INTELLIGENT EMAIL: AIDING USERS WITH AI

Mark Harel Dredze

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial
Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2009

Fernando Pereira
Supervisor of Dissertation

Rajeev Alur
Graduate Group Chairperson

This dissertation is dedicated to the memory of my mother

Helen Marlene Berkovitz Dredze, Ph.D.,

who was the first Ph.D. I knew.

Acknowledgements

This thesis is the product of both hard work and invaluable assistance from colleagues and friends. In my time at Penn, I have had the privilege of working with some of the finest faculty and students.

First and foremost, I thank my adviser Professor Fernando Pereira, who has expertly guided my research and career. Much can be said about his amazing breadth and depth of knowledge in so many fields. Whenever I mention his name to colleagues in the field, I always hear praise of his kindness and character. This has made him a wonderful adviser and mentor.

Next, I thank my committee, who have shaped both my thesis and my career in the field. Professor Mitch Marcus and I have shared many long and fascinating conversations, many that had nothing to do with computer science. I will miss our frequent discussions and look forward to having more in the future. Professor Ani Nenkova read my thesis with a careful eye and provided invaluable critical feedback. Our conversations gave me confidence and direction in my research. Professor Ben Taskar constantly amazes me with his depth of knowledge and rapid understanding of difficult material. I often learned something new about my own work in our discussions. I am very pleased that Professor Lise Getoor was able to join my committee and am impressed that our relatively few interactions had such a strong impact on this thesis. I look forward to continuing these conversations and to the new work they will produce.

There are many others at Penn who contributed to my success. I have been deeply privileged to have worked with Dr. Koby Crammer. Koby has been a second adviser to me and has taught me so much about how to conduct high quality research. One of my biggest regrets in leaving Penn is that Koby has so much more to teach and I have so much more to learn. I know that his graduate students will be truly lucky to have such a wonderful adviser.

I have worked closely with many of my colleagues in Fernando's research group. John Blitzer was my model for a hard working and successful graduate student. Our close collaboration helped me grow into a mature graduate student. I already miss his two-view hoodie since he graduated last year. Thank to Ryan McDonald for constantly willing to talk about and help with interesting problems; Partha Pratim Talukdar for many excited collaborations, even the ones that didn't work; Alex Kulesza for many fascinating conversations that were a welcome distraction; Kuzman Ganchev for his careful explanations and witty humor; Qian Liu and Axel Bernal, who both showed patience as they explained basic biology concepts each time we spoke so that I could better understand their work. The NLP and machine learning students at Penn make it such a great graduate experience: Ryan Gabbard, Emily Pitler, Qiuye (Sophie) Zhao, Annie Louis, Nikhil Dinesh, Liang Huang, Jenn Wortman, Ted Sandler, Kilian Weinberger, Bill Kandylas, Jeff Vaughn (honorary machine learning/NLP). I also enjoyed working with several excellent visiting students: Kedar Bellare, who *really* taught me CRFs, Joao Graca, who introduced me to the other Fernando Pereira, and Hanna Wallach, who spent hours talking with me about machine learning, user interfaces and women in CS. Thank you to Joel Wallenberg, a linguist who constantly reassured me that I did know something about linguistics and rarely laughed when I didn't. To all of those here and to those I haven't mentioned: you made my time at Penn a wonderful experience.

In addition to these colleagues, I thank the wonderful CIS staff who were a constant support to both myself and other students. They ensured that things always ran smoothly and were there for the times when things did not. I thank Mike Felker for helping me from my first visit to Penn until I handed in this thesis.

Thank you to the talented and numerous undergraduate students who I have worked with on a wide variety of fascinating projects. My work with undergraduate research has been a rewarding experience throughout my graduate career.

In addition to my colleagues at Penn, I have been privileged to work with many others in the field. Tessa Lau advised me during my internship at IBM before I began my graduate career and has continued to mentor me throughout my graduate career. She has been my constant mentor and I look forward to continuing our rewarding conversations. Rie Ando and Tong Zhang both taught me much during my time at IBM; I am privileged to know such talented and respected researchers. I am privileged to have spent two summers working with world-class researchers at Google. Krzysztof Czuba, Peter Norvig and Bill Schilit are among some of the finest researchers and peers in the field. I will always recall fondly my time with these and other colleagues at Google, IBM, and Microsoft.

I would never have even thought of pursuing a PhD had it not been for my close relationships with Professors Larry Birnbaum and Kris Hammond, who advised me during my undergraduate career at Northwestern University. I have many fond and hilarious stories of my time with Larry and Kris.

While these colleagues have ensured my academic success, so many friends outside of school have made my years at Penn some of the best of my life. It would be impossible to name all of my friends these past few years. The 4105 community has been my home. As I join its members who have left and dispersed across the globe, I know that these friends will remain with me throughout my life.

Finally, I would like to thank my family for constantly supporting and caring for me. Their encouragement and love has made me who I am today.

Acharon acharon chaviv, I thank my dear wife Chava Evans, who has been a constant source of support and encouragement. Without her love I would never have been able to achieve all that I have.

ABSTRACT

INTELLIGENT EMAIL: AIDING USERS WITH AI

Mark Harel Dredze

Fernando Pereira

User productivity and attention suffer from email overload. The human computer interaction community has designed new types of interfaces to facilitate email management, including email triage, activity management, search and organization. In this work we draw ideas from machine learning and natural language processing to introduce intelligent email and define it as intelligent systems for supporting email interfaces. Our interfaces are information driven, enabling users to make faster, smarter and less error prone decisions in processing email.

We develop intelligent email in several stages. First, we examine the common problem of the forgotten attachment by building an attachment prediction system, which can support different user interfaces for this problem. Next, we explore the task of email triage, the process of managing large amounts of email. We propose a reply management system supported by a reply predictor, automatically labeling messages that need a reply. To enable cross-user learning, we develop a shared set of deictic features with user specific extraction based on social network analysis of email. We then explore new representations for message content based on latent concept models. Next, we develop a system for email activity classification to support email activity management interfaces. Finally, we extend the popular tool of faceted browsing to email by developing automatic facet rankers to select the most useful facets for display to the user. A large scale evaluation and user survey demonstrates the effectiveness of intelligent email applications in real world settings.

We also consider new learning methods useful for intelligent email: Confidence-Weighted (CW) learning. CW learning is a family of online learning algorithms where online updates are confidence sensitive, favoring larger updates to rarer features. Incorporating language sensitivities improves performance on a number of NLP applications, including important learning settings for intelligent email. We consider how to scale learning in the email setting to very large data environments through parallel training. To reduce the cost labeling email by users, we consider active learning. We show that CW learning improves standard margin-based active learning. Finally, we show how confidence sensitive parameter combinations can be used to perform cross-user and multi-domain learning.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

It has been over ten years since Whittaker and Sidner [251] identified a major shift in the role of email. Their 1996 paper observed that email had evolved from a communications system into a tool for common workplace activities, like task management and personal archiving. They coined this change "email overload;" email was overloaded by other functions.

Over the past decade the term overload has taken on a dual meaning: the overloaded email tool and the overloaded email user. As this single tool became the center of project management, workplace communications, notifications and task lists, the sheer volume of mail increased beyond manageable limits. The primary use cases identified by Whittaker and Sidner remain unchanged: personal archiving and organization, communication and task management. Yet email clients have not kept up by introducing new management features. The dominant paradigm for mail clients remains as a communications tool despite email's many other uses. The lack of tools for managing thousands of messages strains users. Even with new interface features, the lack of information for making email decisions stresses users.

This thesis addresses email overload through artificial intelligence. Our intelligent email systems learn information for the user to make smarter email management decisions. Our work draws from studies and interfaces in the field of human-computer interaction to motivate types of information that can benefit the user. We then apply tools from natural language processing and machine learning to learn this information to support intelligent user interfaces. Furthermore, this thesis develops new learning algorithms for natural language processing inspired by work with intelligent email. Our new Confidence-Weighted algorithms are then applied towards solving learning problems relevant to intelligent email.

We begin with a survey of work in human-computer interaction, motivating our goals and contrasting them with other artificial intelligence studies of email. We then describe our intelligent email framework and the contributions of this thesis.

1.1 Human-Computer Interaction

The field of human-computer interaction has both studied and proposed solutions for email overload [250]. Both Whittaker and Sidner [251], as well as Gwizdka and Chignell [113], studied how users organize and archive email, observing that a significant amount of time was spent foldering messages. More recently, Tang *et al.* [234] studied how these trends extended to new technologies, such as tagging. It is clear that some users spend a significant amount of time organizing their email with the expectation that the organization will later prove useful.

The field has also looked at decision making for incoming email, a scenario called email triage. Venolia *et al.* [243] proposed several ways to reduce email distractions, giving users better tools to understand and quickly process incoming email. A user survey by Dabbish *et al.* [65] studied how users make these decisions, finding that factors such as social information predicted a user's action on a message. Neustaedter *et al.* [180] explored social cues in depth in the context of triage and subsequently developed Snarf, a tool that displays a social overview of an inbox to support triage decisions [181].

Significant work has focused on email activity management and tools to support activities. Workers embed personal information management (PIM) directly into email clients, bypassing traditional PIM tools [90]. Studies have observed two distinct groups of email managers: the cleaners, those who processed email and removed it from their inbox, and the keepers, those who kept and managed information within the mail client. This latter group could greatly benefit from task management tools embedded into the mail client, reducing email clutter and improving organization [112]. Towards this goal a multitude of email task management tools have been suggested, such as Task-Vista [13], TV-ACTA [15, 16], collaborative activity managers [53, 172] and ActivityExplorer [174].

1.2 Artificial Intelligence and Email

Email has also been an attractive area of study for artificial intelligence, in particular natural language processing and machine learning. Many established tasks in these areas have been explored in email domains. Document classification has been applied as a way to sort emails into folders, and approaches have tried popular linear classification techniques [10], TFIDF centroid classifiers [213,214], rule induction [49] and pattern detection from graphs [1]. Another document classification task for email is call center classification, whereby incoming emails to support centers are automatically routed to necessary specialists or replied to automatically [28, 137, 178]. Perhaps the most common document classification task for email is spam detection, of which there are many hundreds of methods and studies [205]. Information extraction extracts free text from emails to populate information structures. Sarawagi and Cohen [207] used Markov models for extracting person names mentioned in emails. Information extraction from emails has also been part of larger email processing systems, such as automated help desk answering [137].

Topic models have also been adapted for email. McCallum *et al.* extended the author-topic model to include recipients, as well as inferring the role a sender might have in an email conversation [158]. Probabilistic latent semantic indexing (PLSI) has also been studied in the context of email [232]. Similar Bayesian graphical models have been applied for other applications, such as recipient prediction [185]. Other standard machine learning techniques have been tried on email for various applications, including co-training [132] and clustering [121, 169].

Established natural language processing tasks have been adapted for email, such as classification of speech acts to express actions like "propose" and "commit" [35, 50]. There has been a large amount of work on developing summarization systems for email. Some efforts have applied standard techniques for extractive sentence summaries, like centroid methods [248] and sentence classification using RIPPER [195]. Others have developed approaches based on properties unique to email threads, like question-answer summarization [165, 248] and summarization using the quotation structure of a thread [30, 31]. Some summarization tasks are unique to email, like indicative summaries of threads in information retrieval systems [179], summarizing topics in an email corpus [182, 183] or task focused summarization of emails [52].

Understanding large collections of email data requires analyzing social structures in email. Social network analysis and relational learning techniques can discover these patterns. As an example, identifying significant relationships in email, often according to known relationship types, is important for identifying relevant social communities in an organization [62, 74]. In addition to learning from communication patterns, latent concept models, such as LDA and PLSI have been used to associate semantic concepts with learned relationships [158, 257, 258]. Parties in the discovered relationships can be classified according to role (directory, manager, associate, etc.) [177] or more general roles like the leader of a group [38, 241]. Learning in these email networks is useful for other tasks, such as resolving named entity references [73, 119, 168].

1.2.1 Intelligent Agents

Clearly, there is no shortage of papers applying established learning techniques or adapting known applications to the email domain. However, the critical difference between these applications and the ones we consider in this thesis is the learning goal. We define learning problems to support intelligent user interfaces that give users better information by which to make email decisions. In contrast, the primary contribution of the above work concerns the learning method or how it can be extended to email. The learning goals are typically email analysis and understanding while ours are user focused.

Several recent major projects have developed intelligent assistants for the email and desktop environments. The goal of the Personalized Agent that Learns (PAL) project[1] is the development of cognitive assistants that improve the way computers support human information management. One aspect of this project, CALO (Cognitive Assistant that Learns and Organizes), has been brought to fruition through the IRIS client, which handles a wide variety of rich applications supported by intelligent assistants [47]. One such tool is LAPDOG, which acquires fully or partially automated procedures by observing a user's interaction with PIM applications [104]. These procedures can span multiple applications and includes tasks like setting up a meeting or tracking a requisition. Another example is TaskTracer, a learning system integrated with the user's desktop environment that infers the current task of the user [78, 230]. Once TaskTracer learns user activities it can support user actions by suggesting relevant webpages [144] or appropriate folders for finding and saving documents [7]. Finally, the RADAR project, also part of PAL, has focused on supporting users engaged in managing tasks like organizing conferences. RADAR developed components to manage user attention, learn important information from emails and learn to carry out tasks from user examples [101]. An automated assistant guides the user through a task based on previously observed behaviors.

These systems all developed learning technologies to support user actions. However, the key difference between these cognitive agents and our intelligent email applications

[1]`http://www.darpa.mil/IPTO/programs/pal/pal.asp`

is the nature of interaction between the user and the system. These applications manage information and applications for the user, relying on a cognitive agent to plan an event or carry out a task. While this may be useful for some tasks, users are wary of trusting complex learning systems. Glass *et al.* [107] found that users lacked confidence in the actions of systems like CALO because of their complexity and lack of transparency. In contrast, or goal is to enhance user actions rather than automate or supplant them. Our tools enhance the current email environment without replacing or redesigning it. Our tools provide users information; the user still completes the task. This style requires less trust from the user. We note that there are some previous systems that fall under our goal of intelligent email, including MailCat (later SwiftFile) which suggests folders to a user [214] and CutOnce, which suggests recipients for outgoing messages [5, 36]. However, this thesis is the first to define the goal of intelligent email as supporting smarter user actions and defining the design and methodology of intelligent email systems.

1.3 Thesis Goals

This thesis reflects my primary interests: helping people manage information. Information overload effects nearly all web users and information workers. The information revolution has brought instant knowledge on every topic – far more than we can consume, whatever our interest. Billions of pages of text, images and videos can be searched instantaneously with efficient storage and retrieval systems. People no longer thirst for information but drown in it. Users suffer from information overload, the inability to process and absorb the volume of presented information.

Organizing this information demands two efforts. First, a user-facing system must consider how to present, organize and highlight information. Second, the system must automatically process text information. The first draws from the field of intelligent user interfaces while the second from machine learning and natural language processing.

The interaction of these research areas demonstrates the research methodology of this thesis. First, we identify an intelligent user interface, whose design is motivated by results from user studies in the field of human computer interaction. We then define a measurable goal of the intelligent component for the interface. Our task will be to implement and evaluate this component, drawing from relevant technologies in machine learning and language processing.

Second, the application of learning technologies to new problems breeds fresh challenges. Not only do we question the success of an application, but we enumerate new challenges to the deployment of real world systems. From the applications explored in this thesis, we formulate new learning problems and address them within the computational linguistics community. This in turn leads to new algorithms which can eventually used to improve email applications.

This thesis is an example of the synthesis of two areas of research, artificial intelligence, in the form of machine learning and language processing, and intelligent user interfaces. Specifically, we direct this synthesis towards one of the most severe forms of information overload: email overload.

1.4 Intelligent Email

We define intelligent email as *intelligent systems that support interfaces for email.* These interfaces are information driven, enabling users to make faster, smarter and less error prone decisions in processing email. Each proposed interface is designed around a type of information learned by an intelligent component and motivated by human computer interaction. We evaluate the effectiveness of the applied learning technologies with standard machine learning metrics. Our primary research question is *can we apply learning methods towards identifying new information for email users?* The applications considered in this thesis answer yes. Our contributions to email research are the development of intelligent components using machine learning and language processing for intelligent user interfaces. We leave interface design and evaluation to future HCI work.

Of course, the far more interesting question is not "can we" but *how do we build intelligent email systems?* What lessons can we offer to those seeking to build these systems? Building intelligent email systems poses two significant challenges. First, typical learning tasks focus on large repositories of shared information, where learning goals are agnostic to specific users. The repositories are shared by multiple users with similar goals. In contrast, email users each have a private personalized information repository and their own set of preferences and relationships. Learning must be able to adapt to each individual email corpus and be specific for each user. Second, users are often wary of intelligent systems that automate data processing since they lack confidence in the system's actions [107]. Additionally, behavior or exposed output of complex learning systems can confuse users [60, 186, 231]. This is particularly true of email since users are accustomed to careful manual management of their data.

We address these two challenges by developing user oriented learning techniques for user supporting interfaces.

1.4.1 User Oriented Learning

Users have unique data and behaviors. Email models must be user specific, where representations are based on user behavior, content and context from the user's mailbox and social interactions. A single shared learning system may be sufficient for some tasks, but tasks with significant differences between users, especially those that require labeled data for supervised machine learning, require shared representations to enable learning across users. Alternatively, unsupervised learning can be user dependent without labeled data. These approaches provide a user oriented focus for learning, enabling successful learning despite the variety of user data.

1.4.2 User Supporting Interfaces

Since users may lack trust in automated systems, we favor interfaces that support user decision processes. These interfaces provide information about the user's email that can be easily incorporated into existing interfaces. Rather than automate, the interfaces are based on specific types of information learned by an intelligent component about email that can make a large email volume manageable. The information is typically non-critical; good performance adds benefit and failures are not intrusive. In this way, intelligent email supports, not supplants, existing user tasks.

1.5 Confidence-Weighted Learning

Research combining intelligence and interfaces is a two way street. Beyond our interest in how learning methods can be applied to intelligent email problems, we ask how intelligent email problems can inform new types of learning. In intelligent email, we contributed to intelligent user interfaces by adapting learning algorithms to our email objectives. In this section, we do the opposite: intelligent email needs inform new types of learning.

Through our intelligent email applications we encounter several learning challenges. First, we seek fast and efficient methods that learn from user actions (online learning.) These methods should scale to many examples as some email systems process extremely large volumes of data. Second, since some email tasks require user specific labels for supervised learning, we favor algorithms that can interact with users to elicit labels for a minimum number of examples (active learning.) Third, many email setting require methods that learn across users, sharing common and learning user specific behaviors (multi-domain learning.)

With these learning settings in mind we introduce a new online learning algorithm: Confidence-Weighted learning. Confidence-Weighted (CW) learning is a family of online learning algorithms sensitive to language distributions. While most online learning algorithms represent the current prediction as a linear classifier (a single parameter for each feature,) CW learning maintains a distribution over linear classifiers. Each feature weight is a Gaussian distribution, where the mean is the current feature weight and the variance the inverse confidence in the weight. For each training example, the algorithm updates the distribution to favor linear classifiers that correctly classify the example. Updates shift the mean proportional to the variance and decrease the variance of the distribution. The algorithm makes smaller changes for common features (low variance) as compared to rarely observed features (high variance.) By using parameter confidence for learning, CW algorithms improve over many state-of-the-art algorithms on several NLP datasets.

