Organizing Email

Master's Thesis

Gabor Cselle
<mail@gaborcselle.com>

Prof. Dr. Roger Wattenhofer
Dr. Keno Albrecht

Distributed Computing Group
Computer Engineering and Networks Laboratory (TIK)
Departments of Computer Science and Electrical Engineering
ETH Zurich, Switzerland

October 3, 2006

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing Group

Abstract

Email clients were not designed to handle the volume and variety of messages users are dealing with today. While this problem, termed “email overload,” is widely acknowledged, there exist only few products which attempt to solve it. This thesis discusses an email client extension aimed at sensibly structuring the user’s incoming email.

We develop our ideas after a thorough review of the problem and existing approaches of organizing email in both research and practice. In particular, we explore automatic foldering, in which machine learning algorithms are used to automatically move emails to folders. However, this method has several drawbacks which we discuss in detail. Instead, we have devised a scheme for topic detection and tracking in email: Related emails are grouped and displayed together. The underlying inbox data is not changed. We evaluated our scheme using existing metrics for news articles, and found good results for clustering quality. We created an extension called “BuzzTrack” for Mozilla Thunderbird, a popular email client, which makes this technology accessible to the end user.

Acknowledgments

Ideas never form in a vacuum, but are the product of interaction. Many people have contributed to this thesis, knowingly or unknowingly, and I'd like to acknowledge those who have provided encouragement, help, and inspiration.

First of all, I'd like to thank my reviewers, Keno Albrecht and Roger Wattenhofer, for their generous support.

I'm very grateful to my friends Erol Koç, Markus Egli, and Fabian Siegel, who read initial drafts of this thesis and encouraged me to stay productive. At ETH Zurich, Bálint Miklós, Fabian Bannwart, Yvonne-Anne Oswald, Moritz Kuhn, Samuel Riedmann, Daniel Dalquen, Sandra Brockmann, Marcus Deluigi, Remo Marti, Thomas Zweifel, Remo Meier, Thomas Gloor, Otmar Caduff, and Peter Mathoy have inspired interesting and productive discussions.

Throughout the last six months, many friends volunteered information about their email behavior and what they would find useful in an email client, among them Apurva Shah, Brian Rakowski, Christian Frank, Christophe Dessimoz, Clara Lee, Darick Tong, Douwe Osinga, Ed Freyfogle, Heather Stevens, Heike Schmitz, Javier Extebeste, Lisa Tannenbaum, Marc Tobias Metten, Michael Kuhn, Nicolas Burri, Nicolai Fischli, Nina Tannenbaum, Paul Buchheit, Paul Graham, Roland Flury, Yves Weber, Vijay Pandurangan, and Vincent Koh.

Some have contributed pieces which I was able to directly incorporate in my work: Abhay Puri suggested the “Reply Expected” indicator in BuzzTrack. Jonas Landolt offered ideas for finding optimum feature weights for the clustering component. Aaron Harnly suggested features that improved clustering quality. William Cohen supplied an implementation of the Ripper classifier. Adam M. Smith and Matt Brezina shared ideas for analyzing email behavior. Peter Dienes and Sascha Brawer provided valuable input on topics of natural language processing. The Globis Group at ETH Zurich supplied the document template.

Last but not least, I'd like to thank my parents, Andrea Cselle and Tibor Cselle, and my brother Andreas, for their loving care.

Contents

1 Introduction

Email is one of the most important communication media of our time. Email enables global communication at no incremental cost. It has contributed to the emergence of organizations that are distributed world-wide, allowing people to communicate across space and time.

While email was designed as a communications application, it is now being used for additional functions which it was not designed for, such as task management, personal archiving, and file transfer. Also, because of email's popularity, users must now deal with rising volumes of email in their inbox. Large amounts of information need to be processed and organized. The term *email overload* was coined by Whittaker and Sidner [1996] to refer to these two problems.

A variety of approaches have been proposed to defuse email overload, which is why we begin this thesis with a review of methods from existing literature. A wide range of approaches exists: For example, email overload can be tackled at the level of the individual, by installing organization software, or at global level where email users worldwide adopt new standards of communication. There is also a time component: Email overload applies to both managing current email and handling communications from the past.

Our scope in this thesis is managing and exploring a single user's email, not repositories of entire organizations. Thus, we focus on the two leftmost squares in Table 1.1.

One popular approach in organization helpers is *automatic foldering* – automatically moving user's email into folders, for which we have implemented and evaluated several methods. However, this scheme has three drawbacks: The first is the poor accuracy of classifiers on real-world data. Second, users distrust automatic schemes in which emails disappear from the inbox, never to be seen again. Third, folders need to be seeded with example data so that the classifiers have instances to learn from.

The increased availability of fast full-text search for email is eliminating the need for elaborately structuring the email archive containing old emails into folders. However, sensibly

	Individual	Organizational	Social
Current	Managing an individual user's current inbox	Managing current email within an organization	Managing current conversations in a public space
Archived	Exploring an archive of an individual's messages	Exploring an archive of an organization's messages	Exploring an archive of a public space

Table 1.1: Approaches taken for organizing and exploring email. Highlighted: the focus of this thesis. Adapted from Perer et al. [2006].

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 context.

Sensibly structuring emails in the inbox is the specific problem we address in this thesis.

We have devised a scheme for *topic detection and tracking in email*: We display emails by topic, instead of the traditional list-based approach. We discuss a clustering algorithm that creates the topic-based grouping, and a heuristic for labeling the resulting clusters to summarize their contents. In this scheme, the user's inbox stays untouched: We simply provide a view on the data, which we integrate into the email client as a plugin.

The rest of this thesis is structured as follows:

- We begin by explaining the problem of *email overload* in Chapter 2.
- *Existing work* on email clients, visualization tools, and data mining to address the problem is discussed in Chapter 3.
- We provide an overview of work in *automatic foldering*, along with some own results, in Chapter 4.
- We present our main contribution, *topic detection and tracking in email* as a solution to email overload in Chapter 5.
- Chapter 6 provides an overview of our software's implementation.

We assume that the reader is already familiar with email terminology and email client applications.

A note on spelling: We will use "email" instead of the commonly used "e-mail." The reason is not one of belief, but of practicality: Most existing literature also follows this convention.

2 Email Overload

In 1982, Peter Denning, then President of the Association for Computing Machinery (ACM), wrote [Denning, 1982]:

> "Electronic mail is rapidly providing faster methods of dissemination than are possible with traditional methods such as copying, mailing, and telephone. [...] But such tools may also increase the verbiage without increasing the number of ideas. [...] Who will save the receiver from drowning in the rising tide of information so generated?"

Since this first article from almost 25 years ago, the number of email messages sent has increased dramatically, from 5.1 billion messages per day in 2000 to an estimated 171 billion messages daily in 2005 [Radicati Group, 2006a]. Email users in today's corporate environments spend significant amounts of time dealing with their email and the problem has found a lot of attention even in the popular media.

Since Denning's article, several studies have been undertaken that looked at the requirements and frustrations of email users (the earliest being Mackay [1988]). This chapter provides a summary of these studies and the problems described therein.

We should point out that email overload is different from the spam or junk mail problem: It does not describe unsolicited, unwanted advertising email, but the need to deal with the large amounts of legitimate email that users receive today.

2.1 Overview

Email overload describes the problem that although email was originally conceived to be a *communications application*, it is now being used for additional functions that it was not designed for, such as *task management* and *personal archiving*.

This problem was originally described by Whittaker and Sidner [1996], based on the results of interviews that were conducted with users at Lotus Development Corporation. Other studies have since supported their findings [Gwizdka, 2004; Ducheneaut and Bellotti, 2001; Venolia et al., 2001; Whittaker et al., 2006].

Email is now used for the following functions:

1. **Task Management**: For many users, the inbox serves as a task management tool. Items in the inbox often serve as reminder for todo items, and users even send items to themselves as reminders.
2. **Personal Archiving or Filing**: Users store email to organize and categorize long-term information, so it can later be retrieved.
3. **Keeping Context**: Workers are usually engaged in several independent, but concurrent conversations, which they need to track at once. In addition, workers frequently need to switch context between these conversations.

We will first explore each of the these tasks, followed by a look at how users currently triage incoming email and decide what to do with it.

2.2 Task Management

Email is now 'co-opted'[1] by its users for many information management functions, such as todos and contact management [Ducheneaut and Bellotti, 2001; Whittaker et al., 2006]. For example, Gwizdka [2004] have found that a certain set of users frequently uses email as a personal organizer, which is the same set of people that often keeps email, constantly checks for incoming messages, and lets email interrupt other work tasks.

There are several strategies for misusing email clients as tools for task management, which consists of keeping track of task status, reminding of outstanding item, tracking status, and maintaining information related to each item. These strategies are summarized in Table 2.1 and contrasted with what task-based email clients, which we will review in Section 3.2.1, could offer.

A popular technique of marking items in the inbox as todo is to leave them unread: These messages may contain information still to read, descriptions of outstanding tasks, remind of outgoing correspondence owed to a contact, or may even have indeterminate status that must be reviewed. Some send themselves email to be reminded of outstanding tasks. However, this approach does not scale well: As items disappear from the first few screens, they are quickly forgotten by the user.

Another approach is to keep one folder for all todos, or an extra folder for each task. This strategy only works well if one develops a habit of returning to inspect these folders, as email clients do not support the notion of reminder alerts. The same drawback applies to

[1]co-opted: made to participate in tasks it was not designed for, against its will

	Messages in inbox, marked unread	*Extra folder(s) for outstanding tasks*	*Task-based email clients*
Pros	No extra effort required. Outstanding tasks easily visible when scanning past mail.	Provides overview of outstanding tasks.	Provides overview of outstanding tasks. Related tasks can be grouped. Reminding functionality (alerts).
Cons	Todo items mixed with other messages. No overview of / context for outstanding tasks. Old tasks quickly disappear from first screens.	Requires extra effort. Need to develop habit to check tasks folder(s).	Lacks support in most popular email clients. Often requires extra organizing and learning effort.

Table 2.1: Strategies for (mis-)using email clients as task managers.

the ‘tagging’ or ‘labeling,’ organization schemes which have found much attention in recent years with the rise of Gmail [Google, 2004] and social bookmarking sites [Delicious, 2003].

Task-based email clients are a popular research field: Emails and other items such as shared objects can be grouped together by the user and collapsed into a single list item. Two basic approaches exist [Whittaker et al., 2006]: With *Centralization*, all personal information management functions would be located in the email client. This is the approach of Microsoft Outlook. *Information extraction* is the opposite approach, aiming to extract information from email and make it accessible to dedicated task management applications using standard data formats. In this thesis, we will cover only research that takes the first approach in Chapter 3.

2.3 Personal Archiving

This section addresses the need of users to store emails that are not relevant to current tasks, but are kept anticipating use in the future. While Whittaker and Sidner [1996] originally dealt with keeping emails in folders only, there are now three popular styles storing and accessing email archives [Whittaker et al., 2006]: *Folders, search, and sort.* Table 2.2 summarizes the respective pros and cons of each approach, which we now review in detail.

2.3.1 Archiving in Folders

All of today’s popular email clients allow users to create folder hierarchies to store email in. This approach is familiar to many users from the file system abstraction present in operating systems. When Whittaker and Sidner [1996] first investigated folder use behavior, they found three common approaches:

No filers made no efforts to organize their email into folders, but used full-text search to find old messages.

	Folders	*Sort*	*Search*
Pros	Reduce inbox clutter. Familiar abstraction. Shows similar emails in context.	Supported by all mail clients. Shows similar emails in context.	No effort needed by user. Fast if supported by email client.
Cons	Takes effort. Creating and choosing folders is a cognitively difficult task. User may forget old folders.	Leads to inbox clutter. Requires remembering information about message content. Sorting choices limited by mail client.	Leads to inbox clutter. Requires remembering information about message content.

Table 2.2: Pros and cons of the three archiving strategies.

Frequent filers made daily passes through the inbox filing or deleting its contents.

Spring cleaners performed intermittent clean-ups (once every 1-3 months).

Manually moving email into folders brings some problems: Organizing email in folders takes extra effort and time on part of the users. In addition, manual classification is a cognitively difficult task: When creating folders, users are required to predict future usage needs. As they organize messages into existing folders, users are inconsistent and may forget the existence of their own long-term folders. In effect, a given folder may end up containing very different messages, or multiple folders may contain similar material.

How many folders need to be created? Bälter [2000] captures foldering behavior with a mathematical model, based on assumptions about the required times for UI operations, such as finding a folder and moving a message into it. He analyzes the trade-off between the number of folders and the time spent organizing, and calculates efficiencies for several user profiles. His result is that extensive and deep filing of email is not as efficient time-wise as a flat and simple file structure with only a few folders. Users should limit the number of folders to retain their usefulness. Folders with only a few messages are too small, do not reduce the complexity of the inbox, and do not gather a significant amount of related material. However, folders can also be too large: A folder containing thousands of unrelated messages does not reduce the complexity the user needs to deal with.

A common proposed solution for taking the burden of filing off the user's shoulder is *automatic foldering*. Machine learning techniques analyze message headers and content, and move messages into the correct folder. We will investigate this approach in Chapter 4.

2.3.2 Searching and Sorting

Ducheneaut and Bellotti [2001] report that: "Surprisingly, very few users report much use of the search feature of their email client, but, on the contrary, almost all [...] use the sort feature a lot." With sorting, the search criteria, such as date and name, can be more quickly specified: One only has to find a name with an attribute equal or similar to the same attribute of the email searched for, and click in the column headers.

While almost all clients support re-sorting lists this way, few have supported fast search based on full-text indexes until recently, which is possibly the reason for this user preference: If searching takes minutes, and re-sorting takes fractions of a second, users will prefer the less powerful sorting method.

Some clients also support the notion of virtual folders, which is at the intersection of search and foldering: The user creates a standing search query whose results are then displayed as a folder. This is a useful feature for tracking emails with certain keywords. However, it our experience, it is seldom used.

Even fast and powerful search, however, is only effective for accessing information already identified as relevant to a given task. The user must recall portions of the context in which to search for messages.

2.4 Keeping Context

Users today are often bombarded with a large number of messages from multiple ongoing conversations. Standard inbox functionality displays emails as a list, often sorted by arrival time. In this flat representation, context is lost.

Whittaker and Sidner [1996] identify requirements to support asynchronous communication in mail clients, which requires adding:

- *Threading* to support context regeneration and the management of conversational history.
- The ability to track the *status* of a conversation.

Thread-based tree views have been a standard feature of mail clients for years. In Chapter 5, we will also explore a technique to automatically cluster *semantically* related documents. This technique allows us to view emails in their conversational context.

2.5 Triaging Email

Email triage is the process of going through unhandled email and deciding what to do with it. Email can be instantly replied to, saved for later reading or action, archived or deleted. Two papers [Venolia et al., 2001; Neustaedter et al., 2005a] have explored aspects of the triaging behavior of users based on surveys inside Microsoft Corporation, and we will summarize some of their findings here.

What *strategies* do users follow in handling email? Neustaedter et al. [2005a] found that users check email and perform triage first thing each morning and check again several times a day, as time allows. About 50% of their participants used a single-pass approach – emails are triaged by sequentially scanning top-to-bottom –, while the other half performed multiple passes. Interestingly enough, users were found to deal with low-importance emails first, as they take only a short amount of time to handle and make important emails more visible.

Venolia et al. [2001] have explored the *factors in message importance*: Emails are important if they are a response to an email sent by the user, specially marked as important, or if they are from a set of people that the user finds particularly important (e.g. managers, project members).

A popular tool in triaging are *message filter rules* which exist in practically all email clients. Neustaedter et al. [2005a] explored the use of rules, and found that rules were more heavily employed as email volumes increased: Low-volume and medium-volume email users (less than 100 new emails / day) had a median of 5-9 rules, while high-volume users (more than 100 emails / day) had 10-14 rules. A majority of these rules were used to handle emails sent to mailing lists.

Overall, these results are hardly surprising to anyone who is confronted with large amounts of email every day. Clearly, the designs of email clients need to take these behaviors into account and should ultimately support triaging activities. We will explore such designs in the following chapters.

2.6 Conclusion

In this section, we summarized existing work about the challenges that users are facing, in particular the fact that email today is overloaded with many tasks it was not designed for. The next chapter will review existing research suggesting solutions to these problems, after which we will focus on a solution to the problems of *archiving* and *keeping context*: The need to filter information, the lack of context and priority, and the problem of poor integration, grouping, and visualization in today's email clients.

3 Existing Approaches

This chapter reviews existing approaches to organize email. It starts with the current state of the field as seen by the user in today's widely deployed products. It then reviews current research work.

Our aim is not to give a comprehensive review of the field of email organization, which would be a huge undertaking. Instead, we try to cover work relevant to the focus of this thesis and highlight ideas and products which we found particularly insightful.

Chapters 4 and 5 carry their own sections with related work.

3.1 Existing Products

In this section, we will give an overview of existing approaches, currently in user domain. Table 3.1 lists the products we will examine.

A recent study [Radicati Group, 2006b] gave the annual revenues of the corporate messaging software market for 2005 as $2.3 billion, with Microsoft having a 29 percent market share, while IBM had around 22 percent.

From a sample of the author's incoming email ranging from May 2001 to April 2006 and consisting of 8900 emails, we derive a different picture, shown in Figure 3.1. It counts the percentage of incoming email which was composed from each user agent. The client software used was derived from email headers (such as the `X-User-Agent` header, or the domain name in case of webmail applications). During this time span, the author received emails from 1125 unique contacts in 472 domains. A total of 65 different email clients were seen, with 374 email client / version combinations. While this should not be interpreted as a definitive measurement of email client market shares, it does demonstrate the diversity of email clients in the field.

	Version	*Platforms*	*Specialties*
Desktop Email Clients			
Microsoft Outlook	2007 Beta 2	Windows	Three-pane view Extensible (COM/MAPI) Variety of plugins available
Mozilla Thunderbird	1.5	Windows, Linux, MacOS	Three-pane view Extensible (XPCOM/JavaScript/XUL) Open source (Mozilla license)
Web-based Email Clients			
Gmail	–	Web-based	Thread-based view Labels instead of folders
Yahoo Mail	–	Web-based	Mimics traditional email clients
Desktop Search			
Google Desktop	4.0	Windows	Extensible
Windows Desktop Search	2.6.5	Windows	Extensible
Analytics Tools			
Xobni Analytics	Beta	Microsoft Outlook	Helps users analyze own email behavior

Table 3.1: Overview: Existing products.

