it. Each target topic is defined by the first email that discusses it. The tracking task is then to classify correctly all subsequent emails as to whether or not they discuss the target topic. For each topic / email pair in the stream, the output can be "yes" (the email belongs to the topic) or "no" (the email does not belong to the topic).

For each decision in each task, the system must output the decision score which describes the level of confidence with which the classification was made. These scores will later be used to find a threshold which presents the optimal trade-off between misses and false alarms.

Detection quality is characterized in terms of the probability of miss and false alarm errors, p_{miss} and p_{FA}. These error probabilities are combined into a single detection cost:

$$C_{det} = C_{miss} \cdot p_{miss} \cdot P_{target} + C_{FA} \cdot p_{FA} \cdot P_{nontarget}$$

where:

- C_{miss} and C_{FA} are the costs of a miss and a false alarm.
- p_{miss} and p_{FA} are the conditional probabilities of a miss and a false alarm.
  Misses are false negatives: p_{miss} is equal to the number of "yes" instances classified as "no," divided by the total number of "yes" instances.
  False alarms are false positives: p_{FA} is equal to the number of "no" instances classified as "yes," divided by the total number of "no" instances.
- P_{target} and $P_{nontarget}$ are the a priori target probabilities ($P_{nontarget} = 1 - P_{target}$).

Relative costs for misses and false alarms depend on the tasks. The a priori probabilities depend on the detection probabilities for the corpus. From our development corpus, we determined a value for P_{target} of 0.3 for the NTD task, and a P_{target} of 0.02 for the TT task. As in the NIST evaluation, we believe that misses are far more dramatic than false alarms: We choose $C_{miss} = 1.0$ and $C_{FA} = 0.1$.

Lastly, C_{det} is normalized such that a perfect system scores 0 and a trivial system which always emits "yes" or "no," depending on whether "yes" or "no" instances are more common, scores 1.

$$C_{det,norm} = C_{det} / \min(C_{miss} \cdot P_{target}, C_{FA} \cdot P_{nontarget})$$

The normalized minimum detection cost characterizes the overall performance of a clustering scheme.

CORPUS

To carry out the evaluation, we needed a corpus to compare clustering results with a defined target. We manually created a corpus of topics from the email of one of the authors.

Why not use or adapt an existing corpus? One consideration was to use the Enron corpus [14] and manually group emails

into topic groups, possibly using as a guide the existing organization structure of users with a large number of folders. However, we rejected this idea for three reasons: First, because we would essentially organize a stranger's email, we would have no idea about what constitutes a topic for that person. Second, one of the features we use in clustering is the threading information present in email headers ("References" and "In-Reply-To"). These headers are present in only few messages of the Enron corpus, although there have been efforts to reconstruct thread structures based on message contents and undocumented Microsoft Exchange headers [25]. Lastly, the author whose email was used kept all communication from the past years – unlike with the Enron corpus, we could be sure no emails would be missing in the evaluation.

The corpus was manually divided into topics by its owner. It covers one academic semester at ETH Zurich, with the first 4 months used as a development set. The test set with the last 2 months of email data was used to get final results only. Figure 4 shows the resulting split. We use a fixed trained model derived from the development set to generate decision scores, since users are unlikely to have large amounts of prelabeled data to train the classifier on.

	Development Set	*Test Set*
Number of emails:	1586	817
Time range:	Oct 1, 2005 to Jan 31, 2006	Feb 1, 2006 to Mar 31, 2006
Languages:	730 English 851 German	420 English 397 German
Number of topics:	537	269
avg(emails/topic):	2.95	3.03
std(emails/topic):	5.64	4.91
max(emails/topic):	69	47

Figure 4: Corpus statistics.

What criteria were used to assign emails to topics? The name "topic" means different things to different people, and the definitions are very subjective. Here are some typical examples for topics in the corpus:

- All emails belonging to the same newsgroup.
- All emails from a small-volume sender with whom only one subject was discussed.
- For larger-volume senders, emails were subdivided further into coherent conversations. For example, one topic contains emails from a colleague inquiring about the owner's experience with a particular brand of digital camera. In a second thread, a number of emails are exchanged discussing image quality. Half a week later, in yet another thread, the sender has bought the camera in question and is proudly sending a test photo.
- Conversations with multiple people, about the same activity. For example, one topic covers all emails of the process of recommending candidates for an internship: Emails are exchanged with the candidate and two company recruiters.