While parameter confidence improves learning, it also benefits other applications. When training across multiple processors in parallel, confidence can be used to guide the combination of many trained classifiers. In margin based active learning settings, we observe that the margin is not a sufficient proxy for confidence in a prediction. Instead, CW learning can provide a distribution over the margin, which gives a better estimation as to the confidence of the current prediction. This reduces the number of examples needed when learning a new task with a new user. We also formulate a new learning setting based on intelligent email: multi-domain learning. An email prediction system must be able to handle many users with divergent behaviors. However, we would like the learning algorithm to capture behaviors common to multiple users. In this multi-domain learning setting, we combine parameters learned both across domains and for individual domains using confidence. This approach learns behaviors both common across domains and specific to individual domains. We also consider a similar approach to domain/user adaptation and learning across many divergent users.

This work contributes to the fields of machine learning and natural language processing. Our new learning algorithm arises from our involvement in building intelligent email applications. In addition to this new algorithm, we consider applications that solve problems motivated by scale (large datasets), interface challenges (active learning) and introduce new problems inspired by intelligent email (multi-domain learning.)

1.6 Thesis Overview

To develop the concept of intelligent email we introduce a variety of new applications. These applications cover the different user modalities identified by Whittaker and Sidner [251]: communication, email activity management, and search and organization. Each of our applications target a user supporting interface and rely on user oriented learning.

We begin by examining the common problem of the forgotten attachment, where a user sends an email but forgets to attach the necessary document. Such mistakes can cause aggravation as important documents go missing. We build an attachment prediction system, which enables interfaces that can alert the user to missing attachments or trigger document suggestion systems. Users can then chose if they want to include an attachment. This task relies on supervised training, which can identify different signals for attachments across different users. We develop relational features that shift learning towards user behavior and not exclusively message content, which improves cross-user performance. This work has been published in the Conference on Email and Anti-Spam (CEAS) [81], the Conference on Intelligent User Interfaces (IUI) [82] and the American National Conference on Artificial Intelligence (AAAI) [88].

Next, we explore the task of email triage, the process of managing large amounts of email. We propose a reply management system supported by a reply predictor, automatically labeling messages that need a reply. This interface supports user controlled reply management, allowing the user to both mark messages manually and retrieve all messages that need a reply. The predictor enhances the interface by marking messages on behalf of the user. Since the reply predictor uses supervised learning to learn when a message needs a reply, it requires labeled data for each user. Therefore, we explore cross-user learning so that only a small set of source users need label their data. We draw upon work in social network analysis to design a new set of features for this task that allow a classifier trained on one user to be applied to another. This work has been published in the Conference on Email and Anti-Spam (CEAS) [80], the North East Student Colloquium on Artificial Intelligence (NESCAI) [79], the Conference on Intelligent User Interfaces (IUI) [82] and the American National Conference on Artificial Intelligence (AAAI) [88].

We then evaluate a series of representations for email, which can be used in other learning systems, such as email summarization. Our representations choose a few words from an email that depend on specific interests of a user, as different users may relate the same message to different semantic concepts. These words are selected using unsupervised learning to construct underlying concept models of a user's mailbox. These models are then used to select words from a message that relate the message to established user concepts. Since models can be learned on a per-user basis, the system does not require user-specific labeled data or cross-user learning. This work has been published in the Conference on Intelligent User Interfaces (IUI) [89] and the American National Conference on Artificial Intelligence (AAAI) [88].

Much HCI work for task management has introduced new interfaces that enable users to organize and view messages by activity [13]. To support these interfaces we develop an email activity classification system that sorts incoming emails into user defined activities. This problem is a multi-class classification system that has many classes without much training data. To improve the system, we use observations from the HCI community about activity composition to create user specific models of activities based on social information, message content and context (threads.) This work has been published in the Conference on Intelligent User Interfaces (IUI) [86] and the American National Conference on Artificial Intelligence (AAAI) [139].

Each one of these applications demonstrate that intelligent components can enable new types of email interfaces that support user behaviors. For each application, we apply user centric learning techniques to learn new information for the user.

The final chapters use intelligent email applications to motivate several learning settings and a new online learning algorithm. We introduce Confidence-Weighted learning for online linear classifiers. Using insights from natural language data, we develop this new algorithm and demonstrate its effectiveness on a variety of NLP datasets. We then consider three settings applicable to intelligent email: large scale learning, active learning and multi-domain learning. We show how parameter confidence can be used in each problem to improve over existing methods. We evaluate our algorithms on both email and other NLP tasks. This work has been published in the International Conference on Machine Learning (ICML) [85], the annual conference of the Association for Computation Linguistics (ACL) [83] and the conference on Empirical Methods for Natural Language Processing [84].

We close with a literature survey of related areas of email research and highlight the contributions of this thesis.

Chapter 2

Attachment Prediction

2.1 Introduction

Writing email is easy. It is easy to send an email instantly to hundreds of people with a single click and users often send dozens of emails a day, giving little time for a careful review of a message. This simplicity means users make mistakes during message composition, leading to annoying or damaging repercussions. Several researchers have addressed a number of such mistakes, such as omitting necessary recipients (recipient prediction) [36, 185] and sending an email to an unintended recipient (leak detection) [5, 34].

In this chapter we investigate one of the most common complaints of sent email: the missing attachment problem. Sending attachments is a convenient way of transferring files and an essential part of collaborative projects. About 22% of messages in the Enron corpus contain an attachment of some kind. Many attachments arrive in the context of activities, such as editing papers or preparing proposals, meaning the attachment is part of a longer email, describing updates to the document or project. This means that attaching a document is often the last step in the composition process, something easily forgotten. As a result, emails arrive without their intended attachments, only to generate a flurry of emails attempting to correct the problem. The missing attachment problem can create a serious disruption in workflow when a critical document is not received on time, especially when the sender is now out of contact.

A number of possible interfaces could address this problem. For example, the system could alert the user to a missing attachment when she sends the message, similar to alerts for missing subjects. This is the approach taken by projects such as the Mac Mail missing attachment plugin. [1] Alternatively, an attachment recommendation system could provide utility while subtly reminding the user to attach a document. Additionally, the attachment button could become more prominent when the message needs an attachment. Whatever the interface, an intelligent component is required to determine if the interface should be activated.

We take attachment prediction as our first attempt to apply AI techniques to support interfaces for intelligent email. When a message is composed, the AI system processes the text of the message to determine if it should contain an attachment. The prediction can then be used to alert the user or offer suggestions. We rely on machine learning techniques and a large corpus of emails (Enron) to learn how to identify messages requiring an attachment. Work in this chapter is based on previously published work in the Conference on Email and Anti-Spam (CEAS) [81], the Conference on Intelligent User Interfaces (IUI) [82] and the American National Conference on Artificial Intelligence (AAAI) [88].

2.2 Learning System

Attachment prediction, as well as many other intelligent email tasks, can be framed as classification problems. We treat attachment prediction as a binary classification problem, where for each email the system outputs a label, either positive (*needs attachment*) or negative (*does not need attachment*). As is common in NLP, each email is represented as a sparse, high-dimensional binary vector $\mathbf{x} \in \{0, 1\}^D$, where each dimension represents a feature. Features can be words in the email or other properties of the message. Real valued features are quantized for representation in this format. Learning is performed

[1] *Avoid sending Mail with unattached attachments* at `http://www.macosxhints.com`

with a logistic regression classifier since it is easy to train and achieves best or close to best accuracy in many text classification tasks [103]. It achieved better performance than other trainable classifiers in preliminary experiments for our task as well. We used the conditional maximum entropy implementation in MALLET (which for this particular problem is equivalent to logistic regression) for its ease of use and text processing capabilities [155]. We evaluated our system using F_1, the harmonic mean of precision and recall, standard metrics for classification tasks.

2.3 Features

The primary indicator as to whether or not a message needs an attachment is text that describes or refers to an attachment. Phrases such as "Please see the attachment" and "the budget proposal is attached" are the best indicators. We extract this signal using bag-of-words features, which are widely used in many text classification problems. In addition to words from the body and subject, we include the recipients and senders of the message as features, a common practice for email classification [36,50,214].

In addition to these baseline features, we developed features particular to this task.

1. The position of the stem "attach" in the email (first or last sentence).
2. The stem "attach" conjoined with all words in close proximity (5 words) and their direction (before/after).
3. If the body of the message was almost empty or totally empty (emails with no text can indicate the presence of an attachment.)
4. Relational Features- Using previous emails sent and received by a user, we constructed relational features that described attachment relationships between users. Features in this class included:
    - Total number of messages sent to and received from a user.

- Total number of messages with attachments sent to and received from a user.
- Percentage of messages that contained attachments that were sent to and received from a user.
- The feature "no recipients receive attachments from this user" and "all recipients receive attachment from this user."
- All of the above for the CC field.

We also explored features based on n-grams, speech acts [35, 50] and a dependency parse of the message; they did not improve performance.

2.4 Corpus

Our attachment prediction system was evaluated on the Enron email corpus [135], a publicly available corpus with about 150 users and 250,000 emails. While this is the largest and currently only corpus for large scale email research, it posed several problems for attachment prediction.

First, the publicly available dataset[2] does not contain attachment information, which was removed during dataset construction. We obtained a database dump of the original FERC corpus and matched it against the public corpus to find emails that contain attachments. Second, a large number of attachments were of type ".html" or ".eml," which are html text alternatives and forwarded messages, not actual attachments. To avoid confusion we removed forwarded messages from the corpus by looking for the string "fwd" in the email subject. Next, many emails that contained attachments did so as inline attachments, such as embedded pictures in a rich-text message. There is no easy way to differentiate these automatically embedded images from user attached pictures. To filter out these attachments we only applied the *needs attachment* label to message's that had attachments with a popular office document format, such as ".doc," ".xls," ".pdf," etc. The complete list of document types that were included are listed in table 2.1.

[2]`http://www.cs.cmu.edu/~enron/`

File Types Included: `doc xls pdf ppt rtf wpd pps pub`

Table 2.1: The types of attachments in the Enron corpus that were labeled as *needs attachment*. All other attachment types were ignored.

Since Enron messages come from production environments, residual artifacts were introduced by various email clients. Many attachment messages included an automatic list of attachments in the body of the message, such as “⟨⟨File: E&Y Memo.doc⟩⟩” or “–E&Y Memo.doc”. Others included automatically added text that directed the recipient to see the attachments. Since these are generated by the client once a user attached a document to a message, they cannot be used for learning. To remove these artifacts, 200 random email messages were hand examined for such attributes and regular expressions were constructed to remove the offending text. This process was repeated until the randomly selected messages were free from such text.

The resulting corpus contained 197,641 messages, of which 43,764 had some type of attachment and X had an attachment used for learning. Since we sought to build a profile for each user, we selected messages only from the sent folders of Enron users. 15,000 messages were selected at random from sent mail folders, comprising 1,020 messages with attachments and 13,980 messages without from 144 users. This mix of messages represents real world data so we did not balance positive and negative examples. Profiles were constructed for each Enron user from emails not in the evaluation corpus.

2.5 Evaluation

The problem of a missing attachment has prompted frustrated users to develop various ad-hoc solutions to the problem. For example, a Mac OS X Mail plugin relies on the presence of the word “attach” to alert users to possible missing attachments. [3] Since these rule-based systems are a common and simple solution, we used them as a baseline for comparison.

[3] *Avoid sending Mail with unattached attachments* at `http://www.macosxhints.com`

Split	*System*	*Precision*	*Recall*	F_1
	Rule Based	0.8223	0.4490	0.5808
User	No Relations	0.8381	0.6647	0.7414†
	All Features	0.8301	0.6706	0.7419†
Cross-User	No Relations	0.8049	0.5461	0.6507$*$
	All Features	0.7981	0.5618	0.6594$*$

Table 2.2: Attachment prediction results on Enron mail for three systems: rule based (the word form "attach"), no relational features, and all features. Numbers are aggregate results across 10-fold cross validation. $*$ and † indicate statistical significance at $p = .01$ and $p = .001$ respectively against the baseline using McNemar's test.

In our rule-based system, messages with the stem "attach" (e.g. "attach," "attached," "attaching," "attachment") are labeled as *needs attachment*. All other messages are labeled *does not need attachment*.

We evaluated two versions of the learning system: one that includes all the features (39,380 features) and is the best system (All Features), and one where relation features – 72 features total – are removed (No Relations). This evaluates the contribution of relational features to the attachment prediction problem.

A major concern of any personalized system is cross-user performance. How well does a system perform when trained on one user and tested on another? Typically, a system is trained and tested on the same group of users. However, when predictions are required for a new email user, there are no existing data or user-provided labels for training. This cross-user setting means no training emails for a test user are available, a problem for systems that rely on the content of the messages. Therefore, we considered both single-user and cross-user (adaptation) settings, the latter simulating out-of-box performance for new users.

Each system was evaluated on the 15,000 Enron messages using 10-fold cross validation for both user and cross-user evaluations. For cross-user evaluations, users were divided into folds so emails from a single user were only used for training or testing but not both. Aggregate results are reported across the 10 folds for each split in table 2.2.

While the rule based system can identify attachment messages with high precision – messages with the word “attach” usually have attachments – it fails to find even half of the emails with attachments. In contrast, the learning system achieves slightly higher precision and much higher recall, finding two-thirds of the attachment messages. These results are significant at $p = .001$ using McNemar’s test. In this task, relational features do not give any noticeable improvement to the results.

The cross-user setting is more difficult. While performance of the learning system still exceeds the rule based system – which required no training data so performance remains unchanged – recall drops by more than 10 points and precision drops below the rule based system. Attachment prediction relies heavily on message content, which changes between users. Here, relational features are slightly more helpful as they are more robust to a change in user, yielding a 0.0087 increase in F_1.

2.6 Conclusion

In this chapter we explored the missing attachment problem and evaluated a system designed to trigger a user interface that could reduce missing attachment rates. We have shown that a learning based system can identify signals in email messages that indicate the intention to include an attachment. These signals are more reliable than rule-based features, which identify less than half of the messages that need attachments. While systems trained on each user can do well, cross-user performance remains a problem. We investigate solutions for this setting in the next chapter.

Chapter 3

Reply Prediction

3.1 Reply Prediction

Email inboxes typically display a limited amount of information about each email, usually the subject, sender and date. Users are then expected to perform email triage—the process of making decisions about how to handle these emails—based on this information. As the number of received email messages increases, tools to assist users with email triage become increasingly important. Additional concise and relevant information about each message can speed up the decision-making process.

In defining email overload, Whittaker and Sidner [251] reported that users received a large number of messages each day and were unable to process the messages properly. One of the main complaints reported regarded replying to emails.

Waiting to hear back from another ...employee can mean delays in accomplishing a particular task, which can ... have significant impact on our overall operations. Depending on the situation, it can either be critical or just frustrating.

One of my pet-peeves is when someone does not get back to me, but I am one of the worst offenders. I get so many e-mails (average 30-40/day) and phone messages (15-20) that I cannot keep up and also do my real job...

Given the sheer volume of stuff that passes through here. I mean I couldn't even give *you a percentage of how much is missed. I mean — not necessarily missed but certainly recorded but never followed up on.*

With an average of 49 messages a day reported by participants, what would be considered a low volume of messages today, users could not properly manage messages for which they were expected to reply. This problem is highlighted during email triage, where a user has a limited amount of time to process a large number of emails. The user may be able to dedicate time to responding to messages, but the messages needing a response are lost in the volume.

Our goal is to provide tools to enable smarter email decisions, in this case providing information to inform choices about which messages to read and reply to. Towards that end we develop a reply management system that provides user tools for managing outstanding reply commitments. Our prototype interface both provides information by indicating which messages require a reply, and allows the user to manage these messages, marking them as replied or needs reply and displaying all outstanding reply commitments. Viewing messages that need a reply is useful, but manually labeling messages as they are read is time consuming.

To automate this process we develop a reply predictor, an intelligent system that automatically identifies and marks messages that need a reply. We begin by considering a standard set of features for email. However, these features do not perform well for the task. More importantly, these features are user specific, requiring labeled data for each individual user for training. This places a prohibitive labeling cost on the user. Instead, we consider features that both improve reply predictor accuracy and are shared between users, enabling cross-user learning. We draw from various studies in email learning to create a shared representation for all users. The features themselves are shared but many are created in a user specific manner. We show that these features both improve learning and reduce error from transfer between users.

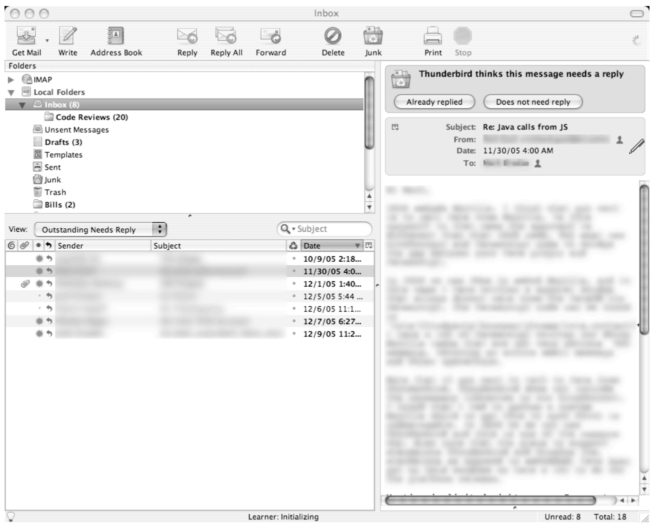
We begin with a description of our interface and show how it can be used for reply management. We then focus on the problem of predicting which messages require replies, and learning challenges for this task. Specifically, we explore the problem of cross-user performance, introduced in the previous chapter. The work in this chapter is based on previously published work in the Conference on Email and Anti-Spam (CEAS) [80], the North East Student Colloquium on Artificial Intelligence (NESCAI) [79], the Conference on Intelligent User Interfaces (IUI) [82] and the American National Conference on Artificial Intelligence (AAAI) [88].

3.2 User Interface

Our interface design had two distinct goals: to be informative to the triage process and to provide a reply management tool. The system is implemented in the Mozilla foundation's Thunderbird[1] mail client and is designed to be useful as a standalone tool for reply management (Figure 3.1). The interface integrates into the Thunderbird client and mimics existing interface models, such as spam filtering, to create a familiar experience for the user. When a new message arrives, the email is passed to the prediction system which assigns a label, *needs reply* or *does not need reply*. The prediction is displayed next to the message in the user's inbox in a new column, with a red arrow indicating needs reply. Similarly, when the message is displayed in the preview pane, a green bar above the message indicates that "Thunderbird thinks this message needs a reply." The user can correct the prediction system through buttons on the green bar or by clicking on the *needs reply* column in the inbox view. By adding a *needs reply* column to the inbox view, the user can use this information when triaging her mailbox.

If a user replies to a *needs reply* message, the red arrow fades to grey and the green bar changes its message to indicate that the reply has been fulfilled. This can be toggled to restore the "outstanding needs reply" status. Finally, we include a view option that allows the user to view all messages that need a reply but are as yet unreplied. This interface allows a user to automatically build a list of outstanding replies and quickly find and

[1]`http://www.mozilla.com/en-US/thunderbird/`.