3.1.1 Desktop Clients

A wide variety of desktop email clients exist – from command-line-based clients like Mutt and desktop applications like Apple Mail [Apple, 2006] to enterprise applications such as Lotus Notes [IBM, 2006]. We focus on two common examples.

Microsoft Outlook [Microsoft, 2006b] is one of the most popular email clients in commercial environments. It is part of the Microsoft Office Suite. It integrates many other functions, such as contact management, todo lists, and calendaring. A small industry exists for creating plugins that extend Outlook's functionality, such as organization tools like Nelson Email Organizer (NEO) [Caelo, 2006] or a variety of spam filtering packages.

Mozilla Thunderbird [Mozilla, 2006b] is probably the most popular open source client, created by the Mozilla Foundation. It is based on the same XUL layout engine as Firefox, which makes it easily extensible. It also follows the three-pane design.

For the last 10 years, one common feature of desktop clients has been the three-pane view. It consists of a pane for folders, a pane for folder contents, and one showing the selected email. Even though mail clients are highly configurable, this has been the standard setup for many users. Figure 3.2 illustrates the current state of desktop clients, using Mozilla Thunderbird as an example.

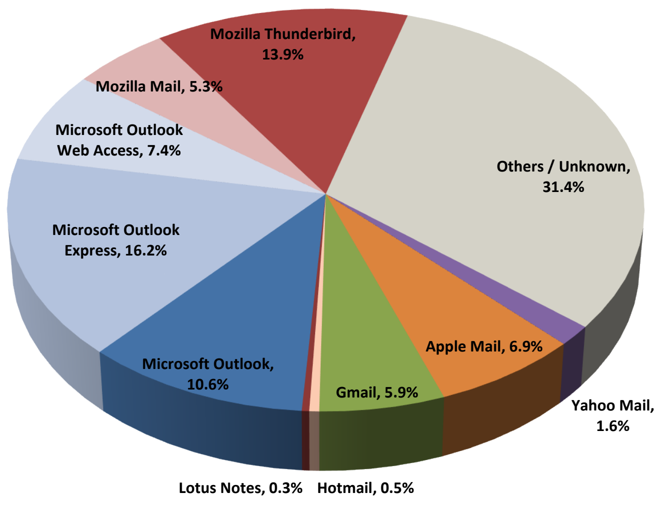


Figure 3.1: Email client usage as percentage of author's incoming mail.

3.1.2 Webmail

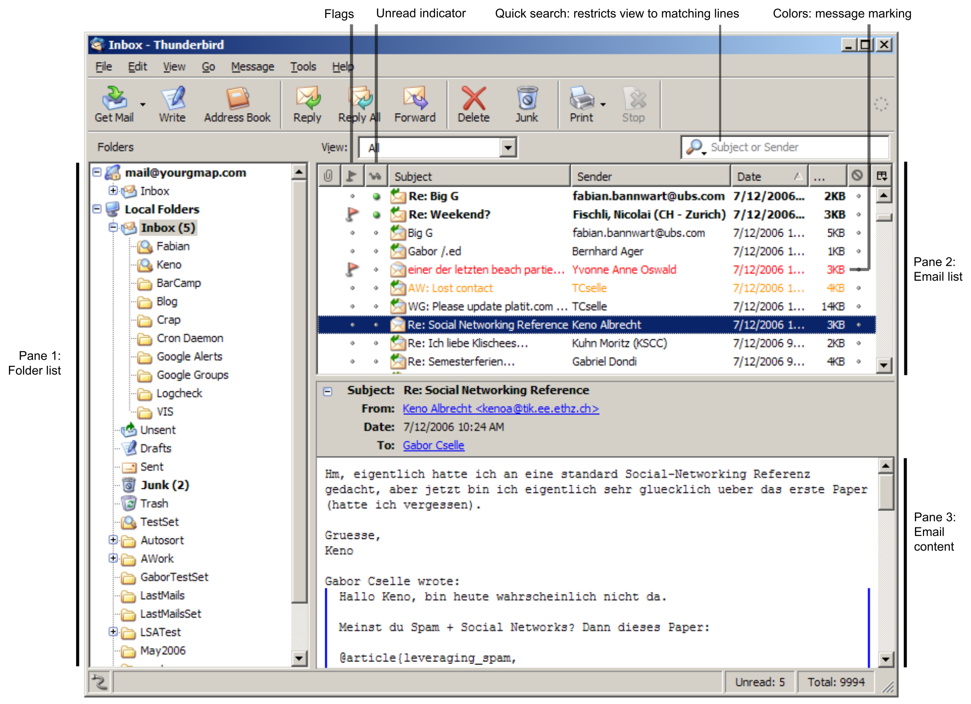
While Microsoft's Hotmail [Microsoft, 2006a] was the original web-based email client, we have chosen to feature Google's Gmail and Yahoo Mail in this category, because their interfaces are more advanced than those of their contemporaries.

Gmail [Google, 2004] groups emails by thread. The basic unit of organization is a conversation: Emails from the same thread are always shown together, as seen in Figure 3.3. Instead of the traditional foldering schema, it allows labeling messages with user-defined labels, which is a more powerful concept than folders: Each email typically resides in just one folder, but it can have multiple labels.

In contrast, the current version of Yahoo Mail [Yahoo, 2006] sticks to traditions. It closely replicates the three-pane email client, thereby showcasing the wide range of possibilities for active applications in the browser.

3.1.3 Search

A paradigm shift is taking place: Users are now turning to fast full-text search functionality when looking for old emails in their archive. Commercial applications such as Google Desktop [Google, 2006], or Windows Desktop Search [Microsoft, 2006c] create a full-text index of the documents on the user's hard drive and make them searchable in a similar way as Internet search engines do for web pages. Both products already support searching through email archives of widely used desktop clients. Fast full-text search takes off the burden from the user to sort emails into folders in a sensible fashion. However, as discussed in Section 2.3.2, the drawback of search is that the user needs to know relevant bits of information about emails she is looking for.



Subject	Sender	Date	...
Re: Big G	fabian.bannwart@ubs.com	7/12/2006...	2KB
Re: Weekend?	Fischli, Nicolai (CH - Zurich)	7/12/2006...	3KB
Big G	fabian.bannwart@ubs.com	7/12/2006 1...	5KB
Gabor /.ed	Bernhard Ager	7/12/2006 1...	1KB
einer der letzten beach partie...	Yvonne Anne Oswald	7/12/2006 1...	3KB
AW: Lost contact	TCselle	7/12/2006 1...	4KB
WG: Please update platit.com ...	TCselle	7/12/2006 1...	14KB
Re: Social Networking Reference	Keno Albrecht	7/12/2006 1...	3KB
Re: Ich liebe Klischees...	Kuhn Moritz (KSCC)	7/12/2006 9...	2KB
Re: Semesterferien...	Gabriel Dondi	7/12/2006 9...	4KB

Subject: **Re: Social Networking Reference**
From: Keno Albrecht <kenoa@tik.ee.ethz.ch>
Date: 7/12/2006 10:24 AM
To: Gabor Cselle

```
Hm, eigentlich hatte ich an eine standard Social-Networking Referenz
gedacht, aber jetzt bin ich eigentlich sehr gluecklich ueber das erste Paper
(hatte ich vergessen).

Gruesse,
Keno

Gabor Cselle wrote:
  Hallo Keno, bin heute wahrscheinlich nicht da.

  Meinst du Spam + Social Networks? Dann dieses Paper:

  @article{leveraging_spam,
```

Figure 3.2: Mozilla Thunderbird screenshot.

3.1.4 Analytics

Few solutions exist for letting users analyze and visualize their email behavior. Xobni Analytics [Xobni, 2006], a plugin for Microsoft Outlook, collects contact, message, and behavior data from Outlook, and displays statistics about the time spent reading and writing email, response times, and optimal times for contacting others.

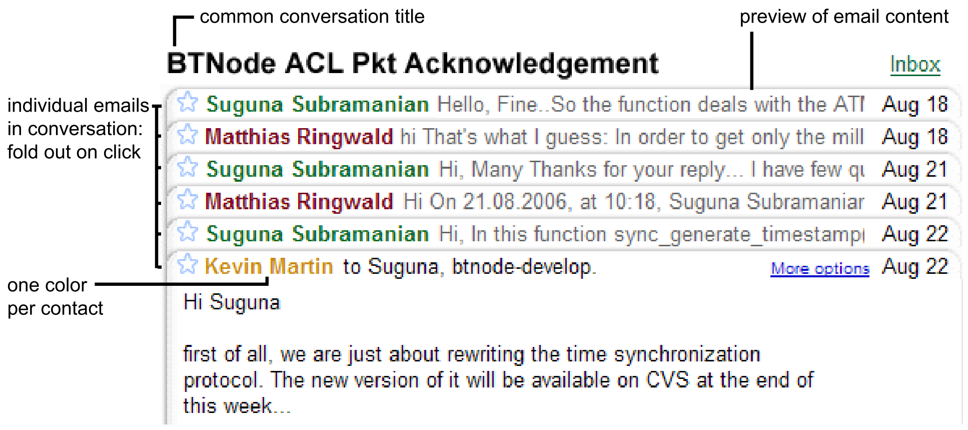
3.2 Existing Research

This section reviews work relevant to organizing individual user's email. We differentiate three areas:

Email Clients Software which implements a redesign of an email client. Two popular ideas in this field are organizing email by task or sender. We will further subdivide this category along these lines.

Visualizations This category comprises work which deals with displaying users' email data in a new way to find interesting features that were not evident through standard email interfaces.

Tools This last category covers new ways of analyzing, processing, querying, and summarizing email data. It covers a large spectrum of work which can be regarded as a 'toolbox' of methods for future email clients.


common conversation title
preview of email content
individual emails in conversation: fold out on click
one color per contact
Inbox
More options

BTNode ACL Pkt Acknowledgement

Suguna Subramanian	Hello, Fine..So the function deals with the ATI	Aug 18
Matthias Ringwald	hi That's what I guess: In order to get only the mill	Aug 18
Suguna Subramanian	Hi, Many Thanks for your reply... I have few qu	Aug 21
Matthias Ringwald	Hi On 21.08.2006, at 10:18, Suguna Subramaniar	Aug 21
Suguna Subramanian	Hi, In this function sync_generate_timestamp	Aug 22
Kevin Martin	to Suguna, btnode-develop.	Aug 22

Hi Suguna

first of all, we are just about rewriting the time synchronization protocol. The new version of it will be available on CVS at the end of this week...

Figure 3.3: Conversation-based email view in Google's Gmail.

The lines are sometimes blurred, but we chose categories according to the main usage type of each contribution.

3.2.1 Email Clients

In this first section, we will cover research that aims to redesign the email client. In contrast to the subsequent section on visualization which also covers approaches that display email or email contents, all entries here are actual software programs. They cover a large percentage of email client functionality, including the ability to send and receive messages. Table 3.2 lists the email client redesigns we will cover here.

Full-Featured Email Clients

In IBM Research's *Remail* project [Kerr and Wilkox, 2004; Rohall et al., 2004; Gruen et al., 2004], the goal was to completely rewrite and redesign the email client. The authors addressed three specific problems of current email clients:

- *Lack of context*: A new email message is a solitary drop in the flood of email, but is often related to older ones. Remail shows context by using thread arcs (described later), and grouping emails by day, name, or in manually defined collections. Remail also supports pivoting, keeping same email selected and focused while changing group order.
- *Co-opting email*: Email has many more uses than simple communication: It serves for file sharing, todo lists, calendars, and reminders. In Remail, emails can be marked by the user with todos, reminder alerts, and annotations.
- *Keeping track of too many things*: One problem in traditional email clients is that as new emails come in, important emails that form its context may drop out of the view. Remail introduces collections, semantically equivalent to Gmail's labels, which can be assigned automatically or manually.

Full-Featured	
Remail	A redesigned email client with calendaring, integrated contact management, and todo lists.
Thread-Based	
Grand Central	Groups messages into threaded conversations.
People-Based	
ContactMap	Virtual desktop with groups of icons representing individual contacts. Focus on social reminding and visual data mining.
Inner Circle	Contact lists ranked into tiers by recency and frequency of contact. Alphabetically sorted. Discovers shared nuggets of information between contacts.
Bifrost	Users manually designate important contacts. The inbox view is grouped into several categories of descending importance, based on the sender.
Task- or Activity-Based	
Taskmaster	Inbox is grouped into "thrasks," task-centric collections of email, bookmarks, and documents. Each thrask can have associated actions, flags, reminders, and deadlines.
ActivityExplorer	Groups related messages, chats, files, folders, tasks, and objects into "activity threads." Collaborative sharing of objects.
Unified Activity Management	Organizes collaborative business activities. Focus on formal business processes, not email.

Table 3.2: Overview: Email clients.

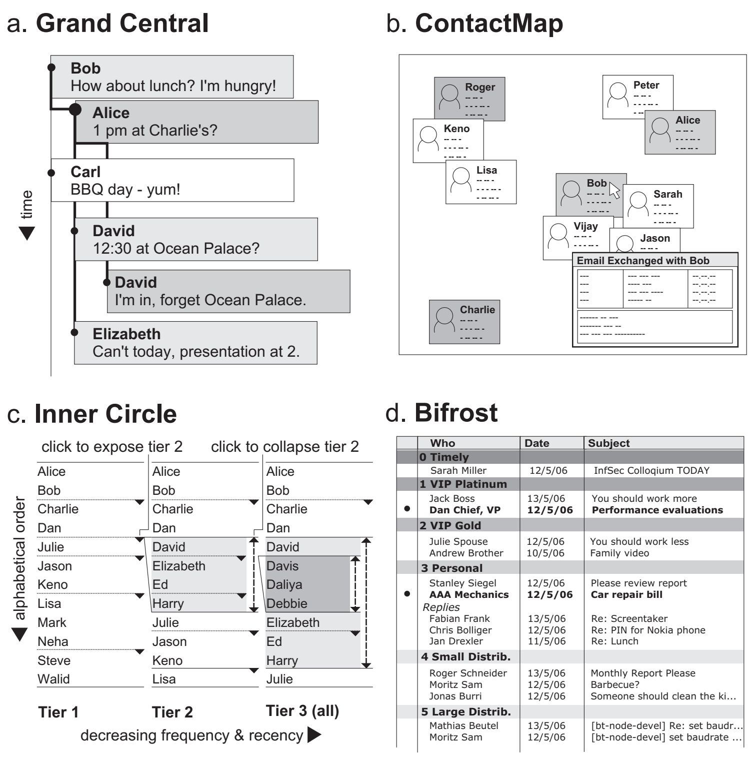
Remail also integrates chat and calendar directly into the email client. Overall, Remail introduced many interesting new features to email clients, and 4 years later, their ideas can be found in many email clients, such as Outlook or Gmail in combination with Google Calendar.

Thread-Based Email Clients

In their work on *Grand Central*, Venolia and Neustaedter [2003] cite the importance of local context when displaying an email, and propose a new method to display emails inside their thread structure (Figure 3.4.a). Their approach is slightly more advanced than Gmail's conversation view (Section 3.1.2), and the authors point out that it may be overkill for conversations that are only one or two messages long.

People-Based Email Clients

ContactMap [Whittaker et al., 2004] presents contacts in the form of colored index cards on a virtual desktop (Figure 3.4.b). Contacts can be manually grouped by moving them spatially close to each other in a 2-D space. Each contact is associated with a picture depicting the person. Clicking on a person displays all email received from the contact. The authors of ContactMap argue that it supports social reminding, social data mining, and social recommendations. Social reminding helps users honor communications and keeping in touch with



	Who	Date	Subject
	0 Timely		
	Sarah Miller	12/5/06	InfSec Colloqium TODAY
	1 VIP Platinum		
	Jack Boss	13/5/06	You should work more
●	**Dan Chief, VP**	**12/5/06**	**Performance evaluations**
	2 VIP Gold		
	Julie Spouse	12/5/06	You should work less
	Andrew Brother	10/5/06	Family video
	3 Personal		
	Stanley Siegel	12/5/06	Please review report
●	**AAA Mechanics**	**12/5/06**	**Car repair bill**
	Replies		
	Fabian Frank	13/5/06	Re: Screentaker
	Chris Bolliger	12/5/06	Re: PIN for Nokia phone
	Jan Drexler	11/5/06	Re: Lunch
	4 Small Distrib.		
	Roger Schneider	13/5/06	Monthly Report Please
	Moritz Sam	12/5/06	Barbecue?
	Jonas Burri	12/5/06	Someone should clean the ki...
	5 Large Distrib.		
	Mathias Beutel	13/5/06	[bt-node-devel] Re: set baudr...
	Moritz Sam	12/5/06	[bt-node-devel] set baudrate ...

Figure 3.4: Email Client Sketches – Thread- and People-Centered Clients: a. Grand Central, b. ContactMap, c. Inner Circle, d. Bifrost

contacts. Even without the explicit reminders in ContactMap, users may remember owed communications when they scan the visual grouping of contacts. People are very good at remembering people's faces, but not their names. ContactMap allows users to revisit contacts they may have forgotten. While the visual nature of ContactMap seems very useful, we are skeptical whether a simple two-dimensional grouping would be a good replacement for contact lists.

Inner Circle [Turski et al., 2005] unites two different organization concepts: Contact ranking and shared objects. Inner Circle is an Outlook-like email client with a redesigned contact list: Contacts are ranked into tiers by a combination of the frequency and recency of contact. The target size for the top tier is 20 to 40 contacts, depending on the list size that fits onto the screen. In effect, only the most important contacts are listed by default – the contact list can then be expanded to cover lower-tier people (Figure 3.4.c). The contact list also serves as a people picker: The email of all selected contacts is shown together in the email pane. In addition, Inner Circle supports the notion of shared nuggets: All attachments and links exchanged with the contacts currently selected by the user are displayed together in a list. There is no need for the user to manually harvest these from individual emails.

In *Bifrost* [Bälter and Sidner, 2002], the user has to manually assign contacts to predefined categories: VIP Platinum, VIP Gold, small distribution mailing lists, and large distribution mailing lists. Emails in the inbox are then automatically grouped according to the sender's assigned group (Figure 3.4.d). There are two additional categories: A "Timely" category for emails with today's or tomorrow's date in the header or another feature that is typically found in time-critical emails. A "Personal" category exists for emails that unknown contacts sent directly to the user.

Bifrost is effective because the sender of an email and the fact that it is a reply to one of the user's emails are important indicators of an email's importance [Venolia et al., 2001]. It is interesting to note that except for differentiating small and large distribution messages, this approach can already be replicated in today's email clients. Users can simply create search folders or message filtering rules which simulate Bifrost's behavior. However, this would put emails into folders and would not offer a one-page overview – instead, users would need to take explicit action.