Two difficulties are *thread drift* and *topic drift*. Thread drift means that a topic contains several threads, as in the digital camera example. The average topic in the development set consists of 2.44 threads. Topic drift means that the same thread contains information about different topics: One particularly annoying example is a colleague who uses the "Reply To" button instead of "New Mail": When emailing the author, he simply searched for the last email from the corpus owner, hit "Reply To," and only sometimes deleted the quoted text. This type of behavior is hard to detect, as continued threads are usually very strong signals of an email belonging to an existing topic. Gladly, the average number of topics per thread in the development corpus is just 1.04 – this problem applies to only a small number of emails.

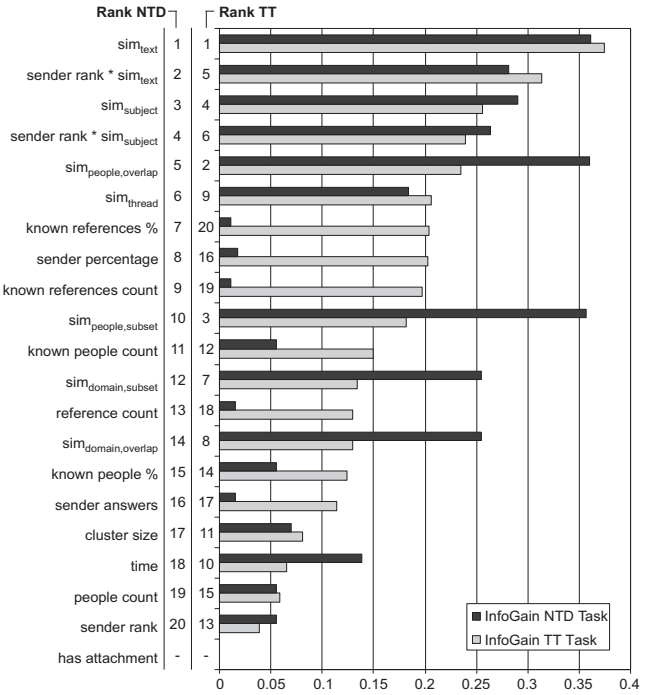

Figure 5: Information gain ranking of features from development corpus data: New Topic Detection and Topic Tracking tasks.

EVALUATION RESULTS

Which features help in making correct decisions? For the development corpus, we calculated the information gain measure for each feature, measured as $H(C|F)$, with class C ("yes"/"no") and feature F. Features that are more useful for decisions will have higher information gain. Figure 5 shows the information gain of each feature for performing both tasks on the development corpus. While textual, thread-based and people-based attributes are useful in both tasks, other attributes seem less valuable. One striking result is that our linear time decay measure does not help in clustering emails, due to the high number of topics being discussed in parallel. One the other hand, it is no surprise that the attachment indicator is useless for clustering purposes.

We use Detection Error Tradeoff (DET) curves [18] to visualize the trade-off between the missed detection rate p_{miss} and the false alarm rate p_{FA}. The curves are constructed by sweeping the clustering threshold through the system space of decision scores. At each point in the score space, p_{miss}

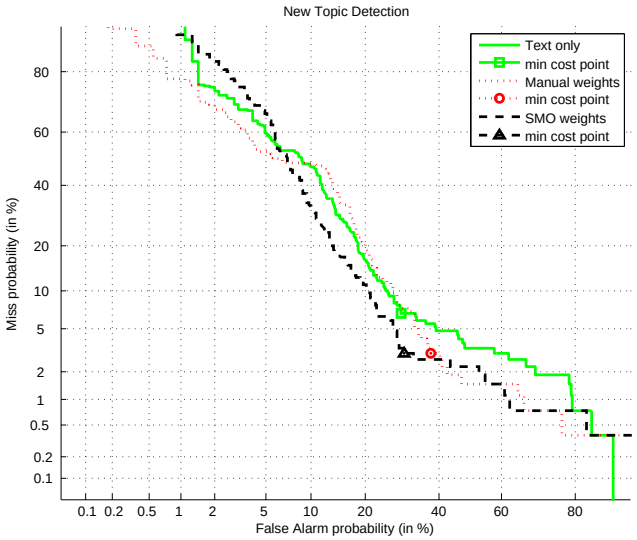

Figure 6: NTD task: DET curve with minimum cost points.