View: Outstanding Needs Reply

Subject: Re: Java calls from JS
From:
Date: 11/30/05 4:00 AM
To:

10/9/05 2:18...
11/30/05 4:0...
12/1/05 1:40...
12/5/05 5:44 ...
12/6/05 11:1...
12/7/05 6:27...
12/9/05 11:2...

Figure 3.1: The reply management interface in Thunderbird has an additional column in the bottom left mailbox pane that displays a reply arrow next to messages that need a reply. Two buttons above the message contents read: “Already replied,” which marks the response as fulfilled and “Does not need a reply.” The user can see messages that still need a reply by selecting the “Outstanding Needs Reply” option (as shown.)

handle them when time permits.

3.3 Prediction System

While our reply management system provides tools for marking and finding messages that need a reply, marking messages is time consuming. We now explore an intelligent system to automate this process.

As in the previous chapter, we treat reply prediction as a binary classification problem, where for each email the system outputs a label, either *needs reply* or *does not need reply.* We use the same learning infrastructure as in the previous chapter, including a logistic regression classifier. (See appendix A.1.2.)

Email classification has been studied for a number of other tasks [10, 49, 50, 214]. A standard feature set has emerged that includes a bag-of-words representation of the body and subject, as well as the recipients and sender of the message. This feature set proved effective for attachment prediction and we use these features as a baseline.

In addition to bag-of-words, we added a special feature for reply prediction. The presence of questions in an email is highly indicative of the need for a reply, even without question marks, such as, “Can you please send me a report today.” To detect these types of requests, the system examines all words in the training data and finds those that best indicate a question by measuring the frequency with which the word appears in a question, ie. the number of times the word appears in a question divided by the total number of times the word appears. The system collected a set of the 30 words with the highest question frequency usage and added a feature *has question word* to messages that contained one of these words. This allowed for the identification of questions even without the presence of question marks.

3.4 Challenges: Cross-User Performance

Despite our use of the standard email classification feature set, we found that our learning system was unable to predict if an email needed a reply accurately. Upon investigation, we determined that two problems prevented successful learning: noisy labels and poor features.

Supervised learning methods for classification are based on the availability of sufficient labeled training data. We inferred labels from previous user actions by examining the sent mail folder; if a message received a response within 48 hours, it was marked as *needs reply*. Otherwise, we marked it as *does not need reply*. However, these labels proved unreliable for learning. All users hand labeled their mailbox through our user interface. These gold labels were compared to our inferred labels; average agreement was only 0.65. A low agreement with previous behavior could be caused by the user missing emails, replying to messages offline, or simply forgetting to reply. Our system could not learn from the behavior of an overloaded user; instead, it performed well on user provided labels. We also had two annotators label mail from two of our users; an inter-annotator agreement of only 0.77 shows the difficulty of the problem.

A dependence on hand labels introduces a new complexity: obtaining labels for a new user. Obviously, it is impractical to ask a user to label hundreds of emails, and it is unlikely that a user would tolerate a badly performing system. Therefore, we needed a system that did well, not only on a source user with training data, but out of the box for a new user without training data: cross-user performance. The standard feature set for email is unlikely to model behavior across users. Consider the feature *from Tahlia*. This feature is only useful to someone who responds to Tahlia, but new users probably have no contact with this person. Similarly, while some bag of words features may perform well across users (e.g. "urgent", "reply") many user specific words will not, such as names of projects, companies, locations, or people. The solutions described above must be tailored to the end user. Email variability makes the standard feature set unlikely to perform well.

In addition to creating problems for transfer, the standard feature set performed poorly in general for the prediction task. Consider an identical email received by two users, Merrick and Jarvis, who may react differently to the message. One may delete it while the other may reply. A system would be unable to discern this behavior just based on the words and header information. Instead, the person's previous behavior towards the sender as well as their relationship with the sender may be better predictors. For example, if Jarvis only CCed on an email sent to Merrick, then Merrick is likely to reply while Jarvis is not. Standard features do not allow for building our reply predictor.

3.5 Features for Reply Prediction

This section develops a new set of features for reply prediction with two goals in mind. First, these features should replace the user specific features, such as sender identity, with generic features that more naturally permit cross-user performance. Consider the two users Merrick and Jarvis who use the reply prediction system. Merrick has a coworker named Tahlia that is unknown to Jarvis. The feature *from Tahlia* is useful only for Merrick.

However, Jarvis may have a similar relationship with a coworker Jaeger. In this model, *from Tahlia* and *from Jaeger* are equivalent features for Merrick and Jarvis respectively. An ideal feature would map both of these coworkers to the same feature, such as *from coworker* of *from frequent contact*.

It is useful to conceptualize the feature function for learning as a product of the user as well as the document and label. A feature function determines the value of a feature based on an example $\boldsymbol{x}$ and a label y, ie. the ith feature is $\Phi_i(\boldsymbol{x}, y) \in \{0, 1\}$ for binary features. If the ith feature is *from Tahlia*, then the function will return 1 iff the example $\boldsymbol{x}$ is an email from Tahlia. By extending these functions to consider the user u, the new function $\Phi_i(\boldsymbol{x}, y, u)$ can change value depending on a user's profile. In this case, we can create a new feature *from frequent contact*, that checks if the sender is a frequent contact of u. This function can return a different value for the same $\boldsymbol{x}$ and y for different users. We think of these as *deictic* features since the values are shared among many users but the meaning depends on the specific user and context. For more details on feature functions, see appendix A.1.

Second, while we expect deictic features to improve cross-user performance, we also believe they will improve performance overall by aggregating features for a single user. In addition to frequently emailing Tahlia, Merrick may often communicate with Dustin. Creating a single feature for both Tahlia and Dunstin increases the observed data for this feature and generalizes learning across features.

While lexical content performs poorly for reply prediction, other content features may be helpful. In particular, the presence of questions in an email may indicate the need for a reply. Along these lines we develop several content-dependent features that can apply to multiple users without user specific feature extraction.

We begin by showing how deictic features emerge from other email analysis tasks based on relational learning for email. These features capture social relationships, user behaviors, group relations and time indicators. We then outline several content features for reply prediction, based on questions in the message, or properties of email in general, like attachments. We also capture second order interactions between some of the most useful deictic and content features by selecting a few feature conjunctions. In total, these features lead to 160 binary features, instead of the roughly 15,000 used for our baseline system.

3.5.1 Deictic Features

The intuition behind deictic features is to replace the specific identity of users with indicators of the relationship or social role of the user. Instead of a single feature *from Tahlia*, several feature indicate the behavior between Tahlia and Merrick, like the frequency of communication and how often Merrick replies. These features are likely common to many senders across many users. Our approach is similar to feature aggregation in relational learning, a setting in which examples are linked in a complex relational network. Email communication graphs are an instance of such a network. Feature aggregation in relational networks induces new general features from several attributes of a single or multiple objects. Perlich and Provost [188] define a hierarchy of features based on feature aggregation for relational learning. The simplest form of aggregation maps several atomic values to categorical or numerical values, such as computing the average value for a group of product purchases. More complex forms are multi-type aggregation, which creates features from different types of entities, such as the slope of product prices over time.

Relational learning as applied to email has generated several learning tasks which analyze the social structure of communication graphs. Three tasks in particular motivate our feature formulations: relationship identification, behavioral analysis and social group analysis. We survey each of these applications and show the resulting features we construct for our task. In each application, we highlight themes common to multiple approaches and develop heuristic features along these themes. In future work, we can consider integrating these relational learning systems with our reply predictor to learn more informative social features.

Relationship Identification

A collection of emails forms a communications graph between users from which social relationships can be inferred. The discovery and characterization of these relationships is often useful in understanding organizational structure. Such methods could be used to analyze the productivity of an organization, find knowledgeable teams of people or find suspicious behavior for legal discovery. One such technique is the discovery of manager subordinate relationships based on communication patterns. Diehl *et al.* [74] use supervised ranking to learn these relationships, using the pattern of communication between users as features for ranking. They observe that these types of patterns are more effective for determining relationship types than the content of the communications. This justifies our pursuit of communication pattern features as being more informative, and more generalizable, for reply prediction than content features. Identifying and characterizing these relationships can be helpful for other tasks. The SocialXplain project identifies relationships in a social network and generates explanations for these relationships in order to make decisions for mobile phone communications [153]. Culotta *et al.* [62] identify personal or close relationships between users to suggest contacts for a contact manager. Similarly, we hope that characterizing these relationships can be useful for reply prediction. This usage of communication patterns is a form of feature aggregation, whereby we induce features for communication patterns from a record of previous exchanges.

While these methods (and our features) use communication patterns alone, others have had success at applying probabilistic models that learn an accompanying descriptor of the content relevant to identified relationships. Zhou *et al.* [258] propose Bayesian models for semantic community discovery in social networks that identify communities of individuals and yield topic descriptors for these communities. Their work builds on latent Dirichlet allocation (LDA) and other topic models to create generative models of communities in terms of semantic topics and users. A similar approach is taken by Zhang *et al.* [257], who employ probabilistic latent semantic analysis (PLSA) to model the content of communications between pairs of users. Their model assumes that the topic dynamics of one user will affect other nearby users in the communications graph. Finally, McCallum *et al.* [158] extend LDA to an author-recipient-topic model, which assumes the interaction between a sender and the recipients influences an email's contents. They also propose a role based model, whereby each sender has a different role or persona that influences the choice of topics and words for a document. While our features are based on relationships and do not include semantic properties, future work could include such features as well. It is possible that a reply may be expected not only when a specific sender or recipient are involved, but when they are involved in discussing a specific topic.

Beyond identifying relationships in email networks, relationships can be classified by type. We mentioned above the work of Diehl *et al.* who identify manager subordinate relationships. Namata *et al.* [177] use supervised learning to identify other organizational titles of Enron employees. Their features include statistics of communications between users to learn titles, such as director, manager and associate. Carvalho *et al.* [38] use similar features from communication patterns for supervised learning to discover the leader of predefined groups of users. Tyler *et al.* [241] identify communities and users in leadership roles using a betweeness centrality algorithm.

In addition to finding groups for email analysis, these groups can be useful for other applications, such as person name disambiguation. Email messages refer to other people in informal ways and often times these mentions are ambiguous, such as "I'll talk to Susan" when there are multiple Susans. In email communications, an effective way to identify the referents of these mentions is through social groups discovered in communication graphs, like the ones described above. Diehl *et al.* [73] achieve name reference resolution through analyzing the communication patterns of users, assuming that the likely referent communicates with both members of the communications pair. Minkov *et al.* [168] use communications between users to build a graph structure connecting users and emails. Random walks through the graph can disambiguate named references. Hölzer *et al.* [119] take a different approach and use the Internet to identify social connections, relying on the collocation of email addresses on the same webpage. This approach can augment one based on the communications graph structure. Finally, Elsayed and Oard [94, 95] merge multiple identities when a single user has multiple email addresses or names. While we do not analyze the content for person names, this application shows the quality of information present in discovered social groups.

To capture the idea behind these learning methods for relationship identification and characterization, we include features that generalize the identity of the sender and recipients using simple heuristics. The success of these features could warrant the application of more complex feature induction. Our primary purpose is to substitute the specific identities of individual users with general properties describing their relationship to the user. We consider the following features.

- **User in addressbook**: Features to indicate if the message sender, a recipient or the first recipient are in the user's addressbook. Addressbook membership typically indicates a closer connection with the user and could be learned by some of the methods describe above, similar to Culotta *et al.* [62].

- **Supervisor features**: Given the email address of the user's supervisor, we include a few features that encode the relationship between the user and their supervisor. Our features include *sender is my supervisor*, *my supervisor is the first in the "to" field*, *my supervisor is the only recipient in the "to" field*, *my supervisor is (in|only one in|first in) the ("to"|"cc") list*. These features generalize across supervisor-subordinate relationships. If the identification of the supervisor is not provided by the user or a predefined organizational structure, learning systems can identify this information automatically, such as in Diehl *et al.* [74]. We can also include other roles through the application of role identification algorithms, such as those of Namata *et al.* [177] and Carvalho *et al.* [38].

- **Sender domain**: Does the message sender have the same email domain as the user. This indicates someone in the same organization as the user, which may indicate a closer relationship. We approximate group membership by domain alone, although features could be added for discovered social groups.

- **Sent to mailing list**: The message was sent to a mailing list. Mailing lists are detected by looking for the string "list" in the recipient address. Some mailing list systems contain special headers that can be used in place of this heuristic. This feature replaces each individual mailing list a user may encounter.

Behavioral Analysis

In addition to identifying and characterizing roles, we can include features that indicate the behavioral patterns with regards to different users. In the applications considered above, these types of features were typically used to learn relationships. In our task, we include these features directly rather than learning a role or relationship. Namata *et al.* and Carvalho *et al.* both use features based on the number and type of messages sent between two users for role identification. Their features include the number of messages sent between two users and the number sent and received by each user. Following their example, we include features based on the history of communications between two users.

We encode the behavioral patterns of a user with regard to specific senders by examining the training data of the user. For each email in the training set, we record the sender, recipients and whether or not the user replied to the message. Using this user profile we extract behavioral features from a message. Our behavioral features include:

- **Number received messages**: The number of messages received from this sender (binned into 0, < 1, < 2, < 5, < 7, ≥ 7).

- **Total emails with recipient**: The number of emails received with this user as a recipient (binned as above.)

- **Times replied to sender**: The number of times the user has replied to this sender (binned as above.)

- **Percentage replied**: The percentage of emails received from this sender to which the user has replied (binned into 0, < 25, < 50, < 65, ≥ 65.)

Others have found these types of behavioral patterns useful for email analysis. In particular, of the many ways to identify spam and spammers in an email network, communication patterns readily identify suspicious behavior. Stolfo *et al.* [229] use email behaviors, such as the volume and velocity of email traffic as well as deviation from established social cliques, to identify emerging viruses as they propagate amongst users. Their work shows that stable behavioral patterns of users can be identified and then used to detect deviations from this norm. A similar motivation can be used to analyze email graphs for suspicious user behaviors, detected by shifts in established email communication patterns. Diesner and Carley [75] apply social network analysis algorithms based on centrality measures to find patterns in email communications and identify influential individuals. By applying these types of algorithms to the Enron dataset they were able to identify shifts in established user behaviors that corresponded to events in the Enron crisis.

Social Group Analysis

The above methods are primarily concerned with the discovery of social groups in an email communications graph. These groups can be useful for other applications, such as populating a contact manager [62]. Furthermore, the task of person name disambiguation shows that these groups can be informative for other tasks. For example, a user may have a secondary role in a group, such as observing a project without participating. Knowing the user was only an observer would decrease the chance the user would participate in communications about the project. While we do not discover these social groups for our application, we can approximate the user's connection to social groups through construction of email headers. In this case, if the user appears in the "cc" field of an email he or she is likely in an observer role. Along these lines, we add several features regarding the user's position in and other relevant properties of these fields.

- **Features about me**: These features indicate how the user appears in the participant list for the email. There are separate features for *only sent to me*, *I am the first one in the "to" field*, *I sent the message*, *I am not in the "to" or "cc" fields*, *I am (in | only one in |first in) the ("to"|"cc") field*.

- **No recipients**: The email has no listed recipients (everyone was BCCed.) These are often announcement messages.

- **Number of recipients**: The number of recipients included on the message binned by the integer value of $\log$ the number of recipients. Messages with a large number of recipients may not expect a reply.

From these studies we conclude three lessons about feature construction. First, learning is more effective when we can generalize away from specific user identities towards roles of users in a group. Much work has focused on learning these groups and identifying roles. We add features indicating the possible role of a person visa vie the user in several ways, such as presence in the user's addressbook or if the sender is a supervisor. Second, beyond labeling roles, we may learn something from the general behavioral pattern of a user. For reply prediction, we include features relevant to the user's reply history in regards to the sender of the message. The learning algorithm can then weigh the predictiveness of previous user actions in future reply actions. Third, the user's role in social groups is useful for many learning tasks, such as person name disambiguation. We approximate the user's role in the current email by extracting features based on the construction of the email headers. These three types of features approximate the possible output from other learning systems and are a first step at using such information for our task.

Temporal Features

A final characteristic we consider is the temporal patterns of communication. Stolfo *et al.* [228] use temporal information to measure profile stability of users through applications of dynamic social network analysis. They found that spammers can be identified by comparing profile stability with legitimate email users. In reply prediction, temporal information may indicate the expectation of a reply. For example, the user may receive work emails during the work day that require a response but is unlikely to receive such emails at night or on the weekend. Along these lines, we add features to indicate the time and day of the email.

- **The time of day**: Four features indicate if the email was received in the morning (6am to 12pm), afternoon (12pm to 5pm), evening (5pm to 10pm) or night (10pm to 6am).
- **Weekday/Weekend**: Two features to indicate if the email was received on a weekday (Monday - Friday) or weekend (Saturday or Sunday.)

3.5.2 Content Features

While we cannot use the lexical items in an email as features directly, there is still important information in the language of the message. For example, a message that begins with "Dear All" or "To Valued Customer" is far less likely to indicate an expected reply, as opposed to salutations of the form "Dear John."[2]Additionally, other properties of the content, such as the presence of an attachment or the length, may be correlated with reply expectations. These features must be extracted away from the words of an email so that features can be shared.

First, we consider the presence of questions in the message. Obviously, emails that ask questions of the user are more likely to expect a reply. Furthermore, the location of the questions in the message may be revealing. For example, emails that end with questions may be different from those that contain questions in the middle of the text.

While identifying questions in formal written text is often as simple as finding sentences ending with question marks, the task is considerably more difficult for informal text. In general, many do not use punctuation marks (or punctuation in general.) Many

[2]Assuming of course that one's name is actually John.

questions in email are stated as declaratives and rhetorical questions do not expect a response. Zechner and Lavie [256] develop a learning system for question detection for summarization and Shrestha and McKeown [220] consider the same problem for email. In this work, we rely on a simple heuristic and leave the application of more complex systems to future work.

We identify questions by the presence of question marks and assume sentences end in either a "?", "!", "." or a new line. Using this heuristic we extract the following features:

- **Contains question mark:** The email contains a question mark.
- **Percentage questions:** The percentage of sentences ending in question marks (binned into 0, < 5, < 10, < 30, ≥ 30.)
- **Number of question marks:** The number of questions marks in the email (binned into 0, < 2, ≥ 2.)
- **Only questions:** All sentences are questions.
- **Question near beginning:** A question appears in the first three sentences of the email.
- **Ends in question:** The last sentence is a question.
- **Question near end:** One of the last three sentences is a question.
- **Subject contains question:** There is a question mark in the subject.
- **Subject ends with question:** The subject ends with a question mark.

We also consider features that depend on other aspects of the message contents, either features unique to email or general features about content.

- **Subject contains RE**: The subject of the email contains the string "RE:", which indicates the email is part of a reply chain.

- **Has attachment**: The email contains an attachment.

- **Contains URL**: The email contains a URL for a website.

- **Length**: The message's length as measured by the $\log$ length as an integer value.

- **Salutation**: The email begins with a salutation to this user. We indicate the presence of a salutation by the words "hello," "hey," "dear," "hi," or the empty string followed by either the first or last name of the user.