Task- or Activity-Based Email Clients

What are the motivations behind a task-based organization of email? In Section 2.2, we noted that email is increasingly being used as a task-management application. In particular, Whittaker et al. [2006] highlight email's role as a unifying application: The inbox, folders, search, and sort are used in today's email clients to support core PIM functions of task management, personal archiving, and contact management. This subsection highlights different task- or activity-based email clients, ordered by increasing degree of collaboration: While Taskmaster only organizes a single individual's tasks, Unified Activity Management handles the business process of a whole enterprise.

Taskmaster [Bellotti et al., 2003] groups all the user's emails, drafts, attachments, and bookmarks into "thrasks." Emails in the same thread are grouped automatically, but the user still has to assign other mails, links, and deadlines manually. Thrasks can have associated actions, such as "call this person," and "review this." Users can also add deadlines and reminders to each task: They are shown as green and red bars as they approach. The great advantage of Taskmaster's approach is that items that belong together are displayed together, thereby providing context using other objects than just email.

The *ActivityExplorer* [Muller et al., 2004] was built on the idea of Taskmaster. It supports sharing objects between different users and supplies access control mechanisms. Objects can be messages, chats, files, folders, screens, and tasks. Objects can also be part of activity threads. The authors tested the system with 20 researchers and 13 student interns at IBM Research, who were pleased with the possibilities that the software offered.

Unified Activity Management is not an email client per se, but software which organizes activities for an entire corporation. Emails are imported from an external email client and are just one of many types of objects that deal with corporate activities, such as documents, contacts, or activity models. The focus is on managing business processes that involve many actors. An example is processing requests-for-proposals for large projects, where data and work are contributed from different teams inside the company.

In closing, many of the sensible proposals from research work such as Remail, Inner Circle, Grand Central, and Taskmaster, have already found their way into run-of-the-mill applications such as Microsoft Outlook or Gmail. As the feature sets of desktop applications grow, it may be necessary to reevaluate actual user needs and limit implementations to the most sensible subset.

	Type	*Organization concepts*	*Description*
Analyzing Contacts		**People**	
SNARF	plugin	metrics	Collects communication data to support triaging email.
Correspondent Crowds	graph	intensities	Scatter plot of contact intensities.
Correspondent Treemaps	interactive	hierarchies	Displays user's contacts in hierarchies.
Sudarsky and Hjelsvold [2002]	application	domain names	Orders contacts by domain name hierarchy.
Exploring the Past		**Time**	
Soylent	application	social network	Development of social network of contacts over time.
Social Network Fragments	interactive	relationships	Development of relationships of contacts over time.
Author Lines	graph	direction	Plots activity in initiating communication.
PostHistory	interactive	intensities	Calendar of email and contact intensities.
Themail	interactive	words	Relationship portraits from significant words.
Structuring Conversations		**Discussions**	
Conversation Thumbnails	interactive	thread hierarchy	Indented boxes.
ThreadArcs	graph	thread structure	Reply-to graph of emails.

Table 3.3: Overview: Visualizations.

3.2.2 Visualizations

This section explores approaches of visualizing the information present in users' past email data. Users generally keep a large percentage of their past email, indifferent of whether it is important or not. Thus, huge amounts of information are "locked up" in the email client, and are currently viewed inside the traditional sortable lists. We will cover a number of approaches for visualizing features of the user's own email, thereby unlocking potentially valuable knowledge. Table 3.3 provides an overview.

Analyzing Contacts

SNARF, the "social network and relationship finder" [Neustaedter et al., 2005b] (Figure 3.5.a), is a data mining engine for personal communications. It is a plugin for Microsoft Outlook which calculates a number of metrics for each contact, such as "emails sent to person," "replies to person," or "emails from person with Cc to the user." It displays sortable bar chart diagrams for each of these metrics, which can be used for email triage, i.e. deciding what to do with incoming email. Collections of email can also be sorted by these metrics.

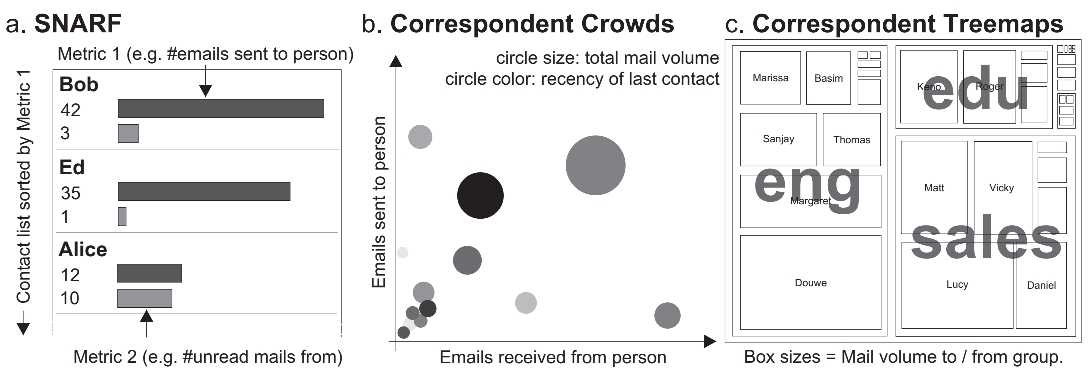

a. SNARF
Metric 1 (e.g. #emails sent to person)
Contact list sorted by Metric 1
Bob
42
3
Ed
35
1
Alice
12
10
Metric 2 (e.g. #unread mails from)
b. Correspondent Crowds
circle size: total mail volume
circle color: recency of last contact
Emails sent to person
Emails received from person
c. Correspondent Treemaps
Marissa
Basim
Keno
Roger
edu
Sanjay
Thomas
eng
Margaret
Matt
Vicky
sales
Douwe
Lucy
Daniel
Box sizes = Mail volume to / from group.

Figure 3.5: Visualization Sketches: Analyzing Contacts. a. SNARF, b. Correspondent Crowds, c. Correspondent Treemaps

For example, contacts can be sorted by the number of messages sent from the user to that person, thus bringing emails of socially important people to the top of the collection. In our opinion, this metric has limited usefulness: While we might send lots of emails to a contact, they may be mostly unimportant emails, such as jokes or news links. In comparison, we send few emails to our supervisors who are arguably some of the most important contacts.

Correspondent Crowds [Perer and Smith, 2006] (Figure 3.5.b) presents scatterplots that contrast the numbers of messages sent to a person with the number of messages received from that person. This method allows users to measure the intensity of email exchange with their contacts.

Correspondent Treemaps, also in [Perer and Smith, 2006] (Figure 3.5.c), are interactive visualizations based on a combination of domain name hierarchies and organizational chart data. Entries are displayed in a treemap [Shneiderman, 1992] and are colored based on the state of read or unread email from that person or person group. This visualization allows users to view the structure inherent in their contact list.

Sudarsky and Hjelsvold [2002] are interested in grouping contacts sensibly and viewing emails exchanged with them. In their tool, they decompose the email addresses along the top-level, domain-level, and subdomain-level parts of the hostname, and group them in a hierarchical list view. For each contact, emails can be viewed and searched as entities along a time axis. A thread-based visualization is also available.

A product in this category Xobni Analytics, which we discussed earlier in Section 3.1.4. It has the ability to plot number data such as the emails exchanged, times of email exchanges, and filter information by contact. Other than Xobni's visualizations, and some simple plugins for non-Outlook email clients, there seems to be a shortage of tools that allow end users to analyze and visualize their email data.

Exploring the Past

Soylent [Fisher, 2004] goes a step further by deriving a social network of contacts from the email repository of the user. Ties are created when two users co-occur on the "From," "To," or "Cc" lines of an email. This graph is then visualized and can be browsed along the dimension

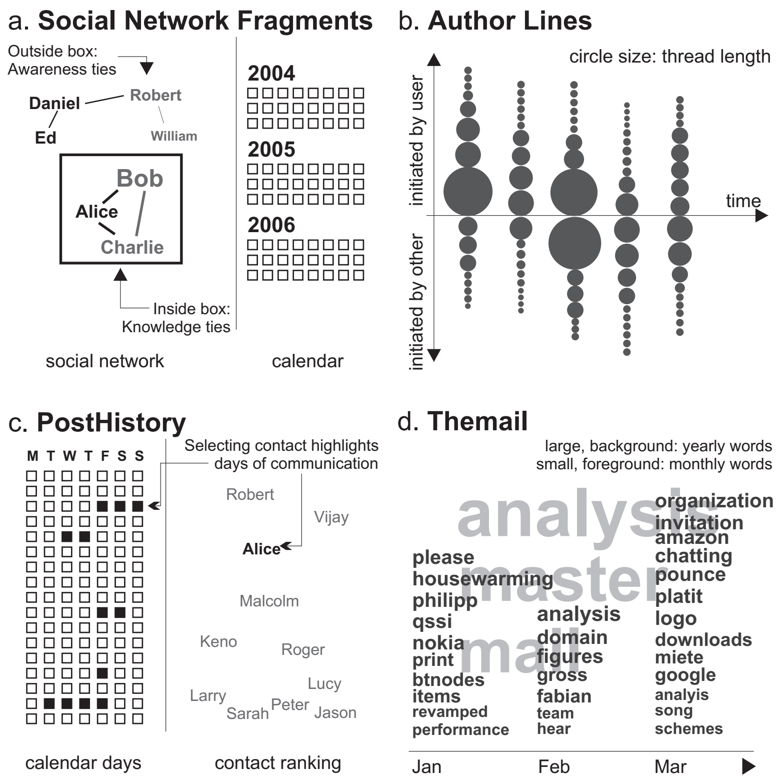


Figure 3.6: Visualization Sketches: Exploring the Past. a. Social Network Fragments, b. Author Lines, c. PostHistory d. Themail

of time.

Social Network Fragments [Viégas et al., 2004; Boyd, 2002] (Figure 3.6.a), a precursor to Soylent, is based on the same ideas: Derive a social network from local email data, and visualize this information against time. Social contacts are segmented into knowledge ties (if A sends a message to B, A "knows" B) and awareness ties (if B receives a message from A, B is "aware of A"). Based on the email address of the contact and manual input, a role for the user is derived (e.g. work email, personal email, school email), and people and their connections are color-coded in the graph. There are 24 time slices per year, for each of which the system generates one snapshot of the social network: This allows exploring how social ties formed, evolved, and collapsed over time.

Author Lines, the third contribution in [Perer and Smith, 2006] (Figure 3.6.b), are a way of plotting the intensity of contacts over a calendar year. A chart is drawn with the top half representing initiated threads, and the bottom half representing replies to threads initiated by others. Histograms are drawn consisting of bubbles, with each bubble representing one thread. Each bubble's size describes the number of messages in that thread. This visualization describes users' communication histories, and forms easily recognizable patterns for typical roles: A tech support employee will have a bottom-heavy graph, as he or she commonly responds to incoming requests, while a salesperson should have a top-heavy graph that shows his or her actively contacting potential clients.

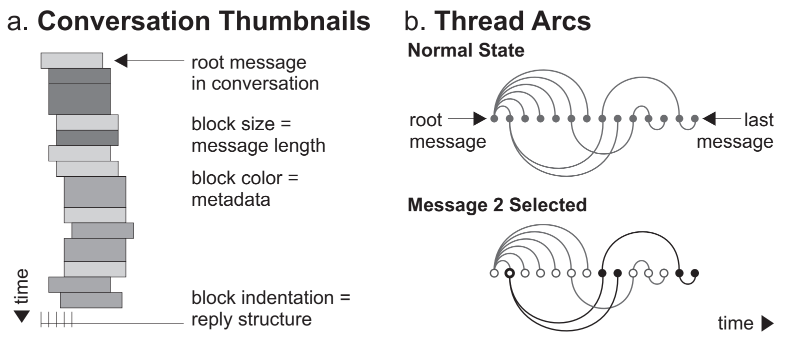

a. Conversation Thumbnails
root message in conversation
block size = message length
block color = metadata
block indentation = reply structure
time
b. Thread Arcs
Normal State
root message
last message
Message 2 Selected
time

Figure 3.7: Visualization Sketches: Structuring Conversations. a. Conversation Thumbnails, b. Thread Arcs

PostHistory [Viégas et al., 2004] (Figure 3.6.c) is another time-based visualization: An entire year is shown in a calendar. Each day is represented by a square whose size represents the quantity of mail received. PostHistory takes directionality into account: Messages where the only recipient is the user are "highly directed," while mailing list emails are not directed at all. The color of each square represents the average directionality of emails on that day. A contacts panel lists the names of people who have sent messages. A click on a name highlights the days on which communication with that person took place. A click on a day's square highlights the contacts for the day. While PostHistory's has a strong focus on contact and time, the actual contents of what was discussed with a contact are missing.

Themail [Viégas et al., 2006] (Figure 3.6.d), on the other hand, deals specifically with the content of past emails exchanged with a certain person. In its typographic visualization, the horizontal axis represents time. On the vertical axis, Themail displays distinctive words which characterize emails from a certain period: Monthly words are small words in the foreground which characterize emails from a certain month. Yearly words appear faded in the background. One drawback of Themail is that it can show only data from one relationship at a time, although there is work underway for social content visualizations [Donath, 2006].

The authors of the last three visualizations highlighted the ability of these tools to help in self-reflection – remembering the quality and texture of past experiences. By presenting them with past people, contents, and communication intensities, participants in user studies felt compelled to revisit past experiences.

Structuring Conversations

Conversation Thumbnails [Wattenberg and Millen, 2003] (Figure 3.7.a) help navigate discussions with a large amounts of messages and a nested reply structure. In a small image, every message is shown as a small rectangle, indented to account for reply structure. There is also an interactive component: As the user types words for his own post in the thread, existing messages with the same words are highlighted in the structure. The advantage to this scheme may be that users participating in such conversations are normally unaware of the tree structure: They may simply reply to the last email addressing the general topic, which is not necessarily in the right position of the subtree. These indentation schemes are therefore popular for displaying thread structure in email clients or Internet message boards.

Thread Arcs [Kerr, 2003] (Figure 3.7.b), a part of Remail which we saw in Section 3.2.1, are similar to Conversation Thumbnails in that they also visualize threaded conversations and reply structure. However, their approach is not that of displaying indented boxes, but showing all emails in a thread in a simple graph, with each email as vertex and reply relationships as edges. When an email is selected by the user, it is highlighted in the graph, along with its direct ancestors and children. This approach avoids the problem of users not replying to the correct email, but it may not a good choice for extremely long conversations, as the length of the thread may exceed screen width.

Other ideas of presenting threads and conversation exist, such as the Netscan dashboard by Smith and Fiore [2001], or Gmail's or Grand Central's discussion views, which we described in Sections 3.1.2 and 3.2.1, respectively. Researchers have also explored dealing with very large-scale conversations, such as Usenet newsgroups [Sack, 2000], with volumes falling outside the scope of a single user's email.

Overall, there has been a great number of ideas and tools that highlight certain aspects of email. These aspects are typically not visible in today's email clients. Ours was by no means a complete journey: There has also been work on exploring and visualizing email archives of entire corporations, e.g. that of Enron [Heer, 2005], but they fall outside our scope as our focus lies on the email of just one user.

3.2.3 Data Mining

Email is rich in information. In addition to unstructured content, there is also structured content that can be derived from headers. This information allows deriving meaningful information about the email behavior of the user and contacts from a large email corpus. This section will highlight some of the work in the area of email data mining: Extracting information and indicators from email that are not immediately visible from the content itself. There exists a study by Katakis et al. [2006] which gives a good summary of the field of email mining techniques. We have chosen to highlight fewer methods but go into more detail on each. Table 3.4 provides an overview of the approaches we will cover.

Grouping

Surendran et al. [2005] retrospectively group emails into "personal topics" through sophisticated clustering methods on TF-IDF vectors, which describe word significance (see Section 5.4.3). They extract noun phrases from the subject lines of each email, and present results in a personalized email / document browser. The clustering approach has a major advantage over foldering, namely that the user does not need to manually create groups and seed them with example data. We will explore this approach in more detail in Chapter 5, where we propose a similar scheme.

Newman and Blitzer [2003] aim to summarize conversations by clustering messages into approximate topical groups, and then extract short overviews. Their system uses a similar combination of TF-IDF vectors, probabilistic latent semantic analysis (PLSA, [Hofmann, 1999]) to decrease dimensionality and distance of the vectors, and a single-link clustering algorithm. To create the summary, the process then computes scores for each sentence that

Authors	*Description*
Grouping, Summarizing	
Surendran et al. [2005]	Automatic, retrospective clustering of emails into personal topics.
Newman and Blitzer [2003]	Summarizing archived discussions.
Lam et al. [2002]	Exploiting email structure to improve summarization.
Extracting Specific Content	
Carvalho and Cohen [2004]	Extracting signature and reply lines from email.
Corston-Oliver et al. [2004]	Extracting instructions to perform tasks from email.
Classifying, Prioritizing	
Foldering	Automatic foldering of email messages, treated in Chapter 4.
Dredze et al. [2006b]	Automatically classifying emails into activities.
Horvitz et al. [1999]	Classifying emails by priority.
Dredze et al. [2005]	Classifying emails by reply expectations.
Dredze et al. [2006a]	Learning whether the user wants to include an attachment.
Creating Conversation Models	
Kushmerick and Lau [2005]	Automatically extracting email activity models.
Temporal Patterns	
Kleinberg [2002]	Detection of bursts of intense email activity on given keywords.
Perer et al. [2006]	Displaying, clustering, and querying for contacts depending on the intensity of contact over time.

Table 3.4: Overview: Data Mining.

describe its potential contribution to its summary coherence and coverage. The scores are based on the lexical similarity of the sentences to the cluster centroid and their positions in the document. The focus of this work, however, lies on messages in newsgroups, not personal email messages.

Lam et al. [2002], in contrast, are specifically interested in summarizing email messages, and exploit threading reply chains for this purpose. They use standard summarization software to generate digests, but stress the need for extraneous headers, quoted text, reply information, and email signatures.

Extracting Specific Content

Carvalho and Cohen [2004] present methods to extract signature and reply lines from email: Signature lines are often configured to be added automatically to users' emails and include the contact information or the user's position inside an organization. Reply lines are lines quoted from the previous email in the thread. The features and machine learning algorithms employed in this work detect signature lines and reply lines with 97% and 98% accuracy, respectively.