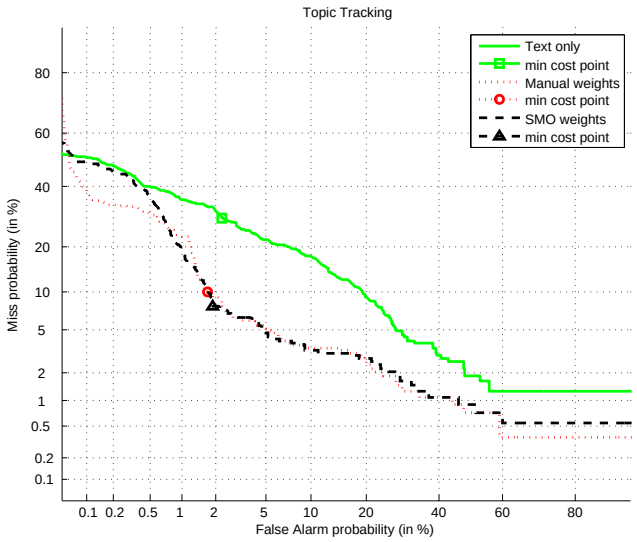

Figure 7: TT Task: DET curve with minimum cost points.

and p_{FA} are estimated and plotted as a connected line. We mark the point on the curve for which the detection cost C_{det} is minimized. Figures 6, 7, 8, and 9 show results for the test corpus. For reference, we included results for simpler decision scores derived from text similarity only – the main feature used in traditional TDT.

These results are comparable in performance to that of current work on TDT in news articles [11]: In comparison, our TT quality is a bit worse, and NTD performance a bit better. This demonstrates that while the text quality of emails is low in comparison to news articles, we were able to leverage the extra information present in email. The difficulty of this model is the highly subjective definition of topics and dependence on tastes of users. Difficulties arise for the clustering algorithm when new emails arrive for a topic without sharing

Task	text only	manual	SMO
NTD	0.575	0.503	0.425
TT	0.396	0.184	0.171

Figure 8: Minimum detection cost $C_{det,norm}$ for NTD and TT tasks.

people or vocabulary with the previous emails. Typical examples are topics which reflect a user's role, e.g. as a system administrator: Printer problems and account requests are in the same topic, but deriving a connection between the two is hard. For the user, the consequence is that topics may often need to be manually merged. However, high false alarm rates also present a problem in on-line new event detection for news content.

In addition to the results presented here, we have also experimented with supervised clustering, where tracking errors are instantly corrected during evaluation. This resulted in improved TT performance, as discussed in [9].

With our current Python 2.4 implementation, running on a PC with a 1.8 GHz Pentium-M processor and 1 GB RAM, 100 active clusters and 350 emails in these topics, clustering an incoming email takes between 10 ms and 500 ms. In the same state, labeling each topic takes approximately 100 ms. Such short delays are acceptable to email client users.

FUTURE WORK

In the future, we want to further improve clustering performance, based on the broad range of existing work for news articles. Also, we plan to investigate methods clustering together related topics, which may be constructed from existing decision scores. The topic corpus we built from email data is actually hierarchical, and the TDT evaluation guidelines already suggest ways of evaluating hierarchical clustering methods. Finally, we would also like to experiment with BuzzTrack's user interface and introduce useful additions. For example, instead of the grouping scheme proposed here, the inbox view could provide context by showing a visual map of emails related to the one currently selected.

CONCLUSION

Users of email in today's corporate environments deal with large amounts of incoming email. To stay on top of it, they rely on methods such as manually sorting emails into folders or searching through an unstructured inbox. We designed, implemented, and evaluated a system which, at no cost to the user, generates a sensible structure for the contents of the inbox. Namely, we structure the inbox by topic – emails revolving around the same question, idea, or theme. This is a

Task		p_{miss}	p_{FA}	*Success*	*Prec*	*Rec*
NTD	text	0.07	0.29	0.78	0.61	0.93
	manual	0.03	0.38	0.74	0.56	0.97
	SMO	0.03	0.30	0.79	0.61	0.97
TT	text	0.03	0.02	0.98	0.13	0.71
	manual	0.10	0.02	0.98	0.20	0.90
	SMO	0.08	0.02	0.98	0.19	0.92

Figure 9: Quality measures in minimum cost point for NTD and TT tasks.

broader perspective than existing approaches which focus on task- or business process-related email sorting. Unlike automatic foldering, the user does not need to manually create folders or create a root set for learning folder contents: New topics are recognized automatically. Our clustering methods and their evaluation were inspired by the work in the TDT (topic detection and tracking) community, and we found that these techniques also perform well on email data. We developed the BuzzTrack plugin for a popular email client to access this functionality through a user-friendly interface. We believe this will be a valuable tool for users in the real world.