3.5.3 Feature Conjunctions

It is possible that our features may interact in ways useful for prediction. An email may be more likely to need a reply if the message was only sent to me by my supervisor than either of those two features alone. Many machine learning algorithms capture second order feature interactions by using non-linear learning methods or kernels.[3] In NLP problems, which typically use linear classifiers with linear kernels, feature conjunctions capture second order interactions. A new feature is added whenever two other features are present in the example. The linear classifier then learns a new weight for this feature, which indicates the interaction between these features. An example of a commonly used feature conjunction is a bi-grams, which is a feature created whenever two words appear adjacent in a document. Other examples of feature conjunctions include feature induction, which progressively adds new features that conjoin two existing informative features [122, 156].

We manually constructed feature conjunctions by selecting some of the highly predictive features. A list of these conjunctions appears in table 3.1.

[3]See Jakulin [123] for an analysis of second order and higher feature interactions.

Conjunctions
i am in to list & my boss in cc list
only to me & sender is my boss
only to me & from my supervisor
subject contains question & email ends in question
subject has RE & subject ends in question
i am in to list & sender is my boss
i am only one in to list & my boss in cc list
(yes|no) question near end & i am only one in to list
does(not) end in question & i am only one in to list
more than 30% questions & i am only one in to list
few questions & i am only one in to list
sender in addressbook & i am only one in to list

Table 3.1: A list of the feature conjunctions created to capture second order feature interactions.

3.6 Evaluation

3.6.1 Corpus

We evaluated our predictor on the inboxes of four users, Merrick, Jarvis, Jaeger and Tahlia.[4] Each user hand labeled his or her messages as either *needs reply* (positive) or *does not need reply* (negative). The datasets represent received messages that went to the user's inbox and not those automatically filtered into other folders upon arrival, excluding most automated messages, listserv mail and spam. The mail included 2391 emails, where each user's mail ranged over 1 to 6 months and included between 443 and 741 emails, with about one-third needing a reply.

[4]For obvious privacy reasons, large corpora with all the information we need are not available. Existing public corpora like the Enron corpus do not have enough information; for instance, they lack full headers and reply data. Finally, hand labeling other corpora for *needs reply* was very difficult since we had no context for any of the messages.

3.6.2 Baseline

Before applying learning to a new task, it is important to gauge the difficulty of the task. Sometimes, simple rules are enough. We constructed a set of rules and classified email as *needs reply* according to the presence or absence of these features. The rules we used were *email contains "?"*, *email contains "reply" or "urgent"*, and *email is from someone in address book*. These rules yielded an f-measure of 0.52 averaged across 10 runs. Clearly, simple rules are insufficient.

3.6.3 Results

We evaluated our reply prediction features (RP Features) and compared them to typical email features (Baseline Features). Our evaluation was conducted using the system described above. For each run, we split the corpus into train (80%) and test (20%) sets. Over-fitting was alleviated with a Gaussian prior on the weights of the logistic regression. All additional parameters were set to the default provided by MALLET [155].

We first evaluated the performance of our system on all of the users together. This shows the relative performance of a classifier using Baseline Features and a classifier using RP Features on all available data. We also included a third classifier trained using both sets of features. This provided a small improvement in performance as it allowed the model to learn some details about the user's specific behavior. Table 3.2 shows that all classifiers achieve good precision, but the RP Features have much higher recall. A variety of content features signal *needs reply* but a smaller set of RP Features captures this information.

Next, we examined the behavior of our system on individual users. By examining how each user behaved, we hoped that the system could learn individual behaviors more accurately. These tests included all available features and indicates the best a classifier can do with all available data for a user. Performance was better than a joint classifier. When each user was tested individually, average performance reached 0.67 F_1. The ideal

Features	*Recall*	*Precision*	F_1
Baseline Features	0.40	0.73	0.52
RP Features	0.56	0.72	0.63
all features	0.59	0.75	0.66

Table 3.2: Results for classifiers trained with Baseline Features, RP Features and both feature sets. Each classifier was trained and tested on all users averaged over ten trials.

User	*Recall*	*Precision*	F_1
Jaeger	0.42	0.55	0.47
Jarvis	0.67	0.77	0.71
Merrick	0.68	0.83	0.74
Tahlia	0.77	0.76	0.77
Average	0.64	0.73	0.67

Table 3.3: Results for 10 randomized trials for classifiers with all features (both RP Features and Baseline Features.) Each classifier was trained and tested on each user. These represent the best classifier for each user, given all features and all user email. Results are averaged over ten trials.

classifier would be tuned to a single user. In this test we discovered that one of our users, Jaeger, did very poorly compared to the other users. Removing his data yielded an average performance of 0.74. This performance is comparable to the inter-annotator score for this task. We were unable to determine the source of poor performance in Jaeger's mail.

Having shown that our features allow for good performance on our task, we now turn to evaluating the ability of our system to do well on a new user. To evaluate transfer, we trained a classifier with Baseline Features on three of the users and evaluated it on the fourth. The results indicate how a prediction system using typical features would perform out of the box. We then repeated this experiment with RP Features. As table 3.4 illustrates, RP Features do very well on the transfer task. The final column shows the difference between the transferred classifiers and the best classifiers F_1 measure on the target user. This indicates the performance lost by using transfer over having a user label email. Typical features (Baseline Features) perform so poorly they are essentially worthless. These features can capture a single user's mail, but when that user's mail is removed from the

Test	*Self*	*Features*	*Recall*	*Precision*	F_1	Δ
Jaeger	0.47	Baseline Features	0.15	0.48	0.23	-0.24
		RP Features	0.56	0.72	0.63	+0.16
Jarvis	0.71	Baseline Features	0.12	0.33	0.18	-0.53
		RP Features	0.52	0.86	0.65	-0.06
Merrick	0.74	Baseline Features	0.01	0.30	0.02	-0.72
		RP Features	0.68	0.73	0.71	-0.03
Tahlia	0.77	Baseline Features	0.28	0.20	0.23	-0.54
		RP Features	0.77	0.59	0.67	-0.10

Table 3.4: A cross-user evaluation for four users averaged over 10 randomized runs. Each classifier is trained on three users and tested on the fourth using either RP Features or Baseline Features. Δ is the F_1 difference between this test and *Self*, the full classifier trained on just the user's data.

training set, as is the case here, the features cannot transfer. In contrast, modeling user behavior through our RP Features yielded a successful transfer. Jaeger's mail, which does worse than the other users (Table 3.3), even performs better with a transferred classifier using RP Features.

3.7 Discussion

RP Features are much better for reply prediction and perform well across users. We now analyze our findings. First, we examined the learned weight vectors of the logistic regression classifiers for a single user to determine the most informative features. Among the top ten positive features for a classifier trained with RP Features were *sender in address book*, *i am in to list*, and *salutation to me*. The top negative features included *does not contain me*, *one recipient*, and *time of day=afternoon*. All of these features apply across users and describe the relation of the user to the message in a general way. In contrast, the top positive baseline features were question words appearing with question marks (why, what, how, etc.), specific senders, and the specific user's email address appearing in the "to" field. Since these features are usually content specific they do not appear in other users' mail and perform poorly after transfer. Similarly, more than half of the top baseline features that were strong negative indicators included a specific person's email address. Given that the best features seem to be social relations (who sent/received the email,) using user specific identifiers would not transfer some of the best features to new users.

While RP Features appear to be shared across users, does this mean that a classifier learned on one user *behaves* like a classifier trained on another? A shared feature space does not necessarily mean that two classifiers will encode the same decision boundary. It is easy to compare two classifiers as follows. Two classifiers are said to be similar if their predictions for a given set of instances are similar. We recorded the predictions of a classifier trained with RP Features with a classifier trained with baseline features. We trained an instance of each classifier on Merrick's training data and another on Jarvis's training data, repeating the test for each of the 10 evaluation splits. A comparison of the predictions between Merrick's and Jarvis's classifiers show the similarity created by RP Features. The two classifiers with baseline features agreed on 71% of Merrick's data and on 70% of Jarvis's. In contrast, the classifiers with RP Features agreed 86% of the time on Merrick's data and 92% on Jarvis's. The high agreement demonstrates that our features produce similar representations for two distinct users.

3.8 Conclusion

We have presented the email reply predictor, an intelligent email interface that predicts when emails require a reply and allows a user to manage her reply queue. Overloaded email users can use reply expectations to make email triage decisions and can quickly find messages that require a reply, avoiding the mistake of forgetting to address messages that expect a response. Our proposed interface addresses each of these use cases by labeling incoming emails with our reply predictor. While many email prediction problems use simple content based features, we found these insufficient for accurately labeling messages. Additionally, because our application of supervised learning requires hand labeled data, we desire features that enable cross-user transfer, allowing a classifier trained on a source user to classify emails for a new target user.

Towards this end we developed a new set of features for reply prediction that draw on lessons from various email learning applications. Our feature functions depend on the email and the user to create a set of shared features extracted in a user-specific manner. These features can be considered deictic since their usage is shared between users but their meaning may be user dependent. We included features that encode social relationships, user behaviors, represent important aspects of message content, indicate temporal information and conjoin useful features to capture second order interactions. Our evaluations show that these features improve over the typical email classification feature set in allowing for accurate reply prediction. Additionally, we have shown that our features allow for good out-of-the-box performance when a trained classifier is applied to a new user with our feature set. This is important as it allows effective cross-user adaptation with the right set of features.

Chapter 4

Analysis of Representations for Learning in Email

4.1 Introduction

As we explored in the previous chapter, providing concise information during triage can help users make better decisions. Previous work on assisting users with email triage has focused on providing users with various types of additional information, including social information [181], short snippets of messages [243] and reply indicators, which we discussed in the previous chapter. While these methods make more information immediately available to the user, they do not provide a summary of message content.

Another approach summarizes email content through automatic machine summarization methods. Several studies have considered email summarization as an effective way of creating a short overview of an email thread [30, 165, 195, 220, 248]. These studies have either extended existing summarization techniques to email, such as centroid based summarization [192], while others have developed new approaches for the structure of email [30,195]. These summaries allow users to summarize discussions, prepare for meetings and see shortened conversations on mobile devices.

These methods require a representation of the content of email for making decisions about which sentences to include in an extractive summary. For example, Wan and McKeown [248] used a centroid based approach that selects sentences in emails that are most similar to an extracted main issue sentence in the first email of a thread. They find the most effective representation for centroid similarity is a vector space model with TF-IDF, which performs better than a Singular Value Decomposition (SVD). Carenini *et al.* [30] develop a new method for selecting important sentences based on a fragment quotation graph of a mail thread. This graph encoded the quotations of previous emails with adjacent fragments (paragraphs) in new emails. Clue Words, words appearing in connected fragments, indicated fragment coherence. The authors were unable to find more effective ways of representing the content of fragments or selecting important words than clue words [31].

While these methods use simple representations of lexical content, other summarization work suggest better representations. Wan *et al.* [247] use thematic information from an SVD to select words for headline generation. Gong and Liu [108] also use an SVD effectively for summarization. It is likely that similar applications may work for email. Muresan *et al.* [176] take this approach by selecting noun phrases from email for summarization. They used a two-stage supervised learning system that selected nouns from individual emails using pre-defined linguistic rules. Human evaluations indicated these phrases to be indicative of message content, which suggests they could be used in an extractive summarization system.

In this chapter, we develop and evaluate a series of representations for the content of emails based on the idea of selecting important words from a message. Following the intuition of Muresan *et al.* we call the resulting list of words a keyword summary of the message. The idea is that these few words may replace the contents of the message for a summarization system. Additionally, while these words may be useful for summarization, they can be useful for a wide variety of learning tasks for email. Therefore, our consideration is for selecting a list of words that describes an email message efficiently for other learning tasks.

While Muresan *et al.* used supervised classification for selecting noun phrases, we prefer unsupervised techniques since they require no annotated training data and can be trained for each user. Our key insight is that a good summary keyword for an email message is not simply a word unique to that message, but a word that relates the message to other topically similar messages. We therefore use latent representations of the underlying topics in each user's inbox to find words that describe each message in the context of existing topics rather than selecting keywords based on a single message in isolation. We compare four methods for selecting email summary keywords, based on two well-known models for inferring latent topics from collections of documents: latent semantic analysis [71] and latent Dirichlet allocation [21].

We favor a list of keywords to efficiently represent each email since the resulting summaries are human readable, which a user may find informative. Additionally, a word list can be used in summarization methods, such as for finding fragment coherence in place of clue words [30]. Furthermore, applications relying on the noun phrases selected by Muresan *et al.* could also use our word lists. Our methodology follows that of Silber and McCoy [221], who evaluate an algorithm for computing lexical chains as an intermediate representation for text summarization. Likewise, we evaluate representations of lexical content as an intermediate representation for summarization and other email learning tasks.

To evaluate our keywords as a representation for learning in email, we conduct two evaluations on email data represented using our method. The two popular email prediction tasks – automatic foldering and recipient prediction – use two different styles of learning systems – multi-class linear classifiers and centroid classifiers – to show how a variety of applications and methods use our representations. Our approaches improve over simpler representations and, in some cases, improve over using the entire email message.

We begin this chapter by discussing what makes a good summary keyword. We then present two methods for selecting keywords: query-document similarity and word association. Each of these methods may be combined with one of two models: latent semantic analysis, and latent Dirichlet allocation. We evaluate the quality of the keywords generated by each method using two email prediction tasks, in which the summaries are used in place of whole messages. The work in this chapter is based on previously published work in the Conference on Intelligent User Interfaces (IUI) [89] and the American National Conference on Artificial Intelligence (AAAI) [88].

4.2 Choosing Good Summary Keywords

The keywords that best represent the content of a message may be different from the keywords used in tasks such as *ad hoc* information retrieval or search. In retrieval tasks, good keywords are those that best distinguish each email from the other messages in a user's inbox. However, these keywords are too specific to be useful for representing the content of a message for summarization and other learning problems. Consider the following example:

> Hi John, Let's meet at 11:15am on Dec 12 to discuss the Enron budget. I sent it to you earlier as budget.xls. See you then, Terry.

The words "11:15am" and "budget.xls" may do a a good job of distinguishing this email from others in John's inbox, but they are too specific to capture the gist of the email and are too obscure. In contrast, "John" and "Enron" may occur in many other messages in John's inbox. This makes them representative of John's inbox as a whole, but too general to provide any useful information regarding this particular message's content. A good summary keyword must strike a middle ground between these two extremes, and be

- specific enough to describe this message but common across many emails,
- associated with coherent user concepts, and
- representative of the gist of the email, thereby allowing informed learning decisions without seeing the entire message.

Our methods address the task of selecting keywords that satisfy all three requirements using latent concept models.

4.3 Latent Concept Models

Latent concept models [21, 109, 118, 157, 245] treat documents as having an underlying latent semantic structure, which may be inferred from word-document co-occurrences. The latent structure provides a low-dimensional representation that relates words to concepts and concepts to documents. In this chapter, we use two widely-used latent concept models to generate email summary keywords: latent semantic analysis (LSA) and latent Dirichlet allocation (LDA). This section provides a brief overview of both methods. For a more detailed description of these techniques, see appendix A.2.

LSA and LDA both represent text corpora such that the distribution of words in each document is expressed as a weighted combination of concepts or topics. In LSA, each concept is a real-valued vector and the the weights are real-valued too; in LDA, the concepts are distributions over words and the weights are mixing probabilities representing distributions over topics. The LSA concepts and weights can be easily computed using singular value decomposition, which often works well in practice. Estimating the LDA concepts and weights is more involved, but the model has the benefit of having a clear probabilistic interpretation that is a better fit to text and that supports many model extensions within the framework of hierarchical Bayesian models. As a result, LDA has been shown to improve over LSA in a wide range of applications [21, 249]. Furthermore, the effective application of LDA to the task of selecting email summary keywords opens the possibility of refining the model to the specific attributes of email [157].

4.3.1 Latent Semantic Analysis

Latent semantic analysis, introduced by Deerwester *et al.* [71], models a text corpus as a word-document co-occurrence matrix X, in which each row corresponds to a word in the vocabulary and each column corresponds to a document. The element X_{wd} indicates the number of times word w occurred in document d. LSA decomposes this matrix into a set of K orthogonal factors, using singular value decomposition. This results in the matrices, U, S and V, whose product approximates the original word-document matrix. For a corpus with D documents and W words in the vocabulary, U will be a $W \times K$ matrix, where each row corresponds to a word in the corpus and each column corresponds to one of the K factors. Similarly, V will be a $D \times K$ matrix, where each row corresponds to a document in the corpus. S will be a $K \times K$ matrix consisting of the K orthogonal factors. While the original word-document matrix typically contains a positive function of word-document co-occurrences, U and V are real-valued and indicate the positive or negative association between each word and document and a particular latent factor. The K factors are thought of as latent concepts in the corpus. Words with similar meanings and usage patterns, such as "canine" and "dog," will be strongly associated with the same latent factors, while dissimilar words will not. The most appropriate number of latent factors K depends on the corpus. Throughout our experiments, we set K to 50. We leave automatic determination of K for future work.

4.3.2 Latent Dirichlet Allocation

Latent Dirichlet allocation provides another way of modeling latent concepts in corpora [21, 109, 225]. In contrast to LSA, which represents words and documents as points in Euclidean space, LDA is a generative probabilistic model that treats each document d as a finite mixture over an underlying set of topics, where each topic t is characterized as a distribution over words. For example, a corpus of newspaper articles might contain latent topics that correspond to concepts such as "politics," "finance," "sports" and "entertainment." Each article has a different distribution over these topics: an article about government spending might give equal probability to the first two topics, while an article about the World Cup might give equal probabilities to the last two. LDA is a generative model: each word w in a document d is assumed to have been generated by first sampling a topic from a document-specific distribution over topics $\theta^{(d)}$, and then sampling a word from the distribution over words that characterizes that topic $\phi^{(t)}$. Furthermore, $\phi^{(t)}$ and $\theta^{(d)}$ are drawn from conjugate Dirichlet priors,

$$\theta^{(d)} \sim \text{Dir}(\alpha) \tag{4.1}$$

$$\phi^{(t)} \sim \text{Dir}(\beta), \tag{4.2}$$

and so $\theta^{(d)}$ and $\phi^{(t)}$ may be integrated out. The probability of w is therefore given by

$$P(w|d, \alpha, \beta) = \sum_{t=1}^{T} P(w|t, \beta) P(t|d, \alpha), \tag{4.3}$$

where T is the number of latent topics. Given a corpus of documents, statistical inference techniques may be used to invert the generative process and infer the latent topics and document-specific topic mixtures for that corpus. We used a Gibbs-EM algorithm to optimize the Dirichlet parameters α and β and infer the latent topics and document-specific topic mixtures. Gibbs-EM alternates between optimizing α and β and sampling a topic assignment for each word in the corpus from the distribution over topics for that word, conditioned on all other variables. The number of topics T, like the number of latent factors in LSA, is corpus-dependent. In all our experiments, we set the number of topics T to 100 and ran the Gibbs-EM algorithm for 500 iterations.

4.4 Generating Summary Keywords

In this section we present two ways to select email summary keywords, one based on query-document similarity, and the other based on word association. Each approach may be used in conjunction with either LSA or LDA. For each email, the pool of candidate keywords is restricted to only those words that actually occur in the email.

4.4.1 Query-Document Similarity

In information retrieval the goal is to retrieve the set of documents that are most relevant to a query. Selecting summary keywords for an email message can be viewed as analogous to this task. Each candidate keyword is treated as a one word query and the similarity between that keyword and the email message is computed.