Corston-Oliver et al. [2004] address the problem of poor integration between email clients and todo lists. Emails often contain speech acts that refer to todo items, such as “On the visa issue, I am positive that you need to go to the Embassy in London to get your visa stamped into your passport,” which refers to the task “Go to Embassy in London to get your visa stamped into your passport.” The authors present a system to identify speech acts in each email body sentence, classify it into a set of tasks, and reformulate them as todo items. The respective portions in each email are also highlighted, which may be very interesting by itself for scanning email contents when confronted with many messages.

Classifying, Prioritizing

A popular model of classifying emails is that of automatic foldering: A machine learning classifier learns email-to-folder mappings from a training set and is then used to automatically file newly incoming emails into folders. This approach is explored in detail in Chapter 4.

Dredze et al. [2006b] present a classifier to automatically assign emails to activities. The task-based email clients presented in Section 3.2.1 lacked sophisticated mechanisms to automatically classify emails into the tasks they managed. By combining the threading information contained in headers and metrics that compare the set of people in the activity with that of the new email, they achieve good results for this classification problem.

The Priorities system presented in [Horvitz et al., 1999] derives a means for automatically inferring the importance of an email message using a handful of carefully-picked features. Their classifier achieved very good success rates in identifying critical messages: around 1% false positives and false negatives. One specialty of this approach is the use of text-based features related to time criticality (e.g. “right away,” “as soon as possible”), with the drawback that these may need to be adjusted for each context and language in which the system is to be used.

Reply expectation prediction [Dredze et al., 2005], i.e. predicting which emails will be answered to and which ones not, could also be a useful tool. However, the results here are not as promising as for priority identification: For example, in order to find 80% of the emails that will be replied to, 50% of the emails marked will be incorrect.

The medium of email is used by many for simple file transfer. However, a very practical problem is that users may forget to include the attachment, because they are too focused on writing the email itself. To alleviate this problem, the email client may include a component that analyzes the email being composed and show an ambient indicator to remind the user that a file still needs to be attached. Dredze et al. [2006a] provide a classifier that identifies emails to which an attachment needs to be added.

Creating Conversation Models

Kushmerick et al. [2006] automatically derive conversation process models from emails. These models are comparable to finite state automata. An example is the order process at an online bookstore: The first state is an empty start state. The arrival of a “thank you for your order” email moves the process into an “order placed” state. Subsequent emails may move the process into a states named “partially shipped,” “fully shipped,” and ”done.” Past the "done" state, there may be optional states dealing with warranty clarifications, problem resolutions, and refunds. The algorithms presented there can learn process models with little pre-labeled data, but require each email in an activity contains a distinct identifier, such as Amazon shipment codes in the form "028-1428652-7533330." While deriving such models is less useful for private small-volume emails, it can be very interesting for large organizations trying to track the status of business processes in a sea of incoming and outgoing email.

Temporal Patterns

A common phenomenon in emails as well as news texts are the "bursts of activity" about certain topics that appear, grow in intensity for a period of time, and then fade away. Kleinberg [2002] develops a formal model based on infinite-state automata. In each state of the automaton, messages are generated on a topic at increasingly intense rates.[1] From a corpus of past emails, one can derive the state of the automaton for each time interval, thereby identifying bursts of activity: They are characterized by the fact that the automaton switches to a high-up state for some period of time. This identification scheme could be very useful for retrospective analysis of emails or on-line alerting of users of the surging popularity of certain topics.

Instead of exploring the activity related to topics, Perer et al. [2006] focus on the user's contacts and the development of relationships. From the user's archive, it is possible to generate a graph with the number of emails exchanged with each contact over time. This graph then characterizes the relationship over long terms: For example, a graph that surged in early years and then drops may represent a good friend from school with whom emails are now exchanged only during the holiday season. Such relationship graphs may also be useful in clustering people into groups: Similar people will be characterized by similar activity patterns. Another application may be searching: From visually drawing an activity profile into a chart, the user specifies a relationship profile query. The result of this query would be a set of contacts with a similar activity profile.

Existing research in data mining on email has exposed a variety of interesting and successful techniques. However, few of these techniques have found their way into standard email clients.

3.3 Conclusion

This section was intended to show existing work: What is the current state of commercial email clients? What improvements does research work suggest? What visualization approaches could be used to help users find valuable, previously invisible data in their email? What tools are available for data mining in email repositories? Our tour through existing work showed that there are many interesting ideas and approaches, all waiting to be used in real-world applications.

[1]For Kleinberg, the fact that an email belongs to a topic when a combination of certain topical words, for example "final exams" appears in the email – however, this simple model may be extended to cover more sophisticated approaches.

Lastly, we should note that tools influence behavior: For example, when users are given tools that strongly focus on threads, such as Gmail, they will make sure that similar messages end up in the same thread, to profit from its thread-based view. More sophisticated tools will result in more sophisticated uses that create new needs.

4 Automatic Foldering

Automatic foldering is a commonly proposed solution to email overload. In this chapter, we evaluate its effectiveness.

Many users today organize their incoming email into a hierarchy of folders. However, manually moving emails into a folder takes effort and is a mentally difficult task. Automatic foldering uses machine learning algorithms to automate the moving emails into the correct folder.

Using data from own emails and the Enron corpus, we evaluate the performance of a number of classifiers. These classifiers – Naive Bayes, Sender-based, and Ripper – were chosen for simple implementability or, in the case of Ripper, their interesting approach. While constructing classifier input data, we encountered some questions concerning preprocessing. We evaluated our decisions in the light of their effect on classifier performance.

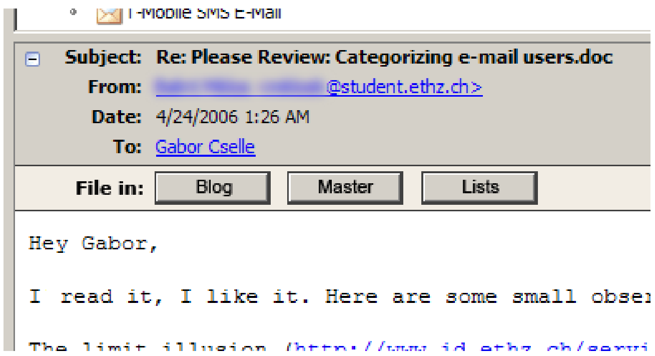
4.1 Related Work

We review existing tools for automatic foldering and compare classification accuracies found in related work.

4.1.1 Classification Tools

There exist programs for automatic foldering in research and for end users, which we will review here.

POPFile [POPFile] works as a POP proxy: It can be set up by the user to fetch its emails from the email server. It then decides on the correct folder and adds a header string indicating the correct folder. The user can then set up filters in the email client to use these headers for moving messages into the correct folder. Training and misclassifications are handled through a web interface.


T-Mobile SMS E-Mail

Subject: Re: Please Review: Categorizing e-mail users.doc
From: [redacted]@student.ethz.ch>
Date: 4/24/2006 1:26 AM
To: Gabor Cselle

File in: Blog Master Lists

```
Hey Gabor,

I read it, I like it. Here are some small obse

The limit illusion (http://www.id.ethz.ch/servi
```

Figure 4.1: SwiftFile illustration: The mail client displays buttons for the the most likely folders in which to file an incoming message.

SwiftFile (formerly known as MailCat [Segal and Kephart, 1999]) is a system designed at IBM Research for Lotus Notes. It shows a method for compensating for the low accuracy of automatic classifiers: Instead of instantly filing an email away, SwiftFile shows buttons with the three most likely folders it may land in: Figure 4.1 shows an example. The likelihood that at least one of the three buttons will be the correct folder is 80-90%. For a large majority of emails, instead of dragging them to the right folder, users only need to click a button.

4.1.2 Classification Accuracies

There exists a great amount of existing work on automatic foldering, using a wide variety of approaches on different corpora and with different results. We have selected a number of papers and compare their results in Table 4.1. (Some accuracy measurements were manually transcribed from diagrams.)

With the exception of Bekkerman et al. [2005], who use the Enron corpus, all of these studies use proprietary corpora with 1 to 5 users each. As the comparison metric, we decided to use classification accuracy, which gives the ratio of correctly classified instances to the total number of instances in the test set. Some studies, such as the one in [Klimt and Yang, 2004], do not give accuracy numbers, and were therefore not included in our comparison.

In evaluating the performance of classifiers, a wide variety of training / test splits are used: From simple 80%/20% splits to more elaborate approaches such as Crawford et al.'s sliding window of 40 emails for training and 10 emails for test. Bekkerman et al. iterate through the corpus with training sets that are increasing multiples of 100, and the test set containing the next 100 emails. In each case, the training set contains only emails that arrived before those in the test set.

			Accuracy		
Method	*Method*	*Test Split*	*Top 1*	*Top 3*	*Top 5*
Segal and Kephart [1999]	TF-IDF	70%/30%	70.2%	86.2%	92.5%
Brutlag and Meek [2000]	TF-IDF	80%/20%			82.6%
	Lin. SVM				83.0 %
	Unigram				81.6 %
Bekkerman et al. [2005]	MaxEnt	100N/100	70.9 %		
	Bayes		57.5 %		
	SVM		71.5 %		
	Winnow		68.6 %		
Cohen [1996]	TF-IDF	progressive	92.7%		
	Ripper	percentages	95.3%		
Rennie [2000]	Bayes	leave-one-out	88.5 %		
Crawford et al. [2001]	Sender	Win40/10	45.0 %		
	Keyword		72.0 %		
	TF-IDF		68.0 %		
	Dtree		63.0 %		

Table 4.1: Results for automatic foldering in existing literature.

4.2 Classification Methods

As we saw in the related work section, there exists a large variety of classifiers based on machine learning methods to sort email. We chose to evaluate several variants of three methods, which we selected according to several criteria. Naive Bayes and the sender-based classifier are simple and all easy to implement. In particular, Graham's Bayesian filter variant, whose guidelines of preprocessing we followed, is a very popular real-world implementations of spam filter classifiers: For example, the spam filter implementations of Mozilla Thunderbird [Mozilla, 2006b] and Spamato [Spamato, 2006] use variants of this approach. We also selected Ripper, a rule generating classifier, whose interesting property is that it generates rules which can easily be reproduced by the user in message filter rules – a tool supported by practically every popular email clients.

Unlike methods that take relatively large amounts of time for training (e.g. support vector machines), all methods evaluated here are quick to train and could therefore be used in popular mail clients.

Formally, we define the problem of automatic foldering as follows: We are given a training set $\mathcal{D}_{train} = \{d_1, ...d_{|\mathcal{D}|}\}$ consisting of labeled email documents d_i. Each document belongs to one of the classes $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$. The goal of automatic foldering is to create a learning algorithm which, given the training set $\mathcal{D}_{train}$ generates a classifier $c : \mathcal{D} \rightarrow \mathcal{C}$ that can accurately classify unseen documents.

4.2.1 Naive Bayes Classifier

Naive Bayes classifiers are often used as a baseline in text classification because they are fast and easy to implement.

We use a multinomial Bayesian classifier, as described by [Rennie, 2000] and [McCallum and Nigam, 1998] and implemented in Weka [Witten and Eibe, 2005]. This model uses a "bag of words" representation for documents, which contains all the words in the document, and accounts for multiple occurrences of the same word. We define $tf_{i,t}$ to be the term frequency: the number of times word w_t occurs in document d_i. We denote the vocabulary of all words as a set V. A parameter θ describes a mixture model of the classes. We further assume that the occurrence of each word is statistically independent from every other.

The core of any Bayesian classifier is the fact that for hypothesis H and evidence E, the probability of H being true given E can be expressed as the probability of E given H.

$$P(H|E) = \frac{P(H)P(E|H)}{P(E)} \tag{4.1}$$

In document classification, the hypothesis H is that a document was "generated" by class c_j. The evidence is the document d_i and the counts $tf_{i,t}$ for the words w_t it contains. We assume that an unseen model θ, whose parameters $\hat{\theta}$ we want to estimate, has generated the document. Thus, analogously to equation (4.1), the probability that a new document d_i belongs to class c_j is expressed as:

$$P(c_j|d_i;\hat{\theta}) = \frac{P(c_j|\hat{\theta})P(d_i|c_j;\hat{\theta})}{P(d_i|\hat{\theta})} \tag{4.2}$$

For our classification, we can disregard the term $P(d_i|\hat{\theta})$, as it is constant over all classes, yielding:

$$P(c_j|d_i;\hat{\theta}) = P(c_j|\hat{\theta})P(d_i|c_j;\hat{\theta}) \tag{4.3}$$

However, we need to calculate the values on top. We start with the first component of equation 4.3, the prior probability of class c_j. Using $P(c_j|d_i) \in \{0,1\}$, the labels from the training data, it is estimated as:

$$P(c_j|\hat{\theta}) = \frac{\sum_{i=1}^{|\mathcal{D}|} P(c_j|d_i)}{|\mathcal{D}|} \tag{4.4}$$

In effect, the prior probabilities mirror the class occurrences from the training data.

Next, we estimate $P(d_i|c_j;\hat{\theta})$, the probability of a document given its class. With $|d_i|$ the number of words in document i, it is simply the multinomial distribution:

$$P(d_i|c_j;\hat{\theta}) = P(|d_i|)|d_i|!\prod_{t=1}^{|V|}\frac{P(w_t|c_j;\hat{\theta})^{tf_{i,t}}}{tf_{i,t}!} \tag{4.5}$$

For our classification, we can disregard the values of $P(|d_i|)$ and $|d_i|$ since they are constant for different values of c_j. For equation (4.5), we are missing $P(w_t|c_j;\hat{\theta})$, the likelihood of a word w_t occurring in class c_j, which we estimate as follows:

$$P(w_t|c_j;\hat{\theta}) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} tf_{i,t} P(c_j|d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|\mathcal{D}|} tf_{i,s} P(c_j|d_i)} \tag{4.6}$$

Plugging the results of equations (4.5) and (4.4) into (4.3), we now classify document d_i into class c_j, with index j:

$$j = \arg \max_{j=1..|\mathcal{C}|} P(c_j|d_i;\hat{\theta}) \tag{4.7}$$

4.2.2 Rule Generation: Ripper

Unlike statistical algorithms like Naive Bayes, Ripper is a rule-based machine learning system. As other rule-based learners, Ripper grows rules in a greedy fashion guided by an information gain heuristic. It is comparable in accuracy to similar rule-based algorithms, but is significantly more efficient [Cohen, 1996].

A partial example for the rules that Ripper generates for the user "beck-s" from the Enron corpus:

```
enron_travel_club_sc :- 'X-From*teal@enron.com'
publications_sc :- 'From*power@enron.com' OR 'derivatives'
corporate_sc :- 'From*announcements@enron.com'
```

For example, the second item means that if the "From" header of an email contains the token "power@enron.com" or the body contains the token "derivatives," the email should be filed in the folder "publications_sc."

4.2.3 Sender-Based Classification

In a fourth, very simple approach, we base foldering decisions solely on the sender of the incoming mail. For each sender, a rule is created that filters new messages into the folder that the sender most commonly goes into. This is equivalent to the "Sender" classifier in Crawford et al. [2001].

4.3 Corpus

For our experiments, we used two users from the popular Enron corpus [Klimt and Yang, 2004]. This corpus, which was subpoenaed by the US government during investigations of the Enron bankruptcy, is now in the public domain for research uses. It contains 200339 emails belonging to 158 users in total. These two users, which were also included in another study of email foldering by Bekkerman et al. [2005], both had a relatively large number of folders with many messages. We cleaned these Enron email repositories by excluding non-topical folders, such as "Drafts," "Notes," "Sent," and "Inbox." From a look at the data, it appeared to be spam-free.

In addition, we added a corpus constructed from own email data with 11 folders whose types and descriptions are given in Table 4.2. This data contains 5280 emails from the time period between October 2004 to March 2006. As we are not interested in spam filtering, we removed all spam mails from this corpus. Table 4.3 gives an overview of all datasets we used.

Name	*Size*	*Description*
ETH Lectures	435	Lectures and classes
ETH Projects	458	Labs and theses
ETH Organization	75	Organizational emails
ETH Newsletters	413	ETH Newsletters
Blog	136	Comments and reviews of blog entries
Work	390	Emails related to part-time work
Internship	520	Emails related to an internship
Travel	146	Trip planning for 9 different trips
Private	2050	Emails from friends
Newsletters	355	Non-ETH newsletters
Other	302	Receipts, newsletters, registration emails, usenet post answers
Total	5280	(average: 480, std: 540.63)

Table 4.2: Description of folders in proprietary corpus.

	Messages	*Folders*	*Smallest folder size*	*Largest folder size*
Proprietary Corpus	5280	11	75	2050
Enron: farmer-d	3710	26	5	1192
Enron: kaminski-v	4471	39	3	1159
Average	4487	25.3	27.6	1467

Table 4.3: Corpus statistics.

4.4 Preprocessing Emails

Our tokenizer largely follows the guidelines of the “Plan for Spam” tokenizer, described in [Graham, 2002, 2003], with the exception of handling HTML:

1. We scan the entire text, including headers, of an email.

2. Tokens may include alphanumeric characters, dashes, dollar signs, equal signs, and @ signs to be part of tokens. Everything else is a token separator, with two exceptions: Periods and commas are constituents if they occur between two digits.

3. Tokens that occur within the To, From, Subject, and Return-Path lines, or within URLs, are marked accordingly: For example, foo in the Subject line becomes Subject*foo, and http://www.gaborcselle.com/ anywhere in the body becomes Url*http://www.gaborcselle.com/.

4. We eliminate tokens that are all digits.
5. We remove all HTML tags and JavaScript.
6. In addition, all words are lowercased.
7. We do not stem words to their base forms.

We will refer to the combination of preprocessing parameter choices as the *Baseline preprocessor*. We will revisit items 5-7 in the additional experiments of Section 4.7.

4.5 Evaluation Methodology

For our evaluations, we chose a simple time-based 80%/20% split into training data: After sorting the messages according to their time stamp, we train the classifier with the first 80% of all messages and test it on the last 20% messages. We ignore test messages in folders that do not exist in the training data. In these cases, the user would manually create a folder when he or she discovers that it is needed.

As our quality metric, we chose accuracy, which is the ratio of correctly classified instances (true positives) to the total number of instances N in the test set.

$$Accuracy = \frac{TP}{N}$$

4.6 Results

The results of our baseline methods are given in Table 4.4 and Figure 4.2.

	Naive Bayes	*Ripper*	*Sender*
Proprietary Corpus	58.8	64.4	59.8
Enron: farmer-d	64.0	52.9	56.7
Enron: kaminski-v	42.4	38.8	25.8
Average	55.1	52.0	47.4

Table 4.4: Baseline accuracies.