ACKNOWLEDGEMENTS

The authors would like to thank Markus Egli, Erol Koç, Fabian Siegel, Michael Kuhn, Roland Flury, Aaron Harnly, Matt Brezina, Adam Smith, and Abhay Puri for helpful discussions, and the anonymous reviewers for their suggestions.

REFERENCES

1. Keno Albrecht, Nicolas Burri, and Roger Wattenhofer. Spamato - an extendable spam filter system. In *Proc. Conference on Email and Anti-Spam (CEAS) '05*, 2005.

2. James Allan, Stephen Harding, David Fisher, Alvaro Bolivar, Sergio Guzman-Lara, and Peter Amstutz. Taking topic detection from evaluation to practice. In *Proc. Hawaii International Conference on System Sciences (HICSS) '05*, page 101.1, 2005.

3. Olle Bälter and Candace Sidner. Bifrost inbox organizer: giving users control over the inbox. In *Proc. Nordic conference on Human-computer interaction (NordiCHI) '02*, pages 111–118, 2002.

4. Ron Bekkerman, Andrew McCallum, and Gary Huang. Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora. Technical Report IR-418, CIIR, UMass Amherst, 2005.

5. Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *Proc. Conference on Human Factors in Computing Systems (CHI) '03*, pages 345–352, 2003.

6. David Blei and John Lafferty. Correlated topic models. In *Advances in Neural Information Processing Systems*, pages 147–154. MIT Press, Cambridge, MA, 2006.

7. Gary Boone. Concept features in Re:Agent, an intelligent email agent. In *Proc. Conference on Autonomous Agents (AGENTS) '98*, pages 141–148, 1998.

8. William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proc. Symposium on Document Analysis and Information Retrieval (SDAIR) '94*, pages 161–175, 1994.

9. Gabor Cselle. Organizing email. Master's thesis, ETH Zurich, 2006.

10. Mark Dredze, Tessa Lau, and Nicholas Kushmerick. Automatically classifying emails into activities. In *Proc. Conference on Intelligent User Interfaces (IUI) '06*, pages 70–77, 2006.

11. Jonathan Fiscus and Barbara Wheatley. Overview of the TDT 2004 evaluation and results. National Institute of Standards and Technology, 2004.

12. Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive alerting. In *Proc. Conference on Uncertainty and Artificial Intelligence (UAI) '99*, pages 305–313, 1999.

13. Yoram M. Kalman and Sheizaf Rafaeli. Email chronemics: Unobtrusive profiling of response times. In *Proc. Hawaii International Conference on System Sciences (HICSS) '05*, page 108.2, 2005.

14. Bryan Klimt and Yiming Yang. Introducing the Enron corpus. In *Proc. Conference on Email and Anti-Spam (CEAS) '04*, 2004.

15. Natural language toolkit. http://nltk.sourceforge.net/.

16. Carman Neustaedter, A. J. Bernheim Brush, and Marc A. Smith. Beyond "from" and "received": Exploring the dynamics of email triage. In *Proc. CHI '05*, pages 1977–1980, 2005.

17. Carman Neustaedter, A. J. Bernheim Brush, Marc A. Smith, and Danyel Fisher. The social network and relationship finder: Social sorting for email triage. In *Proc. Conference on Email and Anti-Spam (CEAS) '05*, 2005.

18. NIST. The 2004 topic detection and tracking (TDT-2004) task definition and evaluation plan. Technical report, National Institute of Standards and Technology, 2004.

19. John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

20. Richard Segal and Jeffrey O. Kephart. Mailcat: An intelligent assistant for organizing e-mail. In *Proc. Conference on Autonomous Agents (AGENTS) '99*, pages 276–282, 1999.

21. Arun C. Surendran, John C. Platt, and Erin Renshaw. Automatic discovery of personal topics to organize email. In *Proc. Conference on Email and Anti-Spam (CEAS) '05*, 2005.

22. Steve Whittaker, Victoria Bellotti, and Jacek Gwizdka. Email in personal information management. *Communications of the ACM*, 49(1):68–73, 2006.

23. Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *Proc. CHI '96*, pages 276–283, 1996.

24. Ian H. Witten and Frank Eibe. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann, 2005.

25. Jen-Yuan Yeh and Aaron Harnly. Email thread reassembly using similarity matching. In *Proc. Conference on Email and Anti-Spam (CEAS) '06*, 2006.