***LDA-doc*:** When using LDA for information retrieval, the document that is most relevant to a given query is the one that maximizes the conditional probability of the query given the document [26, 249]. Similarly, the candidate keyword c that is most relevant to an email message d is the keyword that maximizes the conditional probability of c given d:

$$P(c|d, \alpha, \beta) = \sum_{t=1}^{T} P(c|t, \beta) P(t|d, \alpha), \tag{4.4}$$

where $P(c|t, \beta)$ and $P(t|d, \alpha)$ are posterior distributions obtained from all the emails in the user's inbox (including this one) up to this point in time and a set of corresponding topic assignments from a single Gibbs sample. The candidate keywords with the highest probability are those that are highly probable in the most probable topics for this document.

***LSA-doc*:** The similarity between a candidate keyword c and an email message d can be computed using LSA by taking the dot product of U_c and V_d, where U_c is the c^{th} row of the matrix U and V_d is the d^{th} row of the matrix V [71, 92]. As was the case with the LDA variant described above, candidate keywords that have similar concept membership to the email message will a receive a higher score.

4.4.2 Word Association

Word association scores pairs of words based on their association with each other [225]. A second approach to selecting summary keywords for an email message involves choosing as keywords those words that are most closely associated with the words that occur in the message.

***LDA-word*:** The degree to which a given word c is associated with another word w can be determined by treating w as a cue word and computing the conditional probability $P(c|w)$ that c is generated as a response to cue word w. This probability, under LDA, is given by:

$$P(c|w,\alpha,\beta) = \sum_{t=1}^{T} P(c|t,\beta)P(t|w,\alpha,\beta). \tag{4.5}$$

Candidate word c will have a high probability if it has a high probability in the topics that are most likely according to the posterior distribution over topics given w.

This similarity metric can be extended to measure the extent to which a word c is similar to an entire document, by computing the product of equation 4.5 over all the words in the document. Note that the words in the document are treated as a set: each word occurs only once in the product, regardless of the number of times it occurred in the document:

$$\begin{aligned} P(c|d,\alpha,\beta) &= \prod_{w\in d} P(c|w,\alpha,\beta) \\ &= \prod_{w\in d}\sum_{t=1}^{T} P(c|t,\beta)P(t|w,d,\alpha,\beta) \\ &\propto \prod_{w\in d}\sum_{t=1}^{T} P(c|t,\beta)P(w|t,\beta)P(t|d,\alpha). \end{aligned} \tag{4.6}$$

The proportionality in the last line is obtained using Bayes' rule: $P(A|B) = \frac{P(B|A)P(A)}{P(B)} \propto P(B|A)P(A)$.

***LSA-word*:** A similar technique may be used in conjunction with LSA. The product of the three probabilities in equation 4.6 is a measure of the similarity between two words c and w in document d under topic t, weighted by the probability of topic t in the document in question. In LSA, the similarity between words c and w may be computed by taking the dot product of the vectors that represent these words in the latent space, weighted by the strength of each factor for that document. The association between candidate keyword c and document d is computed by computing the sum of this quantity over all words w in the document:

$$\text{assoc}(c,d) = \sum_{w\in d}\sum_{k=1}^{K} U_{ck}U_{wk}V_{dk}. \tag{4.7}$$

The sum over words w in document d, which combines positive and negative association scores, is the LSA counterpart of the product of probabilities for LDA in equation 4.6.

4.5 Evaluation

SUBJECT: ASE Hypertiles from Final Report Out
Sally -
Attached are the hypertiles from the final report out at yesterday’s ASE Studio Workshop. The CD is finished and on its way to Houston. The files are organized by team:
Hammer - Sales and Marketing, Vision Stmt, Mission Stmt, Target Market, How to Approach, Pricing, SLA
Pliers - Producst and Services - Consulting Based
Saw - Infrastructure Transition Plan
Wrench - Producst and Services - Basic Outsourcing
I hope these help with your meeting tomorrow. Let me know if there is anything else I can do to help.
Lisa P

TF-IDF	*LSA-doc*	*LSA-word*	*LDA-doc*	*LDA-word*
producst	meeting	meeting	team	team
pliers	team	market	meeting	meeting
stmt	services	houston	services	services
hammer	houston	report	lisa	lisa
wrench	report	team	ase	ase
hypertiles	market	final	attached	attached
sla	tomorrow	transition	report	report
studio	final	yesterday	studio	studio
mission	plan	tomorrow	outsourcing	outsourcing

Figure 4.1: An email from the Enron corpus (*beck-s/ase/4*) and the summaries produced by the LDA- and LSA-based methods and a TF-IDF baseline. TF-IDF selects words that are overly specific to this message, including a misspelled word. The methods based on latent concept models select more general words that better capture the gist of the email. For example, *LDA-doc* and *LDA-word* both generate “team,” “meeting,” “ase” and “report.”

The keyword generation methods described in the previous section—*LDA-doc*, *LSA-doc*, *LDA-word*, *LSA-word*—were run on selected users from the Enron corpus [135]. Prior to generating keywords, we removed common stop words and email-specific words, such as “cc,” “to” and “http.”

Different applications will favor longer or shorter word lists. We included nine words per summary since this seemed like a balance between short lists and selecting a sufficient number of words. If a message contained fewer than nine keywords with non-zero scores, a shorter summary was used. An example email and the corresponding summaries can be seen in figure 4.1.

Term frequency-inverse document frequency (TF-IDF) was used as a baseline against which *LDA-doc*, *LSA-doc*, *LDA-word* and *LSA-word* were compared. TF-IDF is a statistical technique for evaluating the importance of a word to a document. Words that occur rarely in a corpus, but often in a document will be ranked as being very important to that document. To generate summary keywords using TF-IDF, the nine highest scoring words for each email, according to TF-IDF, were selected. For completeness, the entire message body was used as an upper baseline.

To evaluate the quality of the representation we selected two email prediction tasks: automated foldering and recipient prediction. These tasks were chosen because they are well-defined, have previously been applied to the Enron corpus and usually rely on the entire message body for making predictions. They are also typically tackled using different learning methods, allowing for a diverse evaluation. For each keyword generation method, both tasks were carried out on the Enron corpus, with the message bodies replaced by the generated summaries. Predictions were therefore made using the summary keywords only.

4.5.1 Automated Foldering

Many email users archive and organize their email messages into folders. Automated foldering is the task of automatically predicting the appropriate folder for a given email message. One of the first such applications of this learning task to a real email client was by Segal and Kephart [214]. Many others have evaluated different learning algorithms for this task, such as rule induction and k-nearest neighbors [187], RIPPER (a rule induction classifier) [49], naive Bayes [191, 199] and SVMs [132]. Bekkerman *et al.* [10] conducted a comparison of numerous techniques on the Enron corpus. Since the messages in each folder are typically related by one or more common topics, automated foldering is a good

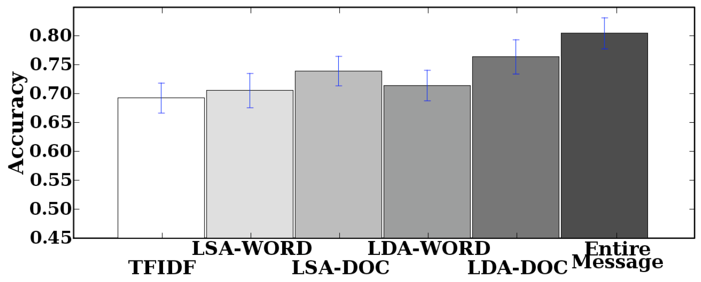
User	*Messages*	*Total Messages*
beck-s	751	10168
farmer-d	3020	11395
kaminski-v	3172	25769
kitchen-l	2345	4691
lokay-m	1966	4299
sanders-r	863	5956
williams-w3	2542	3164

Table 4.1: The number of messages in the ten largest folders for each of the seven Enron users selected for the automated foldering task, as well as the total number of messages for each user.

task for evaluating summary keyword generation methods based on latent concept models.

Typical representations for email foldering use the words of the email directly. Segal and Kephart [214] use a modified TF-IDF representation for a TF-IDF style classifier. Bekkerman *et al.* [10] evaluate a wide variety of classifiers on a bag-of-words feature set for email. Fink *et al.* [98] also use words based on features, but their features have an added temporal element. For each of these methods, we can replace the words of the message with our selected keywords.

We used a learning system similar to that of Fink *et al.* [98] to evaluate the LDA- and LSA-based keyword generation methods. Each email was represented by a binary vector, where each position in the vector corresponds to a summary keyword. These vectors were used as input to a multi-class classifier with one class for each folder. The keyword generation methods were evaluated on the seven Enron users selected by Fink *et al.*, except that prediction was only run on the ten largest folders (excluding non-archival email folders, such as “inbox,” “deleted items” and “discussion threads”) for each user. This was done to compensate for the use of simpler features than Fink *et al.* Even though only a subset of the messages were used for evaluation, the LDA and LSA models for each user were trained on all messages in the user’s mailbox. Table 4.1 indicates the number of messages for each user.



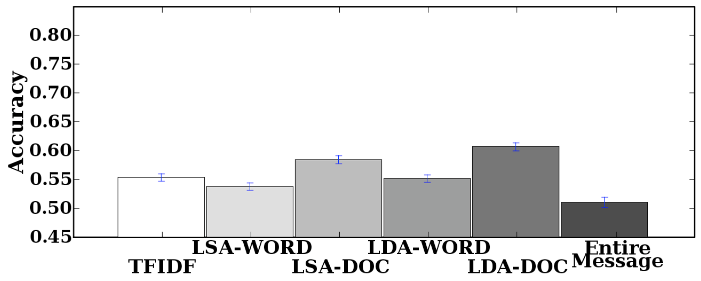


Figure 4.2: Automated foldering results using batch (top) and online (bottom) learning, averaged across all seven Enron users. Each graph shows the accuracy achieved on the foldering task using keywords generated by TF-IDF and the four and LSA-based methods, as well as the entire message. Error bars indicate standard deviation across each test.

The generation systems were evaluated using both online and batch learning algorithms. Online learning algorithms resemble a real-world setting: the algorithm receives a message, predicts a folder, and is then told which folder was in fact correct. All emails were processed in this fashion and the total classification accuracy was computed. For each user and generation method, ten trials were conducted on randomly shuffled data. MIRA [55, 159], a passive-aggressive variant of the perceptron algorithm for large margin online classification, was used to perform the online classification. Batch learning algorithms process all training messages prior to making any predictions. A maximum entropy classifier [155], along with ten-fold cross validation, was used for batch classification. For more information on Maximum Entropy learning and MIRA, see appendices A.1.2 and A.1.4 respectively.

Figure 4.2 shows the classification accuracy on the automated foldering task using both batch and online learning algorithms, averaged over all users. The greater the accuracy, the better the foldering performance. The generation methods based on LDA and LSA all outperform TF-IDF in the batch setting, while all methods except for *LSA-word* outperform TF-IDF in the online setting. In general, batch performance exceeded online performance, which is unsurprising since online learning was run for a single iteration only.[1] Additionally, in the online setting the LDA- and LSA-based methods outperform using the entire message body, because the single-pass online learner can overfit when many words are involved. In contrast, using the entire message body does better in the batch setting. The differences between TF-IDF and the four LDA- and LSA-based generation methods when evaluated in the batch setting are statistically significant at $p = 0.05$ using McNemar's test measured on the aggregate predictions of each method across seven users. [2]

We also considered another style of classification as a baseline for our methods. TF-IDF representations for foldering are challenging since it favors words that are not repeated in other emails. However, we can extend the intuition behind TF-IDF to the classification algorithm. A TF-IDF classifier measures the words that are most informative about a folder according to a TF-IDF score for the folder computed from training data. Given a set of possible folders $\mathcal{F}$ and documents indexed by d with folder label y_d, the score for a given word n and folder $f \in \mathcal{F}$ is:

$$\text{TF}(n, f) = \frac{\sum_{(d|y_d=f)} M_{n,d}}{\sum_{(d|y_d=f)} \sum_{n' \in d} M_{n',d}} \tag{4.8}$$

$$\text{DF}(n) = \frac{\sum_{f \in F} \text{I}(n, f)}{|\mathcal{F}|} \tag{4.9}$$

$$\text{score}(n, f) = \frac{\text{TF}(n, f)}{\text{DF}(n)} \ , \tag{4.10}$$

where $M_{n,d}$ is the number of times word n appears in document d and $\text{I}(n, f)$ is an indicator function that is 1 iff word n appears in a training document labeled with folder f. The

[1]When online algorithms are run in a batch setting, it is common to run them for multiple iterations over the data.

[2]Measuring significance in the online setting is complicated because multiple predictions are issued for each message across the ten randomized runs.

classifier selects the best scoring folder f for a document d as:

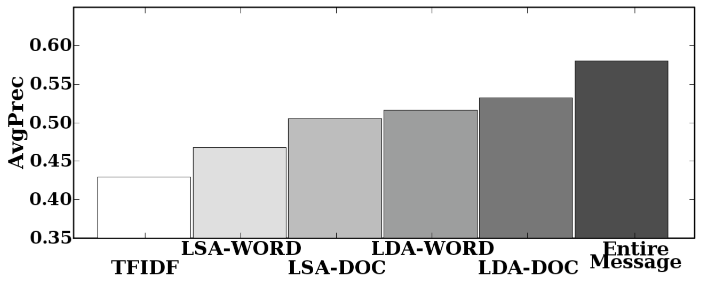
$$\arg\max_{f \in \mathcal{F}} \text{score}(d, f) = \sum_{n \in d} \text{score}(n, f) \tag{4.11}$$

We compared this TF-IDF classifier evaluated on the entire message to one evaluated on our representations. This classifier achieved an average accuracy of 62.51% when run on the entire message. This compares poorly with the results of batch learning with the best method (*LDA-doc*), 75.47%. Additionally, we can run this TF-IDF classifier on our representation for each message in the folder instead of all of the message's contents. Run on *LDA-doc* the TF-IDF classifier achieves 60.28%, about 2% worse than using the entire message. Selecting the most discriminative words for classification by TF-IDF from the entire message gives only a small improvement over using our representation.

In both the online and batch settings, the methods based on query-document similarity outperform those based on word-association. Furthermore, using LDA consistently results in improvements over the results obtained using LSA. *LDA-doc* therefore achieves the best performance. Using the TF-IDF classifier, the entire message improves slightly over *LDA-doc*. Interestingly, in the batch setting, the accuracy obtained using *LDA-doc* comes close to the accuracy obtained using the entire message body. This indicates that the summary keywords generated by this method are a good approximation of the complete message content in the context of foldering.

4.5.2 Recipient Prediction

In large organizations, it is typical for multiple people to be involved in projects. This makes it easy to forget to include one or more recipients on a project-related email. Recipient prediction systems aim to prevent this by suggesting possible recipients to the user during message composition. Previous work has explored several learning methods for constructing recipient prediction systems [36, 185]. These systems are trained on previous messages sent by the user in order to determine associations between the words in each



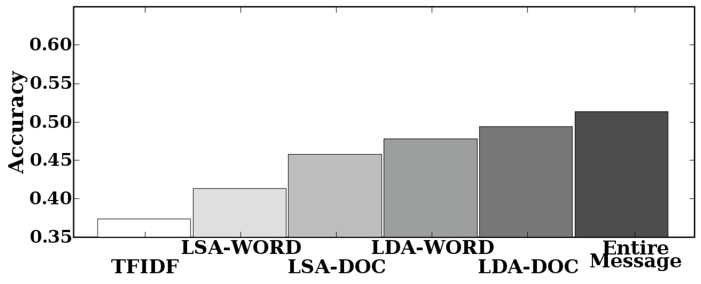


Figure 4.3: Results for the recipient prediction task showing the average precision (top) and accuracy (bottom) averaged across all seven Enron users, for TF-IDF and the four LDA- and LSA-based methods, as well as the entire message.

email and the message's recipients. When given a new message, a ranked list of potential recipients, based on the email's content, is presented to the user. Recipient prediction systems are evaluated by measuring the degree of agreement between the suggested and correct recipients.

Recipient prediction serves as a good proxy task for evaluating summary keywords generated using latent concept models. In our experiments, we use Carvalho and Cohen's system [36]. [3] This system employs both content similarity and statistical learning techniques. Training emails are labeled with their recipients and a K-nearest neighbor classifier is constructed. Given a new email, a list of potential recipients is constructed by voting among the closest messages.

As with the automated foldering task, the text of every email message was replaced with summary keywords. The generation methods were evaluated on the seven Enron

[3]Thanks to Vitor Carvalho for providing access to and assistance with this system.

User	*Sent Messages*	*Total Messages*
geaccone-t	310	1352
germany-c	2692	9581
hyatt-k	400	1532
kaminski-v	1084	25769
kitchen-l	980	4691
white-s	432	2657
whitt-m	273	702

Table 4.2: The number of sent messages for each of the seven Enron users selected for the recipient prediction task, as well as the total number of messages for each user.

users used by Carvalho and Cohen. The list of users and the size of their sent mail folders is shown in table 4.2. For each user, sent messages were ordered chronologically. The last fifty messages were reserved for testing and the system was trained on the remaining messages. The system was run on the summaries generated by TF-IDF and each of the LDA- and LSA-based methods, as well as the full text of the messages.

Average precision and accuracy results, averaged across all seven users, are shown in figure 4.3. For both of these evaluation metrics, all four LDA- and LSA-based generation methods outperform the TF-IDF baseline. The improvements of *LSA-doc*, *LDA-doc* and *LDA-word* over the TF-IDF baseline and *LSA-word* are statistically significant at $p = 0.05$ using the Wilcoxon signed rank test. Furthermore, the generation methods based on LDA outperform those based on LSA and the methods that use query-document techniques beat those that are based on word association. Overall, the best generation method is *LDA-doc*, which performs statistically significantly better than *LSA-doc* and achieves accuracy comparable to using that obtained using the entire message.

4.6 Discussion

The results obtained on the automated foldering and recipient prediction tasks demonstrate that summary keywords generated using latent concept models do indeed serve as a good

Method	*Unique Words*	*Total Usage*
TF-IDF	10896	36.49
Entire Message	18213	29.16
LSA-doc	1793	205.33
LSA-word	3346	120.96
LDA-doc	2059	183.38
LDA-word	3301	123.88

Table 4.3: The average number of unique words in the summaries generated by each generation method, as well as the average number of times each keyword appears in the entire mailbox. Each summary contains a maximum of nine words. Results are averaged over the twelve Enron users used for the automated foldering and recipient prediction tasks.

approximation of message content. In almost all cases, the LDA- and LSA-based methods outperformed TF-IDF, and in many cases the LDA-based methods achieved performance close to that obtained using the entire message. We now explore additional properties of the generated keywords.

One of the requirements for summary keywords was that they be descriptive of the message and neither too general nor too specific. The extent to which keywords generated by the LDA- and LSA-based systems satisfy this requirement can be determined by analyzing frequency information for the keywords. The vocabulary size of the set of all summaries for a given method provides an indication of the specificity of the words in those summaries: a larger vocabulary indicates that each word is used fewer times, that is, the words are more specific. Table 4.3 lists the vocabulary size and average number of occurrences of each word for each method. The summaries for all four of the LDA- and LSA-based methods have much smaller vocabulary sizes than those generated using TF-IDF. This, combined with the fact that the TF-IDF results were typically much worse than those of the other methods, indicates that TF-IDF is selecting keywords that are too specific, while the methods based on concept models are selecting more general keywords that better relate to common words in the users' topics.