Surprisingly, all three approaches have comparable performance on our data sets. While Ripper exceeds on the proprietary dataset, it performs worse than Naive Bayes on user “farmer-d.” One surprise is the performance of the sender classifier: On our proprietary corpus, this simple method has a higher accuracy than Naive Bayes!

Overall, these results may be disappointing, especially the bad performance for user “kaminski-v.” A look at [Bekkerman et al., 2005] confirms these findings: Naive Bayes also performed badly in their evaluations.

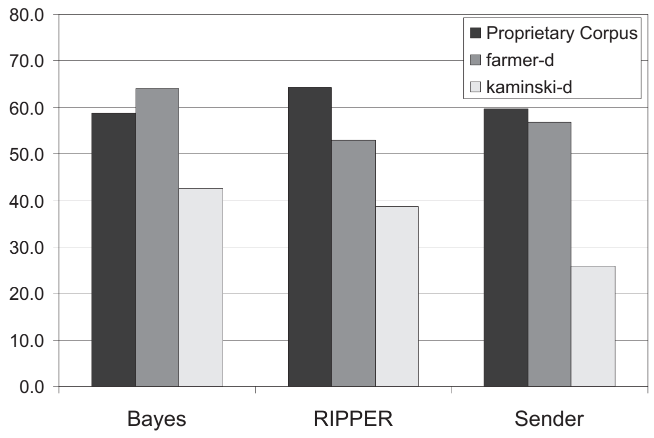


Figure 4.2: Baseline accuracies

4.7 Additional Experiments

We also carried out a number of additional experiments on our corpora, to test the consequence of changes to the baseline methods.

4.7.1 Preprocessing Choices

Our choices of preprocessing methods influence the data. For example, deciding to lowercase all words decreases diversity and quantity of tokens. We decided to review the impact of these choices on classification accuracy with a few simple experiments.

Should Words be Lowercased?

Was the choice of lowercasing words correct? Table 4.5 lists results from an experiment we carried out over our corpora.

	Naive Bayes		*Ripper*	
Token Case	*Lowered*	*Preserved*	*Lowered*	*Preserved*
Proprietary Corpus	58.8	59.2	64.4	61.1
Enron: farmer-d	64.0	64.2	52.9	49.4
Enron: kaminski-v	42.4	43.8	38.8	35.4
Average	55.1	55.7	52.0	48.6

Table 4.5: Accuracies when lower-casing all tokens vs. preserving case.

From our results, accuracy is not strongly affected by lower-casing all tokens, although Ripper's performance is slightly decreased with case preserved

Should Tokens be Stemmed?

We could use simple Porter stemmers to stem words into their original forms. This converts "walks," "walking," and "walker" to the base form of "walk." Since our proprietary corpus contains both English and German emails, we had to apply appropriate stemmers for each language. We used TextCat [Cavnar and Trenkle, 1994] to automatically determine the language of each email.

	Naive Bayes		*Ripper*	
Token Form	*Original*	*Stemmed*	*Original*	*Stemmed*
Proprietary Corpus	58.8	56.8	64.4	60.5
Enron: farmer-d	64.0	65.1	52.9	52.3
Enron: kaminski-v	42.4	43.0	38.8	40.0
Average	55.1	55.0	52.0	50.9

Table 4.6: Accuracies when preserving the original form of tokens vs. stemming them.

As seen in Table 4.6, stemming tokens does not seem to significantly affect classification performance.

Should HTML be Removed?

HTML entities are often very good indicators for spam, but we chose to remove HTML in foldering results, with the suspicion that emails with similar HTML tokens – e.g. messages composed in the same client – would often matched together by accident.

	Naive Bayes		*Ripper*	
HTML	*Removed*	*Kept*	*Removed*	*Kept*
Proprietary Corpus	58.8	49.1	64.4	62.5
Enron: farmer-d	64.0	64.0	52.9	52.9
Enron: kaminski-v	42.4	42.4	38.8	38.8
Average	55.1	51.9	52.0	51.4

Table 4.7: Accuracies when lower-casing all tokens vs. preserving case.

We performed an experiment in which we kept HTML tokens present in emails. As argued by Graham [2003], these tokens from HTML source code are often helpful in identifying spam. Our results for classification are shown in Table 4.7. The Enron corpus contains only very few HTML mails and classification performance is not affected by it. However, classification performance significantly decreases in the case of Naive Bayes on our own corpus. Our initial choice of removing HTML tokens proved to be correct.

4.7.2 Ripper on Enron Corpus

A final measurement contrasts the results of applying Ripper to the results obtained by Bekkerman et al. [2005] for several users from the Enron corpus.

In order to be able to contrast Ripper's Enron performance against other approaches, we decided to slightly change our evaluation methodology to those of Bekkerman et al. [2005], who evaluated the performance of several classifications on the Enron corpus.

The main change to our original methodology is that we use an incremental time-based split instead of re-training after each email, we retrain our classifier once every 100 emails, and use it to classify the next 100 messages.

Like Bekkerman et al., we chose the seven users from Enron corpus with the largest number of folders. We removed non-topical folders (Inbox, Sent, Trash, Notes, and Drafts), removed folders with only 1 or 2 messages, and ignored folder hierarchies. We removed attachments and did not apply stemming. However, unlike Bekkermann et al., we did not remove the 100 most frequent words – these would be ignored by Ripper anyway as they would not constitute good classification features.

User	*Folders*	*Messages*	*MaxEnt*	*Naive Bayes*	*SVM*	*WMW*	*Ripper*
beck-s	101	1971	55.8	32.0	56.4	49.9	46.3
farmer-d	25	3672	76.6	64.8	77.5	74.6	66.6
kaminski-v	41	4477	55.7	46.1	57.4	51.6	42.3
kitchen-l	47	4015	58.4	35.6	59.1	54.6	42.4
lokay-m	11	2489	83.6	75.0	82.7	81.8	72.2
sanders-r	30	1188	71.6	56.8	73.0	72.1	58.1
williams-w3	18	2769	94.4	92.2	94.6	94.5	94.8
Average	39	2940.1	70.9	57.5	71.5	68.4	60.4

Table 4.8: Results for the Ripper classifier on the Enron corpus. All other results taken from [Bekkerman et al., 2005]

Our results in Table 4.8 show that while Ripper once again outperforms Naive Bayes, it does not exceed the results of the advanced classifiers. However, Ripper's advantage over more sophisticated approaches remains: Users can more easily understand classification decisions, since they are readable in rule form.

4.8 Conclusion

Any results in automatic foldering are strongly dependent on the corpus used for measurements. Our corpora are those of a student and of two employees in a corporate environment. These are representative for contexts in which automatic foldering applies, but do not cover a large portion of the potential userbase.

Similarly, in our evaluations, we focused on simple classifiers, as they could be easily implemented by real-world developers interested in creating simple automatic foldering schemes. However, we were disappointed by their classification performance.

As future work to improve performance, it could be interesting to apply a part-of-speech tagger to the text data. This would allow testing classification accuracies with only nouns or common noun phrases as features. Previous work on standard text corpora, such as Scott and Matwin [1999] have shown that these methods improve classification performance. Some own initial research on the Enron Corpus has suggested that foldering behavior is subject to bursts: For certain periods, a large proportion of incoming emails is assigned to a single folder. After the burst is over, incoming emails become more evenly spread. This phenomenon may also be used to boost accuracy rates.

As mentioned in Chapter 2, automatic foldering has several drawbacks. Even if automatic foldering worked perfectly, there may still be ambiguities about which of the existing folders fits a given email. In addition, in creating folders, the user needs to account for future needs: Any email may be the start of a new conversation which will eventually require a folder for itself. In addition, new folders have to be given a memorable name. Lastly, users distrust classifiers and would like to see them before moving them anywhere. In particular, none of our classification accuracies are sufficiently high that all users would be comfortable with entrusting their email to such schemes. In Chapter 5, we explore a completely automatic scheme of organizing emails which will allow us to circumvent many of the drawbacks of automatic foldering.

5 Topic Detection and Tracking

This chapter presents 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.

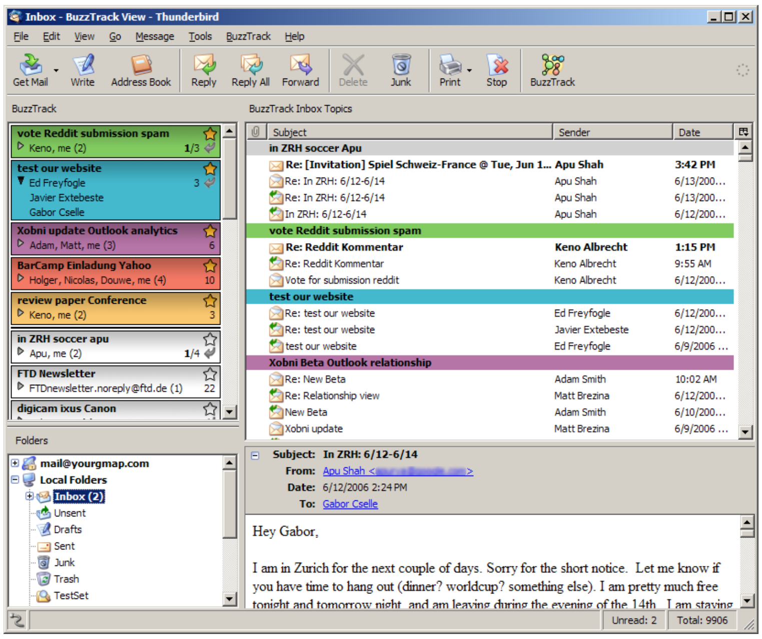
5.1 Introduction

What are the design considerations behind BuzzTrack and how did we arrive at our solution? As we observed in Chapter 2, a paradigm shift is taking place: Users are now turning to fast full-text search functionality when looking for old emails in their archive. Therefore, we chose to focus on sensibly organizing the unstructured inbox, which remains a challenge. It is not a problem that can be solved with search, as users require an overview of different incoming items, not an answer to specific queries. In particular, we identified two problems that email users face: First, they must deal with ever-increasing amounts of email. In addition, users have no overview of the context of messages. Inbox data is typically viewed in a list sorted by arrival time: There is no sense of importance, coherence, or context.

We explored several approaches of solving email overload in Chapter 3, including automatic foldering, which we reviewed in detail in Chapter 4. We rejected automatic foldering because of conceptual problems: First, accuracy rates are too low to be sufficiently reliable for users. Second, this method moves email out of the inbox, and therefore, out of sight: Users may never see misclassified important emails that were lost in the folder hierarchy. In Section 3.2.1, we examined task-based email clients, which offer a good approach of solving these problems. However, the task metaphor is too constricting to cover all email: A large proportion of emails in the inbox may not directly be related to business processes or task items. In addition, while schemes have been proposed to automatically assign emails to tasks or activities Dredze et al. [2006b], current task-based email clients do not offer this functionality.

This train of thought led us to address the 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.

With BuzzTrack, 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 5.1.



vote Reddit submission spam
Keno, me (2) 1/3

test our website
Ed Freyfogle 3
Javier Extebeste
Gabor Cselle

Xobni update Outlook analytics
Adam, Matt, me (3) 6

BarCamp Einladung Yahoo
Holger, Nicolas, Douwe, me (4) 10

review paper Conference
Keno, me (2) 3

in ZRH soccer apu
Apu, me (2) 1/4

FTD Newsletter
FTDnewsletter.noreply@ftd.de (1) 22

digicam ixus Canon

Subject	Sender	Date
in ZRH soccer Apu		
Re: [Invitation] Spiel Schweiz-France @ Tue, Jun 1...	**Apu Shah**	**3:42 PM**
Re: In ZRH: 6/12-6/14	Apu Shah	6/13/200...
Re: In ZRH: 6/12-6/14	Apu Shah	6/13/200...
In ZRH: 6/12-6/14	Apu Shah	6/12/200...
vote Reddit submission spam		
Re: Reddit Kommentar	**Keno Albrecht**	**1:15 PM**
Re: Reddit Kommentar	Keno Albrecht	9:55 AM
Vote for submission reddit	Keno Albrecht	6/12/200...
test our website		
Re: test our website	Ed Freyfogle	6/12/200...
Re: test our website	Javier Extebeste	6/12/200...
test our website	Ed Freyfogle	6/9/2006 ...
Xobni Beta Outlook relationship		
Re: New Beta	Adam Smith	10:02 AM
Re: Relationship view	Matt Brezina	6/12/200...
New Beta	Adam Smith	6/10/200...
Xobni update	Matt Brezina	6/9/2006 ...

mail@yourgmap.com
Local Folders
Inbox (2)
Unsent
Drafts
Sent
Junk
Trash
TestSet

Subject: **In ZRH: 6/12-6/14**
From: Apu Shah <[redacted]>
Date: 6/12/2006 2:24 PM
To: Gabor Cselle

Hey Gabor,

I am in Zurich for the next couple of days. Sorry for the short notice. Let me know if you have time to hang out (dinner? worldcup? something else). I am pretty much free tonight and tomorrow night, and am leaving during the evening of the 14th. I am staying

Unread: 2 Total: 9906

Figure 5.1: Inbox view with BuzzTrack plugin; topic sidebar on the left, grouped inbox view on the right.

A large proportion of users live with a flat inbox structure, with just a few folders and filter rules for newsletters [Neustaedter et al., 2005a]. 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 [NIST, 2004] aimed at organizing news articles into coherent topics. As in TDT for news articles, a text similarity measure is at the core of our clustering algorithm. However, email text is often of lower quality than newspaper content. On the other hand, email is richer in non-textual information: We can work with 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 Chapter 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.

5.2 Related Work

The problem of email overload, which we reviewed in Chapter 2 is now widely acknowledged and has found a lot of attention even in the popular media. We explored a variety of possible solutions in Chapter 3, including task-based [Whittaker et al., 2006], activity-based [Dredze et al., 2006b], priority-based, [Horvitz et al., 1999], and sender-based [Bälter and Sidner, 2002] organization schemes. For example, Dredze et al. [2006b] 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. [2005]. 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 and less computationally intensive schemes of topic clustering and labeling. Giacoletto and Aberer [2003] propose a system that automatically creates folders using similar methods as ours, but they resort to a manual evaluation of the resulting clusters. Unlike BuzzTrack, their tool is not integrated into the mail client.

Automatic foldering (Chapter 4, [Bekkerman et al., 2005; Segal and Kephart, 1999; Boone, 1998]) 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 [Bälter and Sidner, 2002].

The work presented here uses the same techniques as topic detection and tracking for news articles [NIST, 2004]. Allan et al. [2005] investigated making this technology more accessible for normal users, 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 [Bellotti et al., 2003], which helps users group emails and other information by outstanding task, and Bifrost [Bälter and Sidner, 2002], 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.

5.3 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 5.1 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.

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 5.2 provides an example. When the user clicks on a sidebar entry, the email list scrolls to the topic.

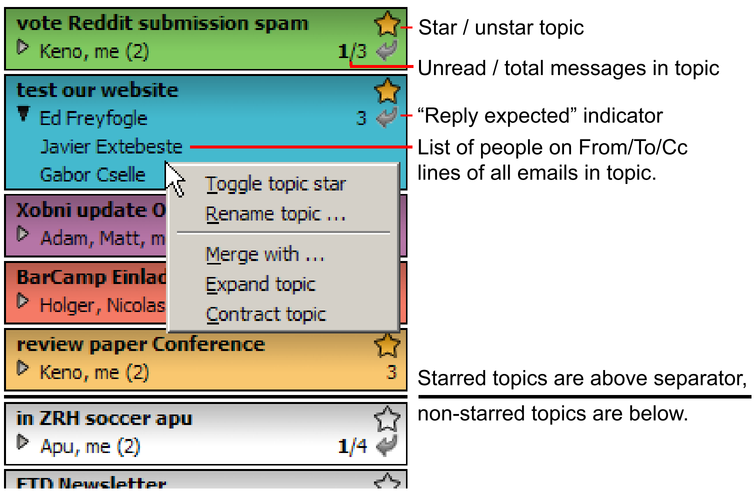


Figure 5.2: BuzzTrack sidebar.

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 [Whittaker and Sidner, 1996]. 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.

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 [Albrecht et al., 2005].

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

5.4 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-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.

5.4.1 Single-Link Clustering

In single-link clustering, a new email is compared with every other email in the inbox and assigned to the cluster of the email to which it has the highest similarity. The clustering step consist of a search over all emails in the inbox and determining the decision score, which measures similarity, with each of them. In the following sections, we discuss how we measure similarity using several features and combine them to the decision score used for clustering.

Manning and Schütze [1999] point out that single-link clustering forms clusters with good local coherence, since the similarity function is locally defined. However, clusters can be elongated or “straggly”: As a topic develops over time, newly incoming emails may be further and further away from the original email that started the topic. We have found single-link clustering to perform well on our problem, but it may be useful to explore other, more complex clustering schemes in the future.

5.4.2 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 [Natural Language Toolkit].
- We run the full text of subject and body through TextCat [Cavnar and Trenkle, 1994], 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.

5.4.3 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. These weights are determined with a TF-IDF metric. This scheme estimates the importance of a word in a document: It considers words important if they appear often in the current document but seldom in other documents. For a preprocessed email m_j, we refer to the term frequency of term t_i as $tf_{i,j}$ – this is an occurrence count for the word in the given document. We denote the document frequency of t_i as df_i – this is a count of how many documents the word occurs in. 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:

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

This combination of weight and text similarity measures is usually referred to as "ltc" in standard notation [Manning and Schütze, 1999].

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, Cc, and Bcc 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 [Dredze et al., 2006b]. 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 [Neustaedter et al., 2005b; Dredze et al., 2006b], 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}.

[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.

- *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/Bcc 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/Bcc 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.

5.4.4 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 a 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 [Platt, 1998] ("SMO"), as implemented in the Weka toolbox [Witten and Eibe, 2005].

5.4.5 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 [Kalman and Rafaeli, 2005] 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 such as 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.

5.5 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 [Surendran et al., 2005], 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-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-IDF values, and likely to appear in the topic label. In previous work [Surendran et al., 2005; Allan et al., 2005], 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-IDF weights, set the label to the common subject (with "Re," "Fw," "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-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 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-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 5.3 provides an example graph and result.

Note that we need to keep an extra TF-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

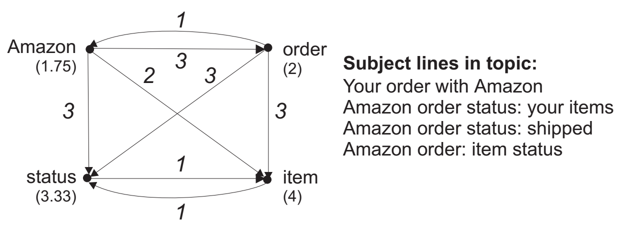


Figure 5.3: Example graph of occurs-after relationships between high-TF-IDF terms. The numbers in parentheses give the average first position. Here, "your," "with," and "shipped" are not nouns and not high-TF-IDF words. Due to stemming, "item" and "items" map to the same term. The resulting label would be "Amazon order status items."

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-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.

5.6 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 [NIST, 2004]. 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).

In addition, we will report results for *Supervised Topic Tracking*. Here, the assumption is that the user reads stories immediately and instantly corrects errors in topic tracking. This is an appropriate model for users who regularly check their inboxes. This task reports errors in the same manner as defined in the topic tracking (TT) task. However, errors made by the clusterer are corrected after each decision by assigning the email to the correct cluster.

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.

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

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

5.7 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 the author.

Why not use or adapt an existing corpus? One consideration was to use the Enron corpus [Klimt and Yang, 2004] 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 [Yeh and Harnly, 2006]. Lastly, the author 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. The corpus 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. Each email received was assigned to exactly one topic. Table 5.1 shows the resulting split.

	Development Set	*Test Set*
Number of emails:	1586	817
Time range:	Oct 1, 2005 to	Feb 1, 2005 to
	Jan 31, 2006	Mar 31, 2006
Languages:	730 English	420 English
	851 German	397 German
	5 other	0 other
Number of topics:	537	269
avg(emails/topic):	2.95	3.03
std(emails/topic):	5.64	4.91
max(emails/topic):	69	47

Table 5.1: 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.

5.7.1 Corpus Statistics

Instead of listing a variety of example data for topics, we have decided to generate cumulative statistics of our corpus to describe its properties. In particular, we contrast our topic-based corpus with a simple thread-based approach. We have also extracted thread groups from our corpus for this purpose.

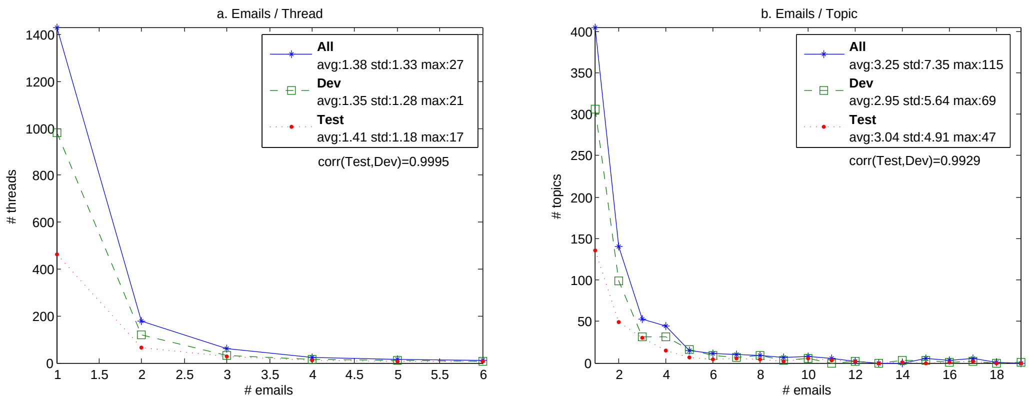


Figure 5.4: Corpus statistics: Sizes of threads and topics.

In our first graph (Figure 5.4), we show that topics in our corpus contain more than twice the number of emails as threads. Our topics generally cover more related emails than threads, which are based on reply-sequences only.

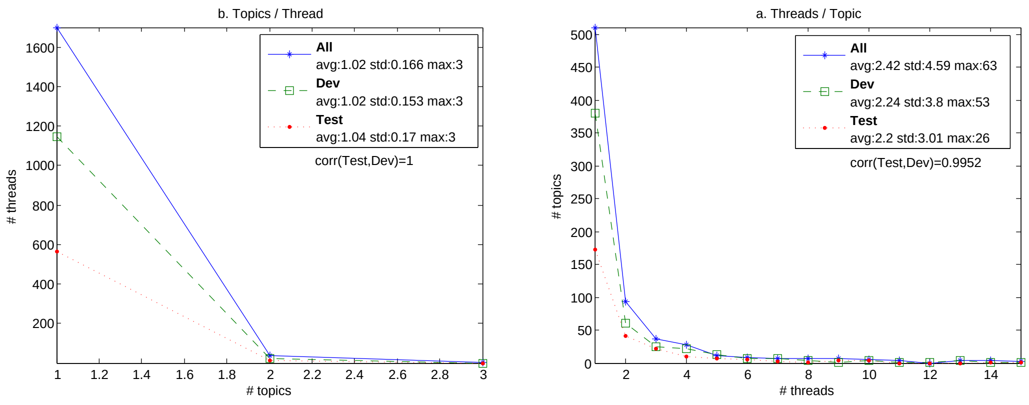


Figure 5.5: Corpus statistics: Topic drift and thread drift.

Next, we examine the occurrence of topic drift and thread drift in our corpus, which we defined in the previous section. Figure 5.5.a shows that topic drift is a rare occurrence in our corpus: Most threads are only to be found in a single topic. As expected, each topic contains a multitude of threads, as seen in Figure 5.5.b.

What are the time intervals that pass between the arrival of two consecutive emails in the same thread or topic? Figure 5.6 answers this question. In threads, one email follows another: There are only few instances in which a thread is picked up again after weeks have passed. Topics more often contain emails that are further apart in time.

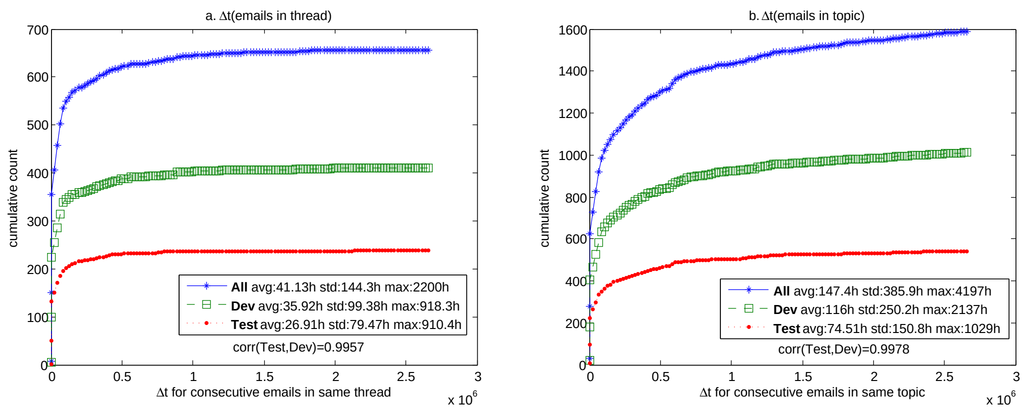


Figure 5.6: Corpus statistics: Email inter-arrival times in threads and topics.

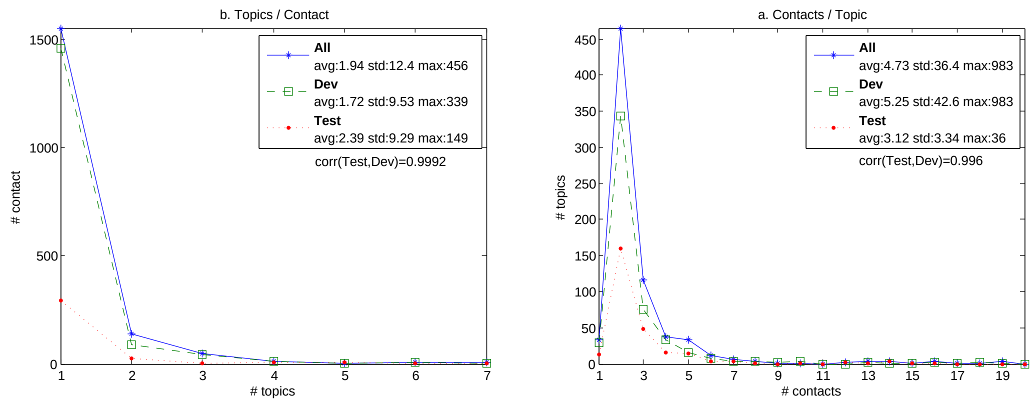


Figure 5.7: Corpus statistics: Contacts vs. topics.

We identified contacts in the corpus by their email address. Each of the author's 1785 contacts in the whole corpus was present in the header fields of 2.81 emails on average (std= 10.6). We examined the distribution of persons over topics and vice versa. Not surprisingly, the distribution of persons per topic is, once again, exponential (Figure 5.7.a): There may be many contacts with whom we discuss only a small range of topics, but some special people with which we exchange email on a wide range of ideas.

Usually, a topic contains just 2 persons conversing with each other (one of them being the author), although there may be many more (Figure 5.7.b). The development set contains an email where a person accidentally used Cc instead of Bcc for a large-distribution message, causing the outlier with 983 persons in a single topic.

These statistics demonstrated how we built our topic-based corpus. The topics we constructed were much more general than threads. While topic drift is rare in our corpus, thread drift occurs often. In addition, topics may be inactive for longer periods of time than threads, which often live for only a few days.

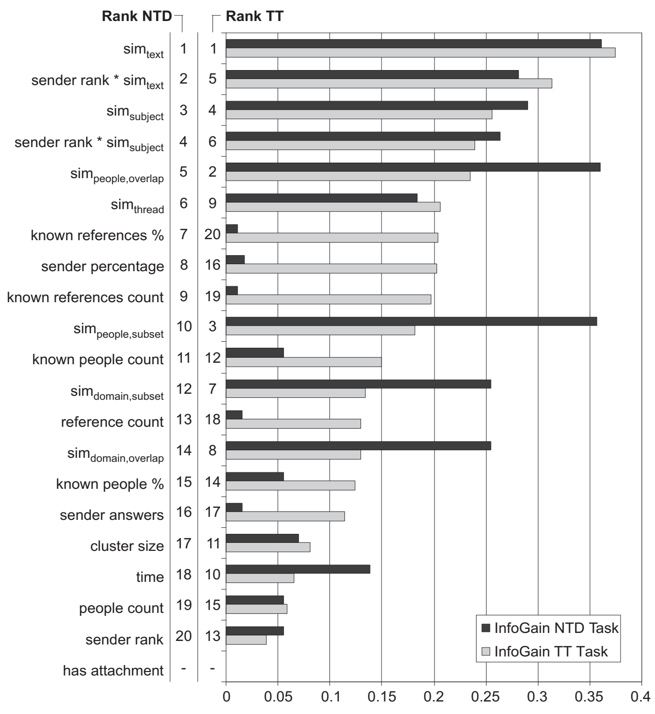


Figure 5.8: Information gain ranking of features from development corpus data: New Topic Detection and Topic Tracking tasks.

5.8 Evaluation Results

This section presents and summarizes results from our evaluation.

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 5.8 shows the information gain of each feature for performing both tasks on the development corpus.

We use Detection Error Tradeoff (DET) curves [NIST, 2004] 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} 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 5.9, and 5.10, and 5.11, as well as Tables 5.2, and 5.3 show results for the test corpus. For reference, we also included results for simpler decision scores derived from text similarity only. Text similarity is the main feature used in traditional TDT.

These results are comparable in performance to that of current work on TDT in news articles [Fiscus and Wheatley, 2004]: In comparison, our TT quality is a bit worse, and NTD performance is 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 data.

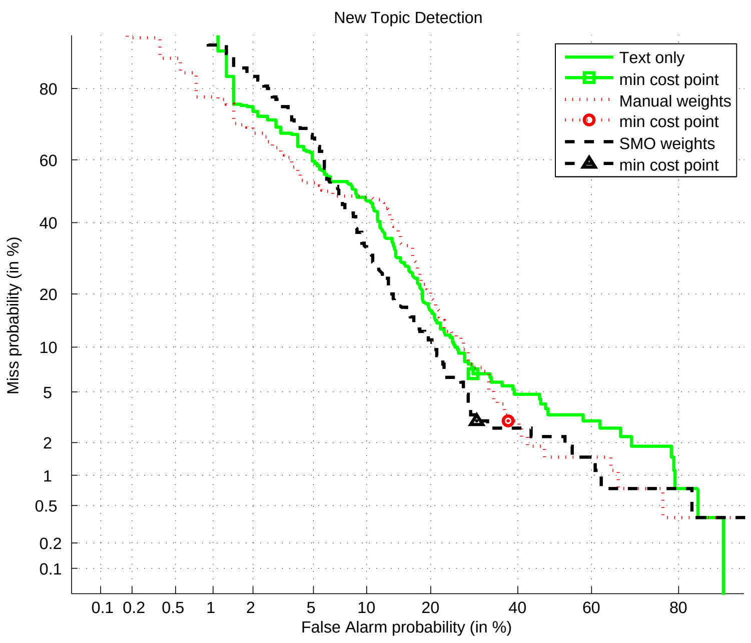


Figure 5.9: NTD task: DET curve with minimum cost points.

The difficulty of this model is the highly subjective definition of topics and dependence on tastes of users. Problems arise for the clustering algorithm when new emails arrive for a topic without sharing 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, the high false alarm seen in Figure 5.9 rates also present a problem in on-line new event detection for news content.

Figures 5.10 and 5.11 contrast supervised and unsupervised tracking. Supervised tracking assumes that errors are corrected instantly: This is typical for users who often check their inbox. Our results show that supervised tracking reduces miss rates: In straggly clusters, where new emails are different from the first ones, unsupervised topic tracking may miss a single email which served as a bridge, and lose the rest of the topic, causing high miss rates. Supervised tracking solves this problem.

We did not evaluate supervised new topic detection: Because of the single-link clustering mechanism we employ, new topic detection does not benefit from correcting errors. In our model, the decision of whether an email represents a new topic or not depends on the max-

Task	text only	manual	SMO
NTD	0.575	0.503	0.425
TT unsupervised	0.396	0.184	0.171
TT supervised	0.224	0.172	0.166

Table 5.2: Minimum detection cost $C_{det,norm}$ for NTD and TT tasks.

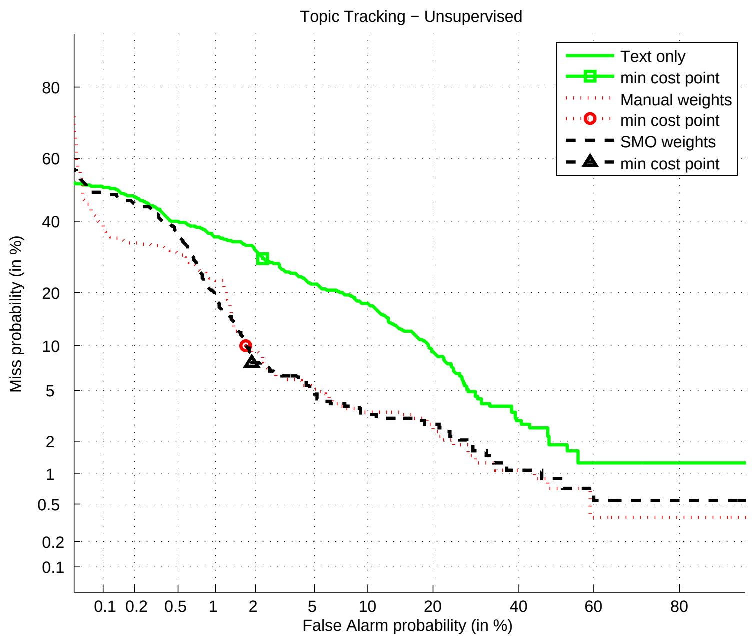


Figure 5.10: Unsupervised TT Task: DET curve with minimum cost points.

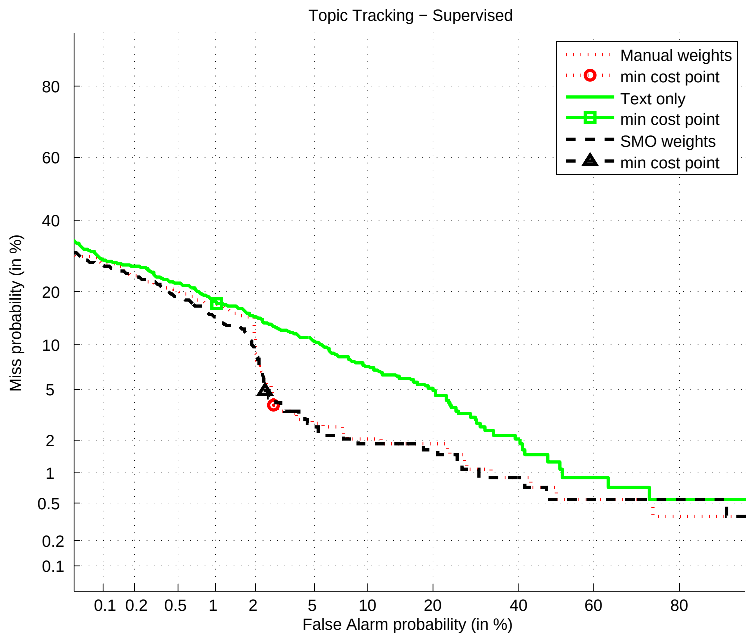


Figure 5.11: Supervised TT Task: DET curve with minimum cost points.

imum similarity to any email currently present in any topic. Cluster assignments, which we correct in supervised tracking, are not considered in this process.

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

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 unsupervised	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
TT supervised	text	0.18	0.01	0.99	0.27	0.82
	manual	0.04	0.03	0.97	0.14	0.96
	SMO	0.05	0.02	0.98	0.16	0.95

Table 5.3: Quality measures in minimum cost point for NTD and TT tasks.

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.

5.9 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. At the moment, our entire corpus contains only data from a single user: We need to find volunteers who contribute additional corpora on which we could test our methods. One potential result could be that the perception of what constitutes a topic may be very dependent on the user, in which case we could devise several clustering profiles which the user can choose from. Another interesting experiment would be interactive reclustering: letting the user retroactively modify the clustering threshold to change the number and size of topics.

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.

5.10 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 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.

6 Implementation

This chapter gives an overview of the software implementations we created as part of this thesis: An package for automatic foldering and BuzzTrack. We begin by discussing the common preprocessing architecture and then review both tools.