In addition to evaluating summary keywords as an approximation to message content,

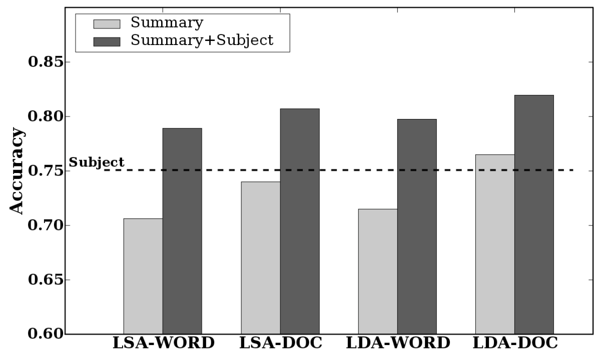


Figure 4.4: The accuracy, averaged across seven Enron users, for the automated foldering task using batch evaluation for the message subject, the summaries generated by the the LDA- and LSA-based methods, and the summaries combined with the subject.

we can also compare their value with other existing indicators of message content. Specifically, every message has a subject line that is written by a user to describe the contents of the message. To compare the subject line with our summaries, two additional experiments involving the automated foldering and recipient prediction tasks were carried out: one with the entire message replaced by the subject line and the other with the message contents replaced by both the summary keywords and email subject. Figures 4.4 and 4.5 show the results obtained in these experiments. The result obtained using just the email subjects is shown as a dashed line. On the automated foldering task, only *LDA-doc* achieved better performance than this. However, when combined with the subject, all the LDA- and LSA-based methods had significantly better results than those obtained using the subject alone. These results are statistically significant at $p = 0.05$ for *LSA-doc*, *LSA-word* and *LDA-word* and at $p = 0.01$ for *LDA-doc* using McNemar's test. On the recipient prediction task all four methods based on latent concept models give performance improvements over using only the subject. The improvements obtained using the LSA-based methods are significant at $p = 0.05$ using the Wilcoxon signed rank test, while the those obtained using the LDA-based methods are significant at $p = 0.001$.

These results indicate that summary keywords generated using LDA- and LSA-based methods do indeed provide a good representation of email content. Furthermore, these

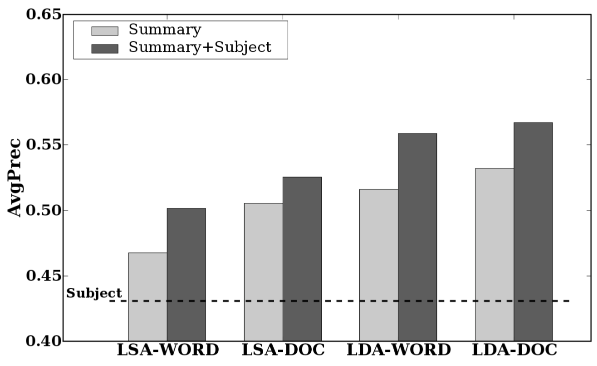


Figure 4.5: Average precision for the recipient prediction using only the message subject, the summaries generated by the LDA- and LSA-based methods, and the combination of the two.

keywords do better at summarizing message content for foldering and recipient prediction tasks than sender-written subject lines. Combining summary keywords with the email subject significantly increases the quantity of useful information.

4.7 Conclusion

Many email learning tasks use a representation of the lexical content of a message. In this chapter, we explored selecting a few words from the message to replace the entire message. Such keyword lists can potentially be used in automatic summarization methods, which sometimes rely on identifying important words, or other email learning tasks. In this chapter, we explored two such tasks: automatic foldering and recipient prediction. Of the four methods we considered, the keywords generated using an approach based on LDA and query-document similarity concepts are consistently better than those generated using other methods addressed in this chapter. Additionally, summary keywords generated using the LDA- and LSA-based methods presented in this chapter were shown to provide additional information over email subject lines, and are therefore most effective when used in conjunction with email subjects. Given these results, it is likely that more advanced topic models for email may yield better representations. For example, summarization typically operates on the thread level instead of on individual messages. A topic model could incorporate thread structures to improve topic estimation. We leave the investigation of such techniques and integration to summarization systems as future work.

Chapter 5

Email Activity Management

5.1 Introduction

So far we have explored tools for supporting common email behaviors, message composition and triage. These are universal actions supported by the existing email paradigm: a communications tool. As email usage increases in the workplace, its role is changing to encompass other tasks, such as activity and project management. In this chapter we will describe an intelligent system for supporting email activity management. The work in this chapter is based on previously published work in the Conference on Intelligent User Interfaces (IUI) [86] and the American National Conference on Artificial Intelligence (AAAI) [139].

The goal of activity-centric research is to provide tools for people to manage their activities more effectively. We view activities as a representation of collaborative work practice. Activities involve a set of people, who may each play different roles in the activity (e.g., the coordinator vs. participants). Activities often have state (e.g., completed vs. in-progress). They typically have a goal (e.g., producing a report or making a decision), and they might contain formal or informal processes for reaching that goal. Activities may be related to other activities in ways such as composition (one activity being a part of another) or dependency (one activity must be completed before another can proceed).

Examples of activities include: organizing a conference, reviewing papers, purchasing equipment, managing candidate interviews, and making design decisions, for example.

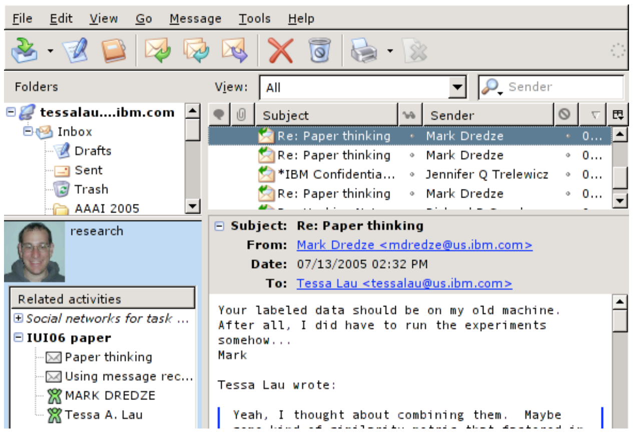
Several activity-centric systems have emerged recently [53, 78, 172, 174]. Most of these systems have focused on the cross-application nature of activities, and provide a means to gather together an activity's documents, emails, people, chat transcripts, and related information into a single view, so that when working on the activity, all of the artifacts related to that activity are near at hand.

At the same time, email has become the primary communications mechanism for people to coordinate collaborative work [90, 251], particularly as workplaces are distributed across multiple geographies. Yet despite email's importance, none of the activity systems to date provide a compelling story for integrating email with activity management. Our goal in this research is to provide better tools for people to manage activities more effectively, focusing on email as the medium people use to communicate about activities.

In this chapter, we focus on the problem of classifying emails into activities, in order to automatically populate activities with the emails related to them. Our approach leverages two characteristics of activities: the observation that activities connect groups of people together and the observation that activity-related email tends to center around particular topics. In addition, we have found that a combined approach, which votes together the predictions of the base models, can perform better than each individual model alone, resulting in 93% overall accuracy and an F_3 of 0.81 on a dataset of 1146 emails.

Specifically, we make the following contributions:

- The SimSubset and SimOverlap algorithms for email activity classification that compare the people involved in an activity against the recipients of a message;
- The SimContent algorithm for email activity classification based on content similarity using iterative residual rescaling [2], a form of latent semantic indexing [71];
- An empirical evaluation of our algorithms showing that each one performs better than baseline approaches of either message threads or a naïve Bayes classifier; and



Subject	Sender
Re: Paper thinking	Mark Dredze
Re: Paper thinking	Mark Dredze
*IBM Confidentia...	Jennifer Q Trelewicz
Re: Paper thinking	Mark Dredze

Subject: Re: Paper thinking
From: Mark Dredze <mdredze@us.ibm.com>
Date: 07/13/2005 02:32 PM
To: Tessa Lau <tessalau@us.ibm.com>

```
Your labeled data should be on my old machine.
After all, I did have to run the experiments
somehow...
Mark

Tessa Lau wrote:

  Yeah, I thought about combining them.  Maybe
```

Figure 5.1: Email Activity Management (EAM) Thunderbird extension displaying the contextual activity pane in the lower left.

- Empirical evidence showing that a simple voting method for combining the results of the individual algorithms performs better than each algorithm alone.

5.2 Email-based activity management

In order to motivate our work on email activity classification, in this section we describe several different user interfaces we have prototyped to illustrate the benefit of integrating email with activities.

5.2.1 Activity pane

Figure 5.1 shows an extension to the Thunderbird email client that displays an activites pane in the lower-left corner. Each activity in this display can contain a set of email messages and a set of people involved in the activity; in the future, this representation could incorporate documents (such as mail attachments), instant messaging transcripts, and other artifacts that may be related to an activity.

The pane shows all the activities that involve both the current user and the sender of the currently-selected email message. The activity list is prioritized by relevance, using the SimOverlap metric described later in this chapter; activities whose members are most similar to the set of recipients in the message are displayed at top of the list.

A new activity can be created by right-clicking on an email message. The user is prompted to enter the name and description of the activity, and the set of people involved in the activity; the name and people are automatically extracted from the subject and recipients of the target email. New messages can be associated with an activity by dragging and dropping them onto the activity pane.

Activities are published to a central server, and every participant in an activity will see the activity displayed in her activity pane when relevant. Shared activities are a collaborative artifact: any changes to an activity are made visible to all members of the activity. Thus all participants share responsibility for updating the activity, and all members benefit from the organizational work of other members. We have implemented this interface in Thunderbird, and it is currently in use by a handful of researchers.

While the contextual activity pane automatically displays all activities that may be relevant to a particular selected message, the system does not automatically associate messages with activities. Currently this is only accomplished manually though drag-and-drop. In order to build up a more complete view of the email exchanges that contribute to an activity, users must collectively expend a lot of effort to manually associate messages with activities.

We address this problem of automatically classifying messages into activities. The algorithms we have developed here could be used to extend this prototype to automatically populate activities with relevant messages.

5.2.2 Activity inbox and Activity manager

One of our future research goals is given a set of messages in an activity, automatically extract the structure of that activity. For example, an activity might include a formalized process, so the structure could include the steps in the process as well as the user's current

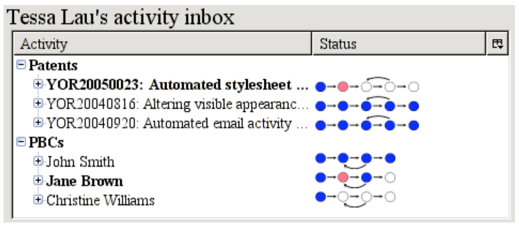
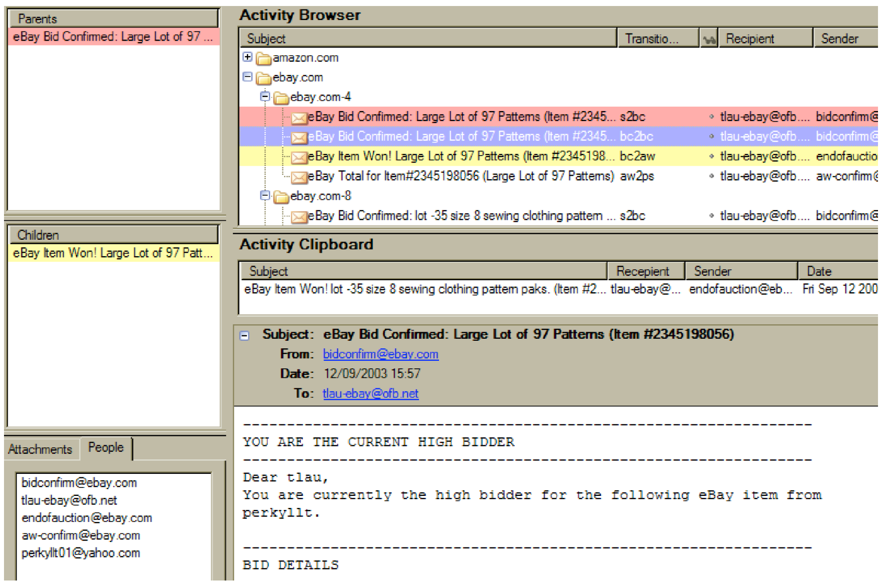

Tessa Lau's activity inbox
Activity
Status
Patents
YOR20050023: Automated stylesheet ...
YOR20040816: Altering visible appearanc...
YOR20040920: Automated email activity ...
PBCs
John Smith
Jane Brown
Christine Williams

Figure 5.2: An activity inbox displaying six instances of two activities, along with the process structure of each (bubble diagrams) and the current state of each (light grey dots). Highlighted messages indicate "unread" progress through the activity.

state in completing that process. Alternatively, an activity might have a submission deadline, with messages leading up to that deadline acquiring more importance. The vision is to provide activity-centric tools rather than message-centric tools, and let users manage activities as a whole, rather than individual mail messages.

Figure 5.2 shows a conceptual UI for email-based activity management. Suppose that instead of checking one's email inbox for new messages, one could check one's activity inbox to see what has happened on each activity since the last time it was checked. This interface groups messages together by activity, and shows that (for example) there is action on one of the "patent" activities and one of the "PBC" activities. The colored-dot diagrams show a finite-state-machine representation of the structure of each activity; the light grey dot shows where the new message occurred, within the context of the rest of the activity.

Figure 5.3 shows a third activity-centric email client. It models activities as instances of finite-state *process models* in which messages correspond to transitions between states [138]. This model is general enough to handle both structured computer-human activities (e.g., e-commerce transactions) and informal human-human activities (e.g., meeting scheduling, collaborative document editing). This client seemlessly integrates both ends of this spectrum: the algorithm of Kushmerick and Lau [138] is used to automatically label structured activities ("message 1 is an order confirmation", "message 2 is a delay notification"), and the algorithm of Khoussainov and Kushmerick [130] is used to label informal activities with speech acts [50] ("message 1 is a meeting request", "message 2 is a meeting



Parents

eBay Bid Confirmed: Large Lot of 97 ...

Children

eBay Item Won! Large Lot of 97 Patt...

bidconfirm@ebay.com
tlau-ebay@ofb.net
endofauction@ebay.com
aw-confirm@ebay.com
perkyllt01@yahoo.com

Activity Browser

Subject	Transitio...	Recipient	Sender
amazon.com			
ebay.com			
ebay.com-4			
eBay Bid Confirmed: Large Lot of 97 Patterns (Item #2345...	s2bc	tlau-ebay@ofb....	bidconfirm@
eBay Bid Confirmed: Large Lot of 97 Patterns (Item #2345...	bc2bc	tlau-ebay@ofb....	bidconfirm@
eBay Item Won! Large Lot of 97 Patterns (Item #2345198...	bc2aw	tlau-ebay@ofb....	endofauctio
eBay Total for Item#2345198056 (Large Lot of 97 Patterns)	aw2ps	tlau-ebay@ofb....	aw-confirm@
ebay.com-8			
eBay Bid Confirmed: lot -35 size 8 sewing clothing pattern ...	s2bc	tlau-ebay@ofb....	bidconfirm@

Activity Clipboard

Subject	Recepient	Sender	Date
eBay Item Won! lot -35 size 8 sewing clothing pattern paks. (Item #2...	tlau-ebay@...	endofauction@eb...	Fri Sep 12 200

Subject: eBay Bid Confirmed: Large Lot of 97 Patterns (Item #2345198056)
From: bidconfirm@ebay.com
Date: 12/09/2003 15:57
To: tlau-ebay@ofb.net

```
-----------------------------------------------------------------
YOU ARE THE CURRENT HIGH BIDDER
-----------------------------------------------------------------
Dear tlau,
You are currently the high bidder for the following eBay item from
perkyllt.

-----------------------------------------------------------------
BID DETAILS
```

Figure 5.3: Activity manager with a list of activities and associated emails. The corner pane shows people and attachments related to the current activity.

confirmation and a commitment to send an updated document").

As these examples show, our larger goal is to infer structure from the unstructured email that make up these activities, and to build tools that make this structure accessible to users to help them manage their activities more effectively. As manually categorizing messages is tedious and infeasible given the volumes of email users receive, automated email activity classification is the first step towards this goal.

5.3 Email activity classification

All of the UIs described in the previous section would benefit from a system that can identify, given a new message, which activity it belongs to. Thus, we define the *email activity classification* problem as follows:

Given a set of existing activities $\mathcal{A} = A_1, A_2, ..., A_N$, the null activity ϵ, and a message M, output a probability distribution P over activities such that $\sum_{i \in \mathcal{A} \cup \{\epsilon\}} P(i|M) = 1$.

Note that this is an incremental learning problem — the set of class labels can (and will) change over time as new activities are created by the user. However, at a particular point in time, the classifier is only expected to be able to predict activities that have previously been created. The intent is that ϵ is a distinguished activity label meaning the message is not associated with any activity.

5.3.1 Experimental methodology

Our evaluation was designed to simulate a single user's experience with an activity management system. We assume that messages are received sequentially in temporal order. According to our model of system usage, when a user sees the first message that defines a new activity, she creates a new activity and associates that message with the activity. The user will then expect the system to correctly identify and associate any further activity-related messages. This model leads us to an incremental online learning evaluation.

We evaluate our system using a corpus of 1146 email messages gathered from one user's email account, of which 149 were labeled with the activity to which they belonged.[1] The corpus spanned a period of 90 days. Since activities were manually labeled, we assume that the labels are correct but not complete; that is, messages that have an activity label have the correct label, but not all messages that belong to an activity may have been labelled as such. A total of 27 distinct activities were identified, ranging in size from one to 38 messages. Examples of activities included organizing an event at a conference, reviewing papers for a workshop, planning a visit to a university, brainstorming about new features to include in an upcoming product, ordering ergonomic equipment, and interviewing a job candidate.

We designed the following experimental methodology. First we sort the messages in temporal order, simulating the order in which the user sees them. We assume that messages not marked as belonging to any activity are labelled with an explicit "no activity" label ϵ (i.e., the null activity). For each message, the system predicts which activity this message

[1]While in the general case messages may belong to multiple activities, we leave that as a topic for future work.

Correct label	**Predicted label** No activity	Activity A	Activity B
No activity	**(A) Correct**	(B) False positive	
Activity A	(C) False negative	**(D) Correct**	(E) Incorrect

Table 5.1: Confusion matrix showing possible predicted outcomes.

belongs to, including the null activity. We then incrementally train the classifier with this message, and repeat on each remaining message. One exception to this rule is the first labelled message of every activity; it would be impossible for the system to correctly predict the label on these messages because it has never seen an instance of that activity before. We expect the user to manually identify each new activity as it starts. So in our experiment, we detect the first message of each activity, assume that the user has intentionally labelled it, and do not test on this message.

Depending on how the classifier will be employed in an actual system, the straightforward approach of measuring accuracy does not always represent the quality of a user's experience with the system. Table 5.1 shows a confusion matrix that categorizes different errors the system can make.

If the message is not part of an activity, our system can either label the message as no-activity, or as belonging to some activity. The former is correct, the latter is incorrect. If the message actually belongs to activity A, our system may either incorrectly predict no-activity, correctly predict activity A, or incorrectly predict activity B.

For instance, a recommendation-style UI might present a ranked list of the top 3 predicted activities given a message, and have the user explicitly select one of them. (This is the UI paradigm used in SwiftFile [213].) In this type of UI, the number of false positives (cell B) is largely irrelevant because if the system predicts an activity for a message that the user does not want to associate with an activity, she can ignore the prediction with no harm. On the other hand, false negative errors (cell C) are much more expensive for the user to recover from, because it means she cannot simply associate the message with one of system's predicted activities, but must instead browse the complete list of activities in order to locate the correct one.

The *recommendation accuracy* of a system measures how useful it is at recommending the correct activity from a list of N activities. We define the recommendation accuracy of a system $Acc_{rec,N} = \frac{D}{D+E}$ in terms of Table 5.1, where N is the number of recommended activities. For example, $Acc_{rec,1}$ is the accuracy when the system only recommends a single activity, and $Acc_{rec,3}$ is the accuracy of a system that recommends three activities and is correct when the correct activity is in the set of three recommendations.

Alternatively, in a UI such as Figure 5.1 where messages could be automatically added to activities, errors in box (B) might be less important than errors in box (C), under the assumption that a user could easily remove extraneous messages from an activity, but would find it difficult to locate "lost" messages that are not associated with any activity.

In a more extreme case, if a user comes to depend on an activity inbox view (Figure 5.2) to be notified of new happenings on his activities, then failing to categorize an incoming message as part of one of those activities could be disastrous. On the other hand, miscategorizing a message as belonging to one activity when it actually belongs to another seems to be less critical, because once a user has been notified of the incoming message, he can correct the system's classification to place it in the correct activity.

The *recognition accuracy* of a system, Acc_{rcg}, measures how useful it is at classifying messages into activities. It depends on the system's performance on both messages that do belong to an activity, and messages that are not part of an activity. We define recognition accuracy as $Acc_{rcg} = \frac{A+D}{A+B+C+D+E}$.

However, the accuracy does not necessarily reflect the true usefulness of a system. We also measure the precision $Precision = \frac{D}{B+D+E}$ and the recall $Recall = \frac{D}{C+D+E}$ as well as the F-measure $F = \frac{(\beta^2+1)*Prec*Rec}{\beta^2*Prec+Rec}$ which is a combined precision/recall number that decreases the weight of no-activity examples and emphasizes the system's performance on positive examples based on the parameter β. In our experiments we set $\beta = 3$. We believe that F_3 is a more representative measure of the quality of a user's experience with a recognition system since it favors activity completeness over accuracy. This reflects the higher user cost of missing activity messages rather than viewing several extra messages.

We use two baselines for evaluation of our methods. The first is the use of message threads to identify activity membership. For each message marked as being part of an activity, all subsequent replies or follow-ups to that message are also labeled as belonging to that activity. This is the method used, for example, in Bellotti's Taskmaster system [12, 13]. We also compare against an updatable naïve Bayes classifier, used by many prior systems for email classification.

In order to verify that our results are not specific to the particular ordering of messages in our corpus, in several of our experiments we report average results over a number of runs with randomized message ordering. There may be content within the sequence of messages in an activity that is time-dependent. Therefore, we randomized the order in which activities occurred in the dataset, while preserving the original ordering of the messages within an activity.

5.4 SimOverlap and SimSubset

Our first algorithm for email activity classification takes advantage of the observation that activities tend to involve specific people with specific roles, which is manifested in the fact that messages are exchanged between different people during the activity. For a message M, let $ppl(M)$ denote the set of all people (i.e., distinct email addresses) in the To, From, and CC fields of M. We generalize ppl from messages to activities by defining $ppl(A) = \bigcup_{M \in A} ppl(M)$.

Given this model, we can define the similarity between an activity A and a message M. We have experimented with several different similarity metrics. The first is symmetric, based on the overlap between the people in the activity and the people in the message:

$$SimOverlap(A, M) = \frac{2|ppl(A) \cap ppl(M)|}{|ppl(A)| + |ppl(M)|}$$

This metric is 0 when there is no overlap between the people in the activity and the message, and is 1 when the set of people in the activity is identical to the people in the message. Since the user who received the message is (by definition) a recipient of every message and also belongs to every activity under consideration, we remove the user from both $ppl(A)$ and $ppl(M)$ before computing this metric.

A second metric uses the observation that some messages related to an activity may be exchanged with only a subset of members in the activity. This metric calculates the fraction of people in the message that belong to the activity:

$$SimSubset(A, M) = \frac{|ppl(M) \cap ppl(A)|}{|ppl(M)|}$$

As before, we remove the current user from the activity and the message when calculating this metric. This metric is 0 when the message is not addressed to anyone in the activity, and 1 when all the people addressed by the message are part of the same activity. Compared to $SimOverlap$ however, $SimSubset$ does not decrease when a message is not addressed to all the people in an activity, which matches our informal observations of activity-oriented messaging. We hypothesize that $SimSubset$ will perform better than $SimOverlap$ at email activity classification.

We also experiment with different thresholds, so that if the similarity is not above a certain threshold, the system does not make any prediction. No prediction is considered equivalent to predicting the null activity.

5.4.1 Evaluation: recommendation

We begin by evaluating the person-based algorithms on the recommendation task (predicting the top N activities given a message, for N=1 and N=3, and determining whether the correct activity is in the list). In this task, we assume that the user is only interested in the system's predictions for messages that truly belong to an activity, and that for no-activity messages, the user will simply ignore the system's recommendations with no ill effect. Thus we designed the experiment as follows.

Algorithm	$Acc_{rec,1}$	$Acc_{rec,3}$
SimOverlap 0.5	0.61	0.61
SimOverlap 0.7	0.39	0.39
SimOverlap 0.9	0.24	0.24
SimSubset 0.5	0.87	0.91
SimSubset 0.7	0.83	0.86
SimSubset 0.9	0.79	0.83
Threading	0.58	N/A
Naïve Bayes	0.24	0.40

Table 5.2: Recommendation results for person-based models, averaged over 100 random message orderings and 14 random orderings for naïve Bayes. Each row represents the use of a different person-based similarity metric, and the corresponding threshold. Threading and naïve Bayes are baseline methods for comparison.

For each message that belongs to an activity, use the current model to predict up to N activities. If the correct activity label is in this list, then count this example as correct. Train the activity model with this message, and repeat on the next message that belongs to an activity. The accuracy of the system is the fraction of correct messages out of the total number of activity-labelled messages.

We compare the person-based models against two baseline approaches: threading and a naïve Bayes classifier. The threading approach uses the in-reply-to field in message headers to group together messages in a thread. If a message is a reply to a previous message, then predict the activity to which the previous message belonged, otherwise predict the null activity. As Threading cannot predict more than one activity, we only report results for the N=1 experiment.

We used an updateable (online) naïve Bayes classifier from the Weka [252] toolkit. For features we produced a stemmed version of the message body using a Porter stemmer [190]. In addition, we also provided features from the message header, including the subject and people who sent and received the message. This provided the classifier with the information available to our other methods. We ran each test 14 times using a randomized message ordering as explained above.

Table 5.2 shows the results of this experiment for the person-based activity models. On this recommendation task, the SimSubset metric clearly produces higher accuracy than the SimOverlap metric, as we expected. Surprisingly, the top-1 and top-3 figures are quite similar. We interpret this as meaning that, at least within this data set, the set of people is a fairly good indicator of the activity, and where any match exists, the person-based activity model will make the correct prediction. However, an investigation of the data reveals that in many cases, the set of people in an activity changes over time, which causes a person-based activity model's accuracy to decrease. For instance, in one of the activities involving ordering a piece of equipment, the activity started out with just the employee and a manager, but then expanded over time to include the site equipment manager, a procurement specialist, the manufacturer, a reseller, and an automated workflow system. One limitation of person-based models is that they will not perform well on activities whose membership changes dramatically over time.

5.4.2 Evaluation: recognition

The recognition task is more challenging because the system must distinguish between messages that are part of an activity, and messages that are not. In this experiment, we train the system on all messages, not only messages that are part of an activity. No-activity messages are treated as belonging to the null activity. The model is allowed to predict any previously-seen activity or the null activity. The SimOverlap and SimSubset models output the null activity when there is no matching activity whose score is higher than the specified threshold.

To optimize the naïve Bayes classifier for F_3, we oversampled the activity messages. Instead of training on each activity message once, the classifier trained on each activity message M times while training on no-activity messages only once. The result was a more balanced dataset and an improved F_3. We varied M to produce the optimal oversampling amount; F_3 increased as M increased. We selected an optimal $M = 60$ for our baseline comparison.

Algorithm	Acc_{rcg}	*Precision*	*Recall*	F_3
SimOverlap 0.5	0.82	0.42	0.60	0.58
SimOverlap 0.7	0.87	0.60	0.39	0.40
SimOverlap 0.9	0.85	0.49	0.24	0.26
SimSubset 0.5	0.78	0.39	0.88	0.79
SimSubset 0.7	0.86	0.52	0.84	0.79
SimSubset 0.9	0.87	0.55	0.80	0.77
Threading	0.93	1.00	0.58	0.60
Naïve Bayes($M = 60$)	0.28	0.11	0.29	0.21

Table 5.3: Recognition results for person-based models, on the original message ordering. Naïve Bayes results based on 14 randomized runs with oversampling. $Prec$ is the precision, Rec is the recall, F_3 is the F-measure with $\beta = 3$.

Table 5.3 shows the results for the person-based models on this recognition task. Although according to the Acc_{rcg} metric the SimOverlap methods perform best, examination of the raw numbers reveals that the SimOverlap methods actually perform very poorly at correctly categorizing activity messages into activities (cell D). The high accuracy is primarily due to the fact that they correctly categorize no-activity messages as belonging to the null activity (cell A).

In contrast, the F_3 metric decreases the importance of no-activity messages and provides a more accurate metric of how useful these models would be in an actual activity recognition system. As expected, SimSubset models perform better using this metric than either SimOverlap, Threading or naïve Bayes. The dramatic difference between naïve Bayes and our methods shows the difference between standard foldering and our email activity classification research.

As SimSubset 0.5's recall is better than SimOverlap or SimSubset with higher thresholds, we chose it as the representative person-based algorithm for the rest of the chapter.

5.5 SimContent

While some activities consist of distinct sets of people, it is often the case that the same set of people may collaborate on multiple activities. For instance, a team of coworkers may work together on an IUI submission, while simultaneously preparing the next release of their software into product. Alternatively, as discussed above, the people involved in an activity may change over time. Person-based activity recognition approaches will break down in both of these situations. However, we hypothesize that there is still a common thread in the content of these messages. Thus, our second activity classification algorithm incorporates information about the textual content of messages using a variant of latent semantic indexing [2, 71].

Our SimContent algorithm is based on the observation that messages in an activity tend to concern similar topics. Hence, if message M_1 addresses the same topics as another message M_2 and M_2 is associated with activity A, then it is likely that M_1 is also associated with activity A.

To determine message similarity, we use an algorithm based on latent semantic indexing (LSI). Unlike traditional bag-of-words models and similarity metrics that are based on how many words two documents share in common, LSI projects documents onto a reduced-dimensionality subspace, and computes similarity as the distance between two document vectors in the reduced subspace. The intent of subspace projection is to capture concepts inherent in similar documents, such that each dimension in the subspace corresponds to a different concept in the document corpus.

We use LSI to classify emails into activities as follows. Assume that the user has labelled all activity-related email received thus far (and that unlabelled messages are implicitly labelled with the null activity). Build an LSI index containing all message documents seen thus far. On receipt of a new incoming message, compute its similarity to all documents in the index, and output a ranked list of documents and their corresponding scores. Apply activity labels to the documents in order to produce a ranked list of activities. Given this ranked list, we formulated a number of metrics for producing a ranked list of activities

Algorithm	$Acc_{rec,1}$	$Acc_{rec,3}$
LSI, weighted-score(5,3)	0.84	0.92
IRR, weighted-score(5,3)	0.77	0.90
LSI, top 3	0.86	0.92
IRR, top 3	0.86	0.94
Threading	0.58	N/A
Naïve Bayes	0.24	0.40

Table 5.4: Recommendation results for SimContent models, using several different metrics to calculate the top scoring activities, evaluated only on the original message ordering. Threading and naïve Bayes are baseline methods for comparison.

most similar to a given message:

- **Top N**: output the first N unique activity labels.
- **Weighted score (P, N)**: for the P most similar documents, group them by activity label. Add up the scores for all documents in each activity, and output the N activities with the highest score.

In our experiments, the weighted-score metric with $P = 5$ resulted in the highest performance on the recognition task, so we have used that metric for all further experiments.

One improvement to LSI, iterative residual rescaling (IRR) [2], purports to improve LSI's performance when the document corpus contains a non-uniform distribution of document topics. This distribution is probably an accurate characterization of activity-oriented email; based on our evaluation dataset, for example, 88% of email does not belong to any activity, whereas the remaining 12% is unevenly distributed over 27 different activities. Thus we hypothesize that IRR will do better than LSI at the recognition task.

5.5.1 Evaluation: recommendation

Table 5.4 shows the results for the recommendation task for SimContent. Because the LSI and IRR classifiers take significant amounts of time to train, we report results only on

Algorithm	Acc_{rcg}	*Prec*	*Rec*	F_3
LSI	0.86	0.52	0.76	0.72
IRR	0.87	0.54	0.76	0.73
Threading	0.93	1.00	0.58	0.60
Naïve Bayes($M = 60$)	0.28	0.11	0.29	0.21

Table 5.5: Recognition results for SimContent experiments. The weighted-score (P=5) metric was used to calculate the most likely activity.

the original message ordering of the dataset. Both LSI and IRR perform better than our baselines.

Surprisingly, IRR does not perform better than LSI. We hypothesize that this is because for the recommendation task, the size of topics is relatively uniform (between 1-38 messages), and thus IRR's advantages in handling unevenly-sized topics are not relevant here.

5.5.2 Evaluation: recognition

Table 5.5 shows the results for SimContent on the recognition task, using the weighted-score metric ($P = 5$) for calculating the most likely activity. Again, the results are reported only for the original message ordering.

LSI and IRR produce nearly the same results on this problem. Compared to Threading, LSI and IRR are less precise but have higher recall, indicating that they can more often correctly identify messages as being part of an activity than Threading. This is reflected in the higher F_3 score for LSI and IRR compared to Threading. For an activity management system where it is more important that a message is identified as part of an activity than to have it be correctly classified into the right activity, we believe that the SimContent algorithms are an improvement over the baseline Threading method.

Algorithm	Acc_{rcg}	*Precision*	*Recall*	F_3
Threading	0.93	0.99	0.58	0.60
SimSubset	0.70	0.33	0.86	0.74
SimContent	0.83	0.46	0.74	0.70
Voting	0.93	0.74	0.81	0.81

Table 5.6: Results from voting, over 14 randomized message orderings.

5.6 Combined model

While our SimSubset and SimContent methods perform better than the baseline Threading and naïve Bayes methods, we wanted to improve the system's performance on the recognition task even further. We observed that it was often the case that when two base methods agreed on an activity label, it was likely to be correct. This observation led us to try a simple voting approach: for each message, generate the results of all three base methods. If the method fails to make a prediction, predict the null activity.[2] Give each resulting prediction one vote; the final prediction is the label with the most votes, choosing randomly in case of ties. We selected as our three base learners Threading, SimSubset 0.5 and IRR weighted-score (N=5).

Table 5.6 shows the results of applying this voting method to the three base learners, over 14 different randomized message orderings. The combined Voting algorithm is able to match the Threading method's accuracy, increasing the recall to nearly the level of SimSubset, and having a precision somewhat in between SimContent and Threading. Thus overall, Voting receives the highest F_3 score, indicating that its performance is superior to that of any of the individual base methods on the email activity classification problem. Since our methods achieved an accuracy of close to 95% for the recommendation task, we did not attempt to create a combined approach for recommendation as well.

One concern is that the performance of a system would be very high in the early stages, when there is a small number of activities to predict, and then degrade over time as the

[2]Other weighting schemes could be used, such as using mixture of experts algorithms like weighted majority [150]. For simplicity we used equal weighting for each method.

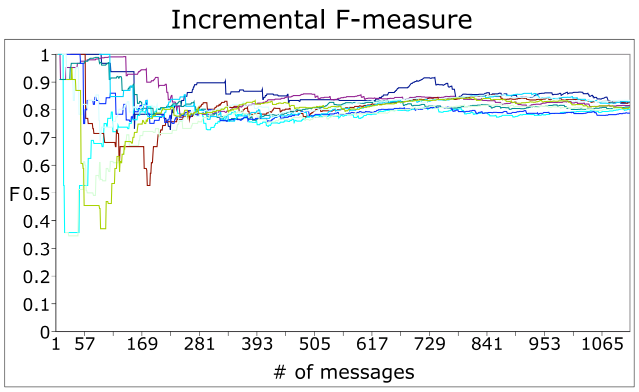


Figure 5.4: Results showing incremental F_3 on 14 randomized orderings.

number of activities increases. However, we have found that despite the difficulty of this problem, our combined approach manages to achieve a fairly consistent performance over the course of the dataset. Figure 5.4 shows the dynamic behavior of the F_3 for several randomized orderings as a function of the number of training messages.

5.7 Related Work

Classifying email into activities is conceptually similar to the traditional problem of email classification. Most work in email classification has focused either on spam detection [205, 212] or automated foldering [1,132,213], which we review in chapter 9. The vast majority of email classification systems have employed text classification techniques such as naïve Bayes, rule learners, and support vector machines. Our work is different from traditional email classification in a number of ways. First, most assume that the set of folders is static and known ahead of time (with the notable exception of SwiftFile [213]), whereas we are investigating the more challenging problem where the set of class labels (activities) grows over time. Second, most email classification systems examine only the words in the message body and headers to predict its folder classification. Since our focus is on activities, which involve features such as a set of people with specific roles to play, rather than folders, which are only collections of messages, we can leverage activity-specific features in order to improve performance. Finally, foldering often assumes every message is placed in a folder, whereas in our data we have found that the vast majority of emails are not part of a managed activity.

A few email classification systems go beyond message content in classifying email. For example, Kiritchenko *et al.* [133] mine temporal patterns in email such as the fact that messages to a mailing list all tend to arrive at the same time. As another example, emailSift [1] uses subgraph detection to find patterns that characterize all the messages in a folder, to build a model of the folder. These techniques may improve activity classification.

Kushmerick and Lau [138] examined email activities that represented structured business processes such as purchasing books online, and reverse-engineered the structure of processes by learning them from representative email transactions. While those techniques worked well for fairly structured activities, this work addresses the more general problem of less-structured activities that are not necessarily generated by formal business processes. We view this work as complementary with that of Kushmerick and Lau, whereby structure can be learned from structured activities grouped by our method.

Khoussainov and Kushmerick [130] inferred relationships amongst messages in less-structured activities. However they assume a batch-processing model in which activities are to be discovered within an archive of past messages; in contrast, here we attack the more challenging problem of tracking an evolving set of activities as they unfold. Similarly, others have attempted to discover activities in a user's inbox using clustering [121,232]. However, these systems are not dynamic. In contrast, we build and evaluate our models in a way that better approximates real-world, ongoing usage rather than a one time clustering.

Several HCI systems surveyed in chapter 9 help users manage their workload more effectively through activity management tools. For instance, Bellotti's Taskmaster system [12] enhances an email client to function as a task management system. It assumes as a starting point a correspondance of tasks to message threads. In our work we have found that this does not sufficiently capture the entire activity. Muller's ActivityExplorer system [174] enables people to collaborate around shared artifacts, but does not integrate collaboration with email, despite the observations made by many email researchers [251] that email is used extensively for collaboration and activity management.

Finally, semantic email [163, 164], in which an email message contains an embedded query allowing a system to process the message as part of a larger task, pursues the same goals as our system. Our projects are complementary in that semantic email seeks to utilize existent meta-data while we attempt to generate meta-data given the emails. Even if semantic email applications are adopted, however, they will still need to interact with legacy systems and this is where a system such as ours could prove useful.

5.8 Conclusion

In this chapter, we have investigated two different approaches to the problem of email activity classification, based on the people involved in an activity and the contents of messages in the activity. We have shown empirically that the SimSubset and SimContent algorithms perform better on our dataset than the baseline approach of message threading and a naïve Bayes classifier, and that a combined model that votes together the predictions of all three base learners performs better than any invididual learner alone.

Furthermore, our results demonstrate that email activity classification is a very difficult problem. For the recommendation task, a naïve Bayes classifier, a standard approach to classification tasks, is unable to correctly identify the activity with 3 guesses half the time. Threading, the most common approach, correctly recommends the right activity less than 60% of the time. Such poor performance makes it difficult to implement the interfaces that we envision. In contrast, our methods produce an accurate suggestion 94% of the time. The recognition task proved to be even more challenging since the activity related messages are only a small fraction of the total messages. Where the best baseline methods performed with an F_3 of 0.60, our combined method achieved 0.81. In practice, our system can automatically populate activities with email such that more than 80% of the relevant messages are automatically identified. The accuracy of our methods allows for the development of a useful and accurate activity management system.

Chapter 6

Faceted Browsing of Email for Triage and Search

6.1 Introduction

So far we have considered intelligent email applications that support the first two email overload scenarios: email triage and email activity management. The third primary use of email is search and organization: critical for finding important information in ever growing mail archives. In recent years, the speed and effectiveness of simple message search has encouraged users to forgo message organization. In fact, the popular Gmail mail service encourages such behavior: “You don’t have to spend time sorting your email, just search for a message when you need it and we’ll find it for you.”[1] Additionally, the sheer volume of messages makes effective organization a time consuming task. The result is that many users have a single folder containing all of their messages. Fisher *et al.* [99] found that a third of email users did no foldering and only 21% frequently filed their mail. Even users who folder may end up with large folders, making message retrieval difficult.

Increased message volume means that inboxes contain more messages for a user to processes. As we saw in previous chapters, user’s have difficulty processing a single large

[1]`http://mail.google.com/mail/help/intl/en/about.html`

list of messages. In this respect, triage and search share a common characteristic: they both require a simple way for a user to understand a long list of messages.

In this chapter we introduce a tool for both surveying a large number of inbox messages quickly and finding specific messages: faceted browsing for email. When a user views a group of messages, either the inbox, the contents of a folder or the result of a search query, our system generates links to select subgroups of the messages based on facets. These facets depend on the current messages, making them relevant to the current user task. For example, a user search for "john" may match many such people. Our system would provide a link to select all messages from "John Smith" or "John Jones." Alternatively, the user may look at the suggested groups of messages in the inbox (ex. all mail to a specific mailing list) and select these messages to process together. For each collection of messages, our system generates links to select relevant groups.

Faceted browsing, browsing objects by selecting groups using characteristics from the metadata, is popular for many e-commerce stores, allowing users to filter results based on properties, such as "brand" or "memory type." Since email messages have structured metadata, such as associated people, attachments and read status, faceted browsing can be extended to email.

Some applications can display all relevant facets. For example, in a faceted search interface to Noble prize winners [2] all the facets (gender, country, affiliation, prize, year) and all their values easily fit on the home page. However, in email there are too many facet/value pairs to list them all since facets include to, from and cc to every correspondent. While some of these facets may be relevant, many are spurious and unhelpful. Effectively applying facets to the email domain requires a method for selecting a small set of useful facets.

We build a faceted browsing system for Gmail (Google Mail) [3] using search operators, a set of special terms that retrieve emails based on message metadata. For each collection of messages, we enumerate all possible search operators and operands that match a subset

[2]http://orange.sims.berkeley.edu/cgi-bin/flamenco.cgi/nobel/Flamenco
[3]http://mail.google.com

of messages. A search operator ranking system orders the available operators, selecting the highest ranked operators to show to the user. The machine learning operator ranker uses a variety of features based on user behavior, the results and operator values. Given a small number of useful operators, the user can then filter messages in the current view, enabling effective triage and search.

We begin with a review of faceted browsing. We then present the user interface and data collection system used by about 200 users to collect usage data. Several ranking systems and features are evaluated using collected data showing that rankers can yield high quality suggestions. A user survey shows how our tool can aid search and triage behavior. We conclude with related work and a discussion.

6.2 Faceted Browsing

New approaches to effectively browsing large repositories of information have focused on providing users with a better understanding of the data. Faceted browsing uses a category system to allow users to navigate and filter a set of data objects. Facets are meaningful labels that convey to the user different characteristics of the data, allowing the user to easily modify the current list of results. When such metadata is available for a dataset, a hierarchical faceted category system provides the advantage of clarity and consistency as compared to clustering systems [116]. Additionally, facets provide a simple way for users to understand the current set of results and to give feedback to the system by modifying the result set [126]. Several studies have shown improved user experiences because of this interaction [184]. Such systems are popular and have been applied to web search [96], blog search [102], mobile search [127] and images [255].

While facets can enhance browsing and search interfaces, various applications of facets pose challenges. For example, images often contain descriptions or captions but not categorical labels, requiring the extraction of categories from text [255]. The application of faceted browsing to the web and text corpora requires identification of facets from text [66] and inducing hierarchies from semi-structured metadata [226].

Facets for email are potentially useful for two critical email activities: triage and search. During triage, facets help the understanding of inbox content and select related messages for processing. For search, facets offer a means of refining the query, a simple way for the user to provide more information for vague queries.

Faceted browsing for email was recently introduced in Seek, an extension for Thunderbird.[4] Seek generates facets for a user's mailbox and provides an interface to filter and find messages using these facets. In the case of email, facets can be created automatically from email headers, extracting information about people, attachments, folders/labels, flags and the read status of a message. While this obviates the need for human generated facets, it yields a large number of facets, making it unclear which to show to the user. While most facet systems show facets that cover all items in the collections, we seek only a few facets to enable fast understanding of the content. While we could show all facets, we seek the most useful. The FacetMap interface by Microsoft Research demonstrates that visualizations of facets aid understanding of the object collection [222]. However, such visualizations require a lot of space. In our application, the limited space means we can show only the most useful facets for the current collection of messages.

In this work we address this issue by creating a system to rank facets, from which a few can then be shown to the user. We also investigate the usefulness of facets for triage and search through a user survey.

6.3 Operator Suggestion System

We built a prototype faceted browsing system for Gmail to collect data for evaluating rankers. We evaluate the usefulness of our system through user feedback. Gmail has several search operators to retrieve messages that match specific metadata queries. For example, users can type "from:john", "to:me", "is:unread" and "has:attachment" to retrieve

[4] `http://simile.mit.edu/seek/`

messages matching those requirements. Search operators are a natural way of incorporating facets into Gmail since each operator and operand pair is a potential facet that will match a subset of the messages. Additionally, it is easy to generate all possible search operators for a given set of messages by enumerating all values in the headers of the messages.

Each operator has both a type (to, from, attachment, label, etc.) and a value (person name, label name, etc.) In addition to generating operators for each person in the to, from and cc fields, we also generated email domains, so that a user could select "to:google.com." A list of the operators used by our system appears in table 6.1. These and other operators are documented in the Gmail Help Center.[5]

Since Gmail is thread (conversation) oriented, our system operates on threads instead of messages. For a given set of threads currently viewed by the user, the system enumerates all possible search operators and their values. The system uses up to the first 100 threads that match the current user view, which contains many more messages.[6] Operators that matched only a single thread were ignored.

Since this yields a large number of operators, 10 are selected for display to the user. We chose 10 operators to maximize the options available while keeping the list short enough to read quickly. Two requirements were considered when deciding what operators to show to a user. Users should be shown useful suggestions since we want to encourage use of the system both for data collection and user feedback. However, since users click only on displayed operators, the collected data is biased to favor the selection heuristic.

To balance these two requirements, a heuristic system was combined with a probabilistic sampling of operators. The operator selection heuristic was based on the idea that good operators will match multiple messages. While there are many subgroups of messages that can be formed in any given mail view, useful operators should form groups with many messages. Therefore, operators were scored according to the number of matching threads. However, always selecting the top operators biases our training data since only

[5]`http://mail.google.com/support/`

[6]Since Gmail sorts threads in reverse chronological order, these were the 100 most recent threads.

Operator	*Definition*	*Example*
from	Message sender	from:bill - Messages from Bill
to	Message recipient	to:david - Messages sent to David
label	Messages with a given label	label:friends - Messages labeled friends
has:attachment	Messages with attachments	has:attachment from:bill - Messages from Bill with an attachment
is:starred	Messages that are starred	is:starred to:me - Messages to me that are starred
is:unread	Messages that are unread	is:unread to:me - Messages to me that are unread
cc	Message cc	cc:bill - Messages cced to Bill
is:chat	Chat messages	is:chat from:bill - A chat from Bill
is:draft	Draft messages	is:draft to:bill - A draft I am writing to Bill

Table 6.1: The types of the search operators used as facets by the search operator suggestion system. The system generates operators of these types paired with a value to create facets for a mailbox view. These and other operators are documented in the Gmail Help Center.

frequently occurring operators are shown. Instead, we randomly selected operators for display using their scores as a probability distribution. Given the number of threads matched by an operator o as N_o, we selected an operator for display with probability $\frac{N_o}{\sum_o N_o}$. Our evaluation will show that users sometimes selected operators that did not match the most messages. Finally, to remove the bias against selecting operators displayed first in the list, we randomized the order of the 10 selected operators.

6.3.1 User Interface

The search operator suggestion user interface was integrated into Gmail's interface. When a user viewed a list of messages, whether the inbox, search results, or any list of threads, the system displayed 10 search operators below the search box and above the thread list. Suggested operators were only displayed when the view contained at least 10 threads. A screen shot of the user interface is shown in figure 6.2. Our primary motivation for placement below the search box was visibility so as to elicit clicks and feedback from users. Additionally, a production system may chose to display greater or fewer operators based on their usefulness, which could be determined by the ranking system.

Each operator is displayed as it would be typed in the search box. We chose this format so that users would become familiar with Gmail's search syntax. Each operator is a link that, when clicked, filters the current view to show only threads matching the operator. The search operator is added to the search box in addition to any existing terms for the current view. For example, when a user selects "is:unread" in the inbox view, the system will show all unread threads in the inbox by executing the search "is:unread in:inbox." When a user searches for "tom" and then selects the suggested "has:attachment" operator, the system will show all threads containing attachments and the word "tom." Additionally, each operator has a "-" link that filters the current view to exclude threads that match the operator. This is useful for selecting views such as "not to me" (-to:me) or "is not to the mailing list announcements@google.com" (-announcements@google.com). For person operators, the person's name was used if available in the message headers; otherwise the email address was shown. We added an additional feature after the data collection phase of the experiment was complete: the ability to highlight threads that match an operator by mousing over the suggested operator.

This system was deployed to Google employees who volunteered to take part in the experiment after reading a description of the experiment and the data collected. 183 volunteers used the system over the course of eight weeks, with users joining and leaving the experiment during that time. To collect data for evaluating search operator rankings, the system logged each view request, the time, user, number of matching threads and any search query. The system also logged all search operators generated for the view, including which were shown to the user, the number and position of matching threads, and other properties of the operator useful for ranking. Additionally, the system logged when a user clicked on an operator, which was used as feedback to determine which operators were useful. No information about message contents was logged.

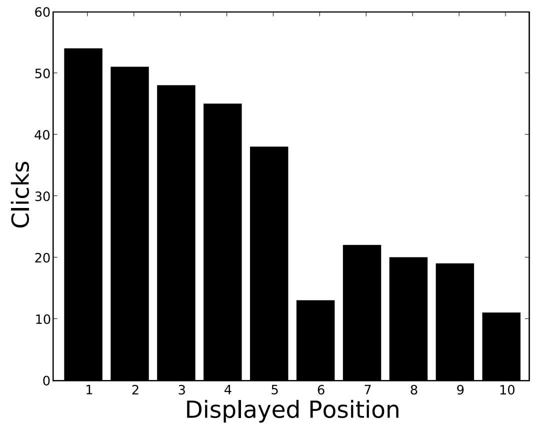
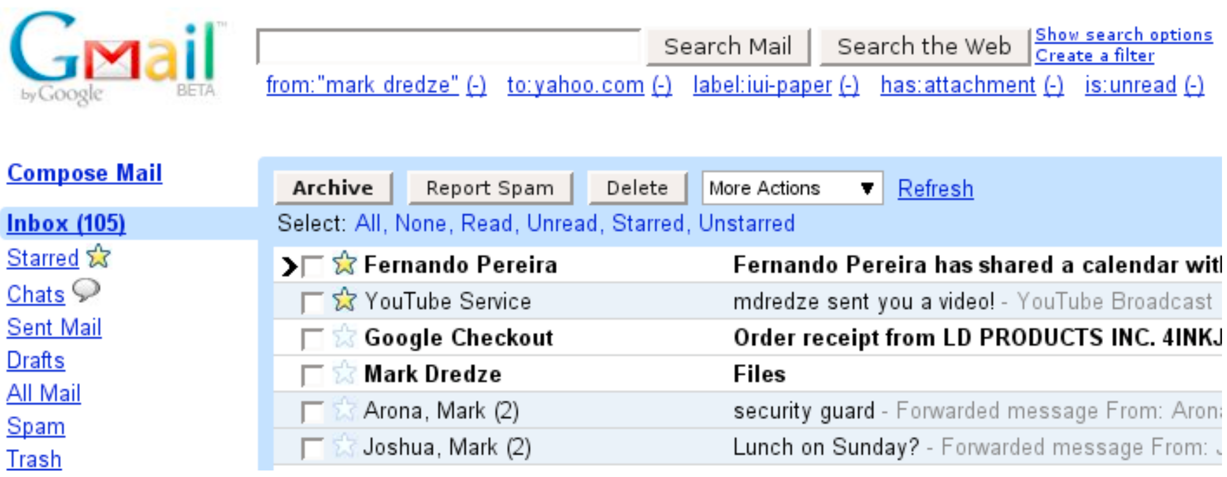


Figure 6.1: Number of clicks for search operators per display position. Users were twice as likely to click an operator in the first five positions compared to the last five. While this indicates a strong bias towards earlier ranked operators, the bias is much less than for web search. Joachims *et al.* [125] found that users clicked on the first result nearly half of the time and on the top 5 results about 80% of the time. Our weaker bias may be due to the ease by which users could view other options. Figure 6.2 shows that all suggested operators fit in a small amount of space. Each option is just 2-3 words, meaning the user can quickly scan all of them. For web search results, the user must scroll through the options, where each result can be a few sentences.

6.4 Search Operator Ranking

To determine which operators to display to the user, we introduce the task of search operator ranking, whereby possible operators are ranked based on the likelihood that a user will click on the operator. We define the task of search operator ranking as follows. A ranking instance z is defined by the pair $z = (\mathbf{x}, y)$, where $\mathbf{x}$ is a set of search operators and y is the operator selected by the user from the set $\mathbf{x}$ ($y \in \mathbf{x}$). Each operator $x \in \mathbf{x}$ is defined by a set of properties or features, which we define below. A ranking function $\mathcal{R}$ orders the set of operators $\mathbf{x}$. For this work we use the simple 0/1 loss function $\mathcal{L}$ to train the rankers, where $\mathcal{L}(\mathcal{R}, z) = 1$ when $\mathcal{R}$ assigns y the highest score and $\mathcal{L}(\mathcal{R}, z) = 0$ otherwise. We investigate more informative metrics for evaluating ranker performance below.



Fernando Pereira — Fernando Pereira has shared a calendar wit
YouTube Service — mdredze sent you a video! - YouTube Broadcast
Google Checkout — Order receipt from LD PRODUCTS INC. 4INKJ
Mark Dredze — Files
Arona, Mark (2) — security guard - Forwarded message From: Aron
Joshua, Mark (2) — Lunch on Sunday? - Forwarded message From:

Figure 6.2: The search operator suggestion user interface for data collection. The suggested search operators appear below the search box.

6.4.1 Training Data

From the logs generated by the operator suggestion system we created a dataset of ranking examples used for evaluating our ranking algorithms. We extracted instances where the user selected a search operator by clicking on the operator or its negation. The clicked operator was taken as y and all of the operators (shown and not shown) formed x to create a ranking instance z. Additionally, we used examples where the user explicitly entered an operator into the search box if that operator was generated by our system. We used the information retrieval community's common approach to collecting feedback for ranking: the user clicked operator is treated as the best operator from the list and the system should rank this operator highest.

We collected 436 ranking examples from 183 users. However, the majority of users did not use the system very heavily, generating a small number of training examples (1-2.) Learning from so few examples per user is problematic, especially since 1-2 clicks may indicate initial experimentation with the system and not actual use. Therefore, we created a second dataset from users who had at least 10 examples. These users had an average of 24 clicks and a total of 235 instances. We name the datasets *All* and *Min10* respectively.

While we displayed operator suggestions for every view, the vast majority of usage came from the inbox and search results. In the *All* dataset, 62% of examples were generated from inbox views while 29% were from search results. While inbox examples were twice as common as search results, viewing the inbox is a much more common task. We investigate usage through user feedback below.

We are aware of several issues with our training data collection method. Assuming that user clicked operators are correct is questionable. First, the user expressed a preference for the clicked operator over the other displayed operators, not over all generated operators. Second, the user may have found several operators to be useful but could only select a single operator. Finally, since the user was presented with an ordered list of operators, operators that occurred earlier in the list are more likely to receive clicks. All three of these problems are common to many ranking problems, such as web search [193]. Joachim's *et al.* [125] carefully studied some of the issues for ranking web search results using click through data. They compared manual relevance judgments, eye tracking behavior and click through rates to identify correlations. They concluded that while clicks could not be interpreted as absolute relevance judgments, the relative preferences derived from clicks are reasonably accurate on average. Therefore, we use click through data for our ranking evaluations.

A second problem is the bias introduced by our heuristic for ranking operators by the number of matching threads. Users are more likely to click on operators that match many threads simply because these are shown more frequently. Our results will indicate that users selected operators that were not in the top ten of counts a significant portion of the time. This means that this simple heuristic cannot effectively rank options: learning systems can improve over this heuristic. We use this method of data collection as a compromise between accurate annotations and showing the user a reasonable number of options.

Since our results below show that we can successfully learn from this data, it seems that these problems do not hinder learning good rankers. Additionally, since the evaluation only credits a ranker with returning a single correct operator, top rankings may actually contain many good suggestions, appearing worse in our evaluations than they would in real world systems.

6.5 Ranking Systems

We now explore algorithms for ranking search