6.1 Preprocessing

The automatic foldering evaluations and BuzzTrack share a common preprocessing infrastructure. This component contains the necessary code to separate mailboxes into individual emails, tokenize them into words, identify their language, apply stemming, identify thread structures, and generate frequency counts. Figure 6.1 shows an overview of the preprocessing component.

The MailboxProcessor cuts mbox files into individual emails. A subclass called EnronMailboxProcessor handles the special storage structure in Enron emails. The source text of each email is then passed to the MailTokenizer which splits it into token sequences and analyzes header information. It has many extra parameters that allowed us to experiment with different tokenizing schemes: It can optionally strip HTML, lowercase tokens, regard punctuation as extra tokens, ignore reply lines starting with a ">" character, ignore header lines, or prepend tokens from header lines with the name of the header (explained in Section 4.4). The MailTokenizer uses Python's email.Parser module to parse headers and the multipart structure of emails. In addition, the MailboxProcessor has the ability to extract common phrases from a text, an option which we did not use in our current implementation.

After having tokenized all emails, the MailPostprocessor calls TextCat [Cavnar and Trenkle, 1994] to identify each email's language, and applies a part-of-speech tagger from the NLTK toolkit [Natural Language Toolkit] and the appropriate Porter stemmer. The results are then written into an EMailStore object.

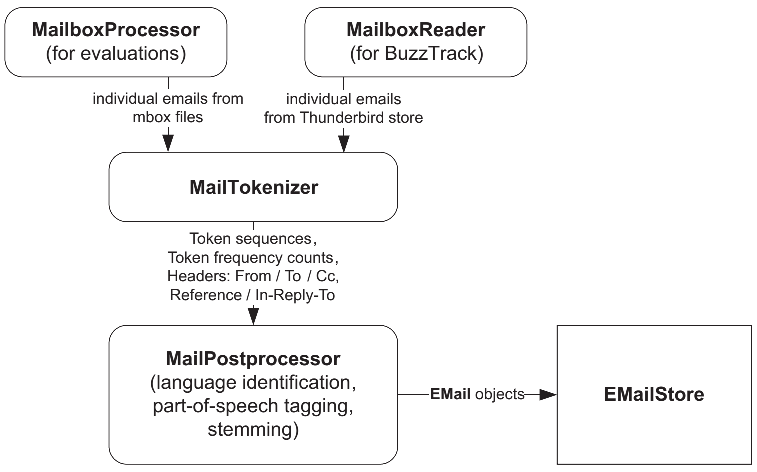

MailboxProcessor
(for evaluations)
MailboxReader
(for BuzzTrack)
individual emails from mbox files
individual emails from Thunderbird store
MailTokenizer
Token sequences, Token frequency counts, Headers: From / To / Cc, Reference / In-Reply-To
MailPostprocessor
(language identification, part-of-speech tagging, stemming)
EMail objects
EMailStore

Figure 6.1: Data flow diagram, preprocessing

The Python code for the preprocessing component can be found in the tokenizer folder of the CD-ROM included with this thesis.

6.1.1 Threading

In Section 5.4.3, we mentioned that the BuzzTrack clusterer uses a thread-based feature for matching together emails. Threading information is derived by the tokenizer from message headers, which typically looks as below:

```
Message-ID: <B8035313ECAA6943ACDE130CDCA3FBD81C16B3@EX2.d.ethz.ch>
In-Reply-To: <2EBB2AB981620D419978BF8928812608165BFA@EX0.d.ethz.ch>
References: <B8035313ECAA6943ACDE130CDCA3FBD81C16AC@EX2.d.ethz.ch>
   <44EF399A.2020005@student.ethz.ch>
   <44F17358.2020805@student.ethz.ch>
   <2EBB2AB981620D419978BF8928812608165BFA@EX0.d.ethz.ch>
Thread-Topic: Lunch 12:00 Uni Mensa?
Thread-Index: AcbJwvLhgkK8TIvnStqMnkaPecB/bQAv+1OaAAWuDQ4=
Subject: RE: Lunch 12:00 Uni Mensa?
```

The Message-ID header is a string which uniquely identifies each email. In-Reply-To contains the message ID of the email to which the current email is a reply; References contains the message IDs of all past emails in the same thread. We use them to identify relationships between messages and group them into threads. We do not use the Thread-Topic and Thread-Index headers, which are also added by some email clients, as they contain redundant information.

6.2 Automatic Foldering

For our experiments with automatic foldering, we take the EMailStore containing output from the MailPostprocessor, and apply a converter which transforms the data into the ARFF

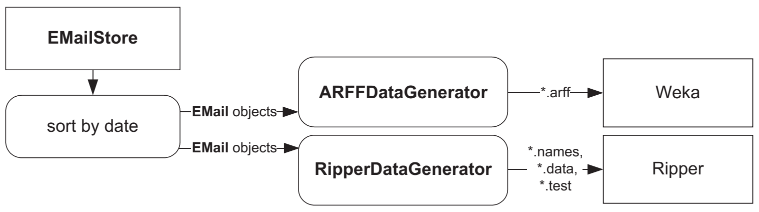


Figure 6.2: Generating input data files for Weka, Ripper classifiers.

format. This data format is required by the Weka Experimenter Framework [Witten and Eibe, 2005], which allowed running our experiments in batched form. We also create input files for the Ripper classifier [Cohen, 1996] in a similar fashion. Figure 6.2 illustrates the process. Both data generators are aware of the training / test split and filter out instances from the test set which belong to classes not present in the training set.

The Python source code for generating these data files can be found in the foldering_exp folder.

6.3 BuzzTrack

This section describes the implementation of BuzzTrack, an email client plugin for the topic detection and tracking scheme we discussed in Chapter 5. BuzzTrack consists of three components:

1. *Clustering:* A core component that takes preprocessed input data and performs new topic detection and topic tracking.
2. *Evaluation:* A framework for evaluating the clustering component according to NIST guidelines.
3. *Plugin:* A plugin for Mozilla Thunderbird that communicates with the clustering component.

All in all, the implementation consists of 3000 lines of JavaScript / XUL code and 14000 lines of Python code. We will now review each of the three components in sequence.

6.3.1 Clustering

The process of topic detection and tracking is handled by the TDT component. Figure 6.3 gives an overview of the clustering process. The first step is updating the TF-IDF data by feeding the stemmed tokens from the EMailStore into the TF-IDF instance. The clustering step can now begin: The Clusterer computes the similarity to each active cluster using ClusterSimilarity instances. The list of active clusters and their emails is obtained through FilteredClusterStore, which is a wrapper around the ClusterStore that supports filtering out emails and clusters which have disappeared from the time window described in Section 5.4.5.

The Clusterer makes its clustering decision based on the similarity value, and records it for later evaluation. The ClusterStore is updated and the cluster created or added to is relabeled by the TopicLabeler. During clustering, all parameters and feature weights are read out of a central instance of the Settings class.

The Python source code for the clustering component can be found in the tdt folder.

6.3.2 Evaluation

For the evaluation, we first need to read the email corpus from disk. The corpus was created manually as a hierarchy of folders in Thunderbird. The CorpusReader class reads the structure of this folder hierarchy, and creates TopicTree nodes for each topic in the tree. The leaves of this tree are individual messages. The training / test data can be obtained from this tree after explicitly setting a split time. Topics which range across the training / test split are automatically divided into two topics. In addition, the tokenized emails are stored into a time-sorted EMailStore. The CorpusStats class is available to generate statistics from the topic corpus.

To carry out the evaluation, we instantiate the TDT component and sequentially feed emails from the EMailStore to it. Its output is then compared to the topics in the TopicTree. Two scripts (run_tdt_ntd_eval and run_tdt_tt_eval) each carry out one type of TDT evaluations (new topic detection or topic tracking) according to the NIST criteria. The run_tdt_eval script is available for batch experiments. The output of these programs is then fed into Matlab to generate the DET Tradeoff Curve plots, described in Section 5.8.

The Python source code for the TDT evaluation can be found in the tdt/eval and tdt/run folders.

6.3.3 Plugin

For implementing the BuzzTrack plugin, we initially considered to integrate it into the Microsoft Outlook or Mozilla Thunderbird email clients. We decided to target Thunderbird for two reasons: First, it is relatively easy to write extensions for this product. Second, while Outlook is used more in corporate environments, Thunderbird is typically the email client of choice for early adopters: Users who are most likely to try out new add-ons.

We considered three options of integrating the plugin with Thunderbird:

1. Building a C++ component in XPCOM [Turner and Oeschger, 2003].
2. Building a Python XPCOM component with PyXPCOM [Mozilla, 2006a].
3. Launching a Python server from within Thunderbird, and communicating via a local socket.

The first option would have brought faster application performance, but involved a large investment of development time, as the tokenizer and clusterer would have needed to be reimplemented in a different language. As the second option involved Python, no rewrite of

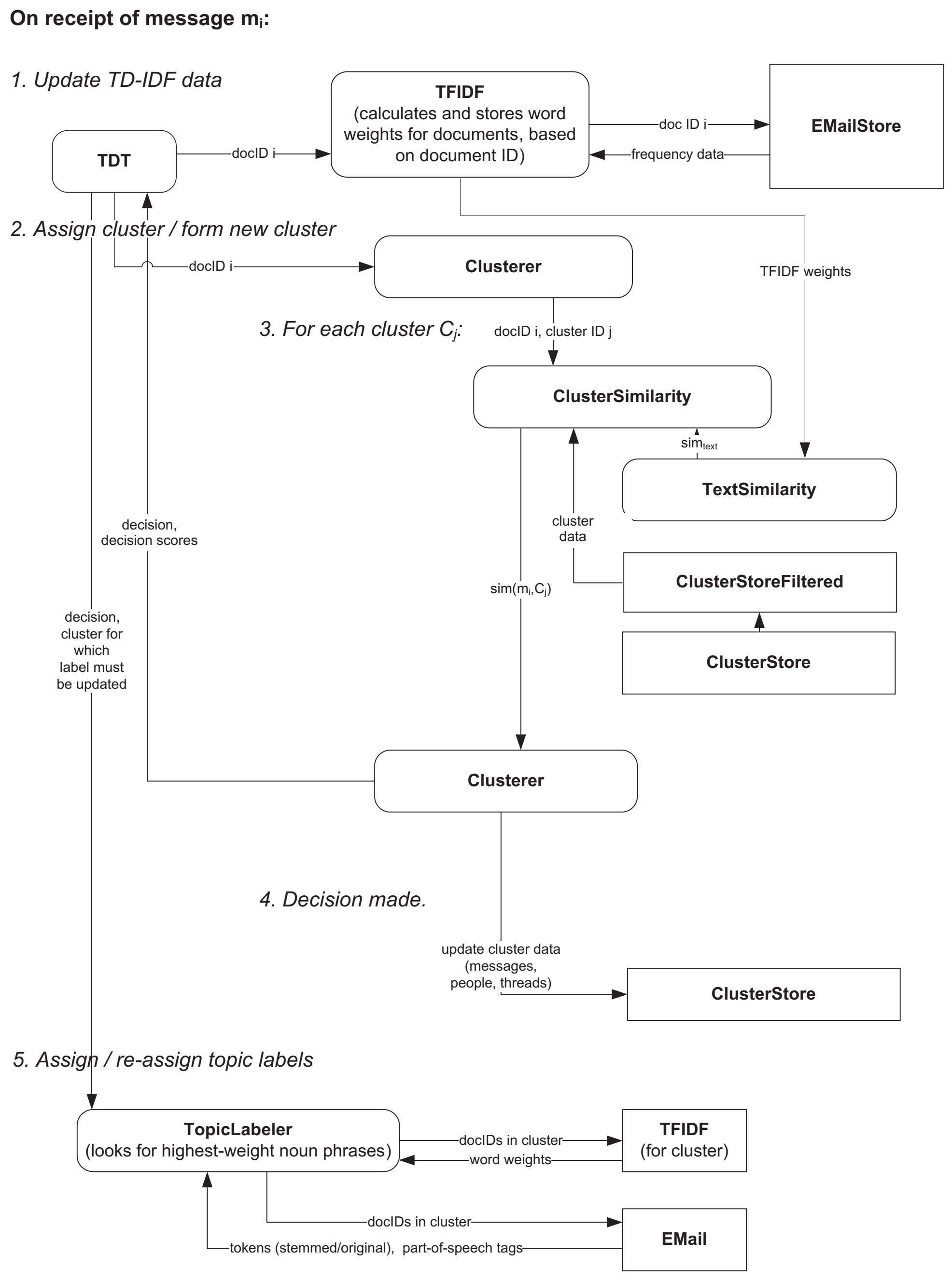

On receipt of message m_i:
1. Update TD-IDF data
TFIDF (calculates and stores word weights for documents, based on document ID)
TDT
docID i
doc ID i
frequency data
EMailStore
2. Assign cluster / form new cluster
docID i
Clusterer
TFIDF weights
3. For each cluster C_j:
docID i, cluster ID j
ClusterSimilarity
sim_text
TextSimilarity
cluster data
decision, decision scores
ClusterStoreFiltered
sim(m_i,C_j)
decision, cluster for which label must be updated
ClusterStore
Clusterer
4. Decision made.
update cluster data (messages, people, threads)
ClusterStore
5. Assign / re-assign topic labels
TopicLabeler (looks for highest-weight noun phrases)
docIDs in cluster
word weights
TFIDF (for cluster)
docIDs in cluster
tokens (stemmed/original), part-of-speech tags
EMail

Figure 6.3: Data flow diagram, clustering.

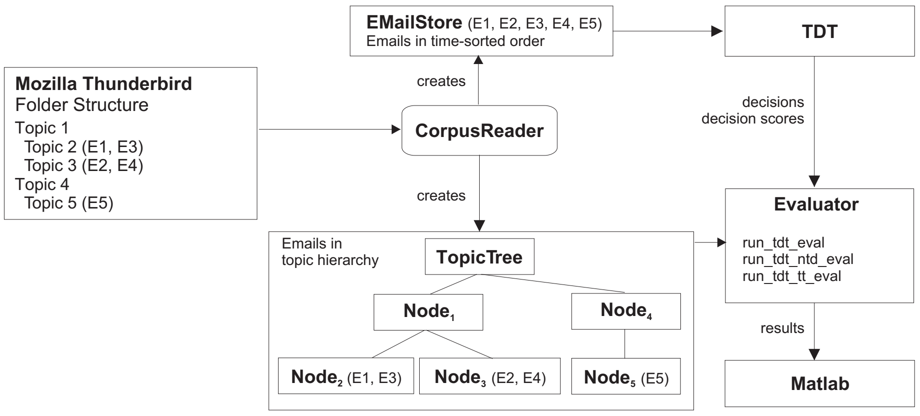


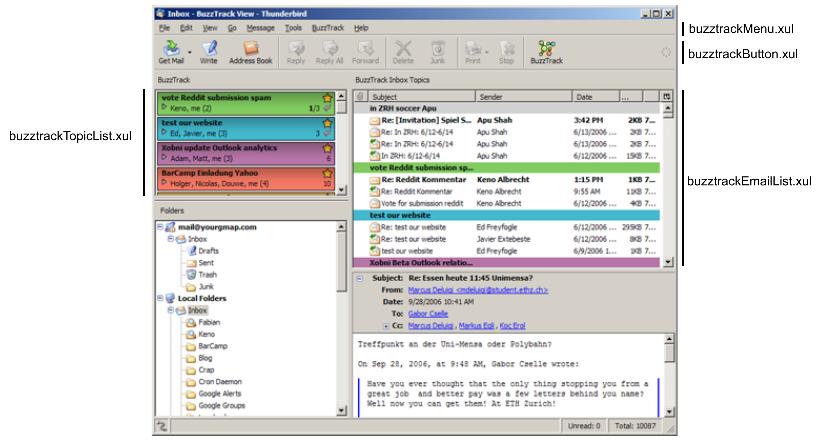
Figure 6.4: TDT evaluation process

our clustering algorithm would have been needed. Still, the drawback was that PyXPCOM is not included in the current versions of Mozilla releases. Instead, the email client needs to be recompiled with special options, making the software undeployable to end users.

We decided the implement a server in Python and interface via a socket (option 3), because it allowed for easy integration, and required little extra implementation effort. This approach is based on the Spamato Thunderbird plugin [Spamato, 2006].

6.3.4 Code Structure

The BuzzTrack plugin is implemented as an extension for Mozilla Thunderbird and consists of several JavaScript source files and XUL overlays. The general techniques of constructing extensions for Mozilla products are described in [XUL Planet, 2006] or [Bullard et al., 2001].



vote Reddit submission spam	Keno, me (2) 1/3
test our website	Ed, Javier, me (3) 3
Xobni update Outlook analytics	Adam, Matt, me (3) 6
BarCamp Einladung Yahoo	Holger, Nicolas, Douwe, me (4) 10

Subject	Sender	Date	...
in ZRH soccer Apu			
Re: [Invitation] Spiel S...	Apu Shah	3:42 PM	2KB 7...
Re: In ZRH: 6/12-6/14	Apu Shah	6/13/2006 ...	2KB 7...
Re: In ZRH: 6/12-6/14	Apu Shah	6/13/2006 ...	2KB 7...
In ZRH: 6/12-6/14	Apu Shah	6/12/2006 ...	15KB 7...
vote Reddit submission sp...			
Re: Reddit Kommentar	Keno Albrecht	1:15 PM	1KB 7...
Re: Reddit Kommentar	Keno Albrecht	9:55 AM	11KB 7...
Vote for submission reddit	Keno Albrecht	6/12/2006 ...	4KB 7...
test our website			
Re: test our website	Ed Freyfogle	6/12/2006 ...	295KB 7...
Re: test our website	Javier Extebeste	6/12/2006 ...	8KB 7...
test our website	Ed Freyfogle	6/9/2006 1...	1KB 7...
Xobni Beta Outlook relatio...			

Subject: Re: Essen heute 11:45 Unimensa?
From: Marcus Deluigi <mdeluigi@student.ethz.ch>
Date: 9/28/2006 10:41 AM
To: Gabor Cselle
Cc: Marcus Deluigi, Markus Egli, Koc Erol

```
Treffpunkt an der Uni-Mensa oder Polybahn?

On Sep 28, 2006, at 9:48 AM, Gabor Cselle wrote:

 Have you ever thought that the only thing stopping you from a
 great job  and better pay was a few letters behind you name?
 Well now you can get them! At ETH Zurich!
```

Figure 6.5: XUL overlays in BuzzTrack extension.

The GUI is contained in four separate XUL files, as shown in Figure 6.5. There exist three files with JavaScript source code:

- buzztrack.js is the main controller for the application. It contains code for initialization on startup, and mediates between the view and communication components. It registers for notifications on incoming email messages and handles them. In addition, it contains the data structures in which the client-side clustering data resides.
- buzztrackComm.js contains code for communicating with the Python server.
- buzztrackView.js manages the user interface.

We will now focus on the integration with the Python clusterer.

6.3.5 Communication with Clustering Component

At startup, the BuzzTrack extension launches the BuzzTrackServer, a Python process which acts as a server for incoming clustering requests. The five main tasks of this server are:

- Exchanging messages with the client.
- Processing newly incoming email.
- Responding to user changes, e.g. merging two topics or moving emails between topics.
- Initializing contents.
- Ensuring persistence.

Figure 6.6 provides an overview of the components involved in communicating between the clustering component and the Mozilla Thunderbird plugin.

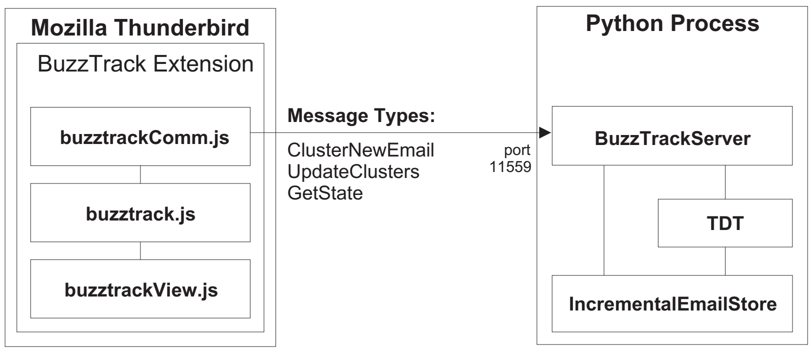


Figure 6.6: Communication with clustering component.

Exchanging Messages with the Client

Messages sent between the BuzzTrackServer and the client plugin are in a proprietary text-based format. One happy coincidence is that Python's and JavaScript's string representations of arrays and dictionaries are very similar. Instead of having to serialize data in Python and parse it in JavaScript, we can simply use JavaScript's `eval()` function to recreate the data structure on the client side.

Processing Newly Incoming Email

Every time the client downloads a new email, we notify the BuzzTrackServer via a ClusterNewEmail message that it should fetch its source text from the local mbox file in which the email was stored. We include the position of the email inside the mbox file in the ClusterNewEmail message. The BuzzTrackServer then reads the email from disk and passes it to the MailTokenizer, which we described in Section 6.1. After the email is tokenized, it is added to the EMailStore. The clustering component, described in Section 6.3.1, is now called and returns a decision on whether this email constitutes a new topic, or alternatively, which topic it should be assigned to. This decision is communicated to the client in the response to ClusterNewEmail.

Responding to User Changes

The user can modify the topic structure and topic contents in the user interface, either by merging topics via the context menu or by drag-and-dropping items in the topic content list. While the plugin maintains its own memory of cluster contents, it needs to make sure that these are reflected in the clustering component, so that future clustering decisions can be carried out correctly. For this, the client sends an UpdateCluster message to inform the server of this change.

Ensuring Persistence

BuzzTrack needs to remember tokenization and clustering results even across several sessions. For this, it serializes its ClusterStore object to disk every time an email is added to a topic or when manually moves emails across topics.

In addition, the disk representation of the IncrementalEMailStore is updated every time a new email is tokenized. This class is a subclass of EMailStore which allows for incremental serialization: Data for newly incoming emails is appended to the file, instead of re-serializing the entire EMailStore object.

Initializing Contents

At startup, the client must first create its own view of the current contents of each topic. It must first wait for the server to start up. During this process, our plugin displays a “Loading topics ...” message to the user. It sends a GetState message to the BuzzTrackServer. In response, it receives the current contents and structure of the ClusterStore.

6.4 Installation

To install the BuzzTrack plugin in Mozilla Thunderbird, one must first install several prerequisites:

- ActivePython, version 2.4 or higher.
- NumArray library for Python, version 1.5.1 or higher.
- NTLK-Lite Toolkit, version 0.6.4 or higher

These items are needed for several of the natural language preprocessing functions of BuzzTrack, such as part-of-speech tagging and stemming. In the base version, the BuzzTrack plugin currently assumes that all emails are written in English. Optionally, one can also install TextCat [Cavnar and Trenkle, 1994] for language identification and language-specific stemming. However, as TextCat is written in Perl, this also requires installing ActivePerl or another Perl installation. To date, BuzzTrack has only been tested on Windows systems.

6.5 Future Directions

This chapter reviewed the implementation of components we created for our automatic foldering evaluations. In addition, we described the components of BuzzTrack, which consists of an email tokenizer, a clustering component, a plugin for Mozilla Thunderbird, and an evaluator of clustering performance.

There exist several potential directions for future work: The application could be reimplemented in C++ to improve performance and ease of installation. However, it would be even more exciting to explore new functionality, as described in Section 5.9: Introducing different clustering methods per user type, retroactive reclustering, and interface changes such as a visual map of emails related to the one currently selected.

7 Conclusion

This thesis deals with organizing email from the view of the end user. We reviewed the problem of email overload, existing work on the subject, and proposed a novel solution.

We investigated automatic foldering, which is one of the most popular approaches. Our findings showed that accuracy rates are very low and that there exist conceptual problems that cannot be overcome.

To resolve these problems, we devised a method for topic detection and tracking (TDT) of email. TDT is a series of competitions organized by the National Institute of Standards and Technology (NIST), and there is a great body of work on applying these techniques to news messages. We are not aware of work that applied TDT methods and evaluation standards to email. We found promising results that compete well with existing work on news articles. We built a tool, BuzzTrack, to make this technology accessible in Mozilla Thunderbird, a popular email client.

We reviewed the implementation details of our automatic foldering tools and BuzzTrack in a separate chapter. The central processing units from both programs were implemented in Python, while the presentation logic was written in JavaScript / XUL.

We believe that BuzzTrack will be a valuable tool for users in the real world.

Bibliography

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

James Allan, Stephen Harding, David Fisher, Alvaro Bolivar, Sergio Guzman-Lara, and Peter Amstutz. Taking topic detection from evaluation to practice. In *HICSS '05: Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Track 4*, page 101.1, 2005.

Apple. Apple Mail. `http://www.apple.com/macosx/features/mail/`, 2006.

Olle Bälter. Keystroke level analysis of email message organization. In *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 105–112, The Hague, Netherlands, 2000.

Olle Bälter and Candace Sidner. Bifrost inbox organizer: giving users control over the inbox. In *NordiCHI '02: Proceedings of the second Nordic Conference on Human-computer interaction*, pages 111–118, Aarhus, Denmark, 2002.

Fabian Bannwart and Peter Müller. Changing programs correctly: Refactoring with specifications. In *FM 2006: 14th International Symposium on Formal Methods*, pages 492–507, Hamilton, Canada, 2006.

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

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 *CHI '03: Proceedings of the 2003 Conference on Human Factors in Computing Systems*, pages 345–352, Ft. Lauderdale, FL, USA, 2003.

Gary Boone. Concept features in Re:Agent, an intelligent email agent. In *AGENTS '98: Proceedings of the Second International Conference on Autonomous Agents*, pages 141–148, Minneapolis, MN, USA, 1998.

Danah Boyd. Faceted id/entity: Managing representation in a digital world. Master's thesis, Massachusetts Institute of Technology, September 2002.

Jake D. Brutlag and Christopher Meek. Challenges of the email domain for text classification. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 103–110, 2000.

Vaughn Bullard, Kevin T. Smith, and Michael C. Daconta. *Essential XUL Programming*. Wiley, August 2001.

Caelo. Nelson Email Organizer (NEO). `http://www.caelo.com/`, 2006.

Vitor R. Carvalho and William W. Cohen. Learning to extract signature and reply lines from email. *CEAS '04*, 2004.

William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

William Cohen. Learning rules that classify e-mail. In *Papers from the AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25, 1996.

Simon Corston-Oliver, Eric K. Ringger, Michael Gamon, and Richard Campbell. Integration of email and task lists. In *CEAS 2004 - First Conference on Email and Anti-Spam*, July 2004.

Elisabeth Crawford, Judy Kay, and Eric McCreath. Automatic induction of rules for e-mail classification. In *Sixth Australasian Document Computing Symposium*, 2001.

Delicious. Delicious. `http://del.icio.us/`, October 2003.

Peter J. Denning. ACM President's Letter: Electronic Junk. *Communications of the ACM*, 25 (3):163–165, 1982.

Judith Donath. Rhythm of salience: A conversation map. *Draft*, 2006.

Mark Dredze, John Blitzer, and Fernando Pereira. Reply expectation prediction for email management. In *CEAS*, 2005.

Mark Dredze, John Blitzer, and Fernando Pereira. "sorry i forgot the attachment": Email attachment prediction. In *Third Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, July 2006a.

Mark Dredze, Tessa Lau, and Nicholas Kushmerick. Automatically classifying emails into activities. In *IUI '06: Proceedings of the 11th international Conference on Intelligent user interfaces*, pages 70–77, Sydney, Australia, 2006b.

Nicolas Ducheneaut and Victoria Bellotti. E-mail as habitat: an exploration of embedded personal information management. *interactions*, 8(5):30–38, 2001.

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

Danyel Aharon Fisher. *Social and Temporal Structures in Everyday Communication*. PhD thesis, University of Irvine, California, 2004.

Enrico Giacoletto and Karl Aberer. Automatic expansion of manual email classifications based on text analysis. In *Proceedings of ODBASE' 03, the 2nd International Conference on Ontologies, Databases, and Applications of Semantics*, pages 785–802, 2003.

Google. Gmail. `http://www.gmail.com/`, April 2004.

Google. Google Desktop. `http://desktop.google.com/`, 2006.

Paul Graham. Better bayesian filtering. `http://www.paulgraham.com/better.html`, January 2003.

Paul Graham. A plan for spam. `http://www.paulgraham.com/spam.html`, August 2002.

Daniel Gruen, Steven L. Rohall, Suzanne Minassian, Bernard Kerr, Paul Moody, Bob Stachel, Martin Wattenberg, and Eric Wilcox. Lessons from the Remail prototypes. In *CSCW '04: Proceedings of the 2004 ACM Conference on Computer supported cooperative work*, pages 152–161, Chicago, Illinois, USA, 2004.

Jacek Gwizdka. Email task management styles: the cleaners and the keepers. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, pages 1235–1238, Vienna, Austria, 2004.

Jeffrey Heer. Exploring Enron. `http://jheer.org/enron/`, 2005.

Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, Berkeley, California, United States, 1999.

Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive alerting. In *Proceedings of UAI '99, Conference on Uncertainty and Artificial Intelligence*, pages 305–313. Morgan Kaufmann, Stockholm, Sweden, 1999.

IBM. Lotus Notes. `http://www.ibm.com/software/lotus/`, 2006.

Yoram M. Kalman and Sheizaf Rafaeli. Email chronemics: Unobtrusive profiling of response times. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 4*, page 108.2, 2005.

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Email mining: Emerging techniques for email management. *Web Data Management Practices: Emerging Techniques and Technologies*, 2006.

Bernard Kerr. Thread arcs: An email thread visualization. In *IEEE Symposium on Information Visualization (INFOVIZ) '07*, 2003.

Bernard Kerr and Eric Wilkox. Designing Remail: Reinventing the email client through innovation and integration. Technical report, IBM Research, 2004.

Jon Kleinberg. Bursty and hierarchical structure in streams. In *KDD '02: Proceedings of the eighth ACM SIGKDD international Conference on Knowledge discovery and data mining*, pages 91–101, Edmonton, Alberta, Canada, 2002.

Bryan Klimt and Yiming Yang. Introducing the Enron Corpus. In *CEAS*, 2004.

Nicholas Kushmerick and Tessa Lau. Automated email activity management: an unsupervised learning approach. In *IUI '05: Proceedings of the 10th international Conference on Intelligent user interfaces*, pages 67–74, San Diego, California, USA, 2005.

Nicholas Kushmerick, Tessa Lau, Mark Dredze, and Rinat Khoussainov. Activity-centric email: A machine learning approach. In *Proceedings of The Twenty-First National Conference on Artificial Intelligence (AAAI '06)*, Boston, MA, USA, 2006.

Derek Lam, Steven L. Rohall, Chris Schmandt, and Mia K. Stern. Exploiting e-mail structure to improve summarization. In *ACM 2002 Conference on Computer Supported Cooperative Work (CSCW2002)*, New Orleans, LA, 2002.

Wendy E. Mackay. More than just a communication system: diversity in the use of electronic mail. In *CSCW '88: Proceedings of the 1988 ACM Conference on Computer-supported cooperative work*, pages 344–353, Portland, Oregon, United States, 1988.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. ISBN 0262133601.

Andrew McCallum and Karl Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, 1998.

Microsoft. MSN Hotmail. `http://www.hotmail.com/`, 2006a.

Microsoft. Microsoft Outlook. `http://www.microsoft.com/outlook`, 2006b.

Microsoft. Windows Desktop Search. `http://www.microsoft.com/windows/desktopsearch/`, 2006c.

Mozilla. PyXPCOM. `http://developer.mozilla.org/en/docs/PyXPCOM`, 2006a.

Mozilla. Mozilla Thunderbird. `http://www.mozilla.org/thunderbird`, 2006b.

Michael J. Muller, Werner Geyer, Beth Brownholtz, Eric Wilcox, and David R. Millen. One-hundred days in an activity-centric collaboration environment based on shared objects. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 375–382, Vienna, Austria, 2004.

Natural Language Toolkit. `http://nltk.sourceforge.net/`, 2006.

Carman Neustaedter, A. J. Bernheim Brush, and Marc A. Smith. Beyond "from" and "received": exploring the dynamics of email triage. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, pages 1977–1980, Portland, OR, USA, 2005a.

Carman Neustaedter, A. J. Bernheim Brush, Marc A. Smith, and Danyel Fisher. The social network and relationship finder: Social sorting for email triage. In *CEAS 2005 - Second Conference on Email and Anti-Spam*, Stanford University, CA, USA, July 2005b.

Paula S. Newman and John C. Blitzer. Summarizing archived discussions: a beginning. In *IUI '03: Proceedings of the 8th International Conference on Intelligent user interfaces*, pages 273–276, Miami, Florida, USA, 2003.

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

Adam Perer and Marc A. Smith. Contrasting portraits of email practices: Visual approaches to reflection and analysis. In *Advanced Visual Interfaces 2006*. ACM Press, Venice, Italy, 2006.

Adam Perer, Ben Shneiderman, and Douglas W. Oard. Using rhythms of relationships to understand email archives. In *Journal of the American Society for Information Science and Technology (JASIST)*, 2006.

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

POPFile. `http://popfile.sourceforge.net/`, 2006.

Radicati Group. Market Update 2006 Q1. `http://www.radicati.com/`, 2006a.

Radicati Group. Microsoft Exchange market share statistics. `http://www.radicati.com/`, 2006b.

Jason D. M. Rennie. ifile: An application of machine learning to mail filtering. In *Proceedings of the KDD-2000 Workshop on Text Mining*, 2000.

Steven L. Rohall, Dan Gruen, Paul Moody, Martin Wattenberg, Mia Stern, Bernard Kerr, Bob Stachel, Kushal Dave, Robert Armes, and Eric Wilcox. Remail: a reinvented email prototype. In *CHI '04: Extended Abstracts on Human Factors in Computing Systems*, pages 791–792, Vienna, Austria, 2004.

Warren Sack. Conversation map: An interface for very-large-scale conversations. *Journal of Management Information Systems*, 2000.

Sam Scott and Stan Matwin. Feature engineering for text classification. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 379–388, 1999.

Richard Segal and Jeffrey O. Kephart. Mailcat: An intelligent assistant for organizing e-mail. In *AAAI/IAAI*, pages 925–926, 1999.

Ben Shneiderman. Tree visualization with tree-maps: 2-D space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, 1992.

Marc A. Smith and Andrew T. Fiore. Visualization components for persistent conversations. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 136–143, Seattle, Washington, United States, 2001.

Spamato. Spamato website. `http://www.spamato.net/`, 2006.

Sandra Sudarsky and Rune Hjelsvold. Visualizing electronic mail. *iv*, 00:3, 2002.

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

Doug Turner and Ian Oeschger. *Creating XPCOM Components*. Mozilla Foundation, 2003.

Andrzej Turski, Debbie Warnack, Lili Cheng, Shelly Farnham, and Susan Yee. Inner Circle: People centered email client. In *CHI '05 Extended Abstracts on Human factors in Computing Systems*, pages 1845–1848, Portland, OR, USA, 2005.

Gina Danielle Venolia and Carman Neustaedter. Understanding sequence and reply relationships within email conversations: a mixed-model visualization. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 361–368, Ft. Lauderdale, Florida, USA, 2003.

Gina Danielle Venolia, Laura Dabbish, JJ Cadiz, and Anoop Gupta. Supporting email workflow. Technical report, Microsoft Research - Collaboration and Multimedia Group, September 2001.

Fernanda B. Viégas, Danah Boyd, David H. Nguyen, Jeffrey Potter, and Judith Donath. Digital artifacts for remembering and storytelling: PostHistory and Social Network Fragments. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences - Track 4*, page 40109.1, 2004.

Fernanda B. Viégas, Scott Golder, and Judith Donath. Visualizing email content: portraying relationships from conversational histories. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 979–988, Montreal, Quebec, Canada, 2006.

Martin Wattenberg and David Millen. Conversation thumbnails for large-scale discussions. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, pages 742–743, Ft. Lauderdale, Florida, USA, 2003.

Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *CHI '96: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 276–283, Vancouver, British Columbia, Canada, 1996.

Steve Whittaker, Quentin Jones, Bonnie Nardi, Mike Creech, Loren Terveen, Ellen Isaacs, and John Hainsworth. ContactMap: Organizing communication in a social desktop. *ACM Transactions on Computer-Human Interaction*, 11(4):445–471, 2004.

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

Ian H. Witten and Frank Eibe. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann, 2005. ISBN 0-12-088407-0.

Xobni. Xobni Analytics Beta. `http://www.xobni.com/`, 2006.

XUL Planet. XUL planet main tutorial. `http://www.xulplanet.com/tutorials/xultu/`, 2006.

Yahoo. Yahoo Mail Beta. `http://mail.yahoo.com/`, 2006.

Jen-Yuan Yeh and Aaron Harnly. Email thread reassembly using similarity matching. In *CEAS '06*, Mountain View, CA, USA, 2006.

List of Figures

List of Tables

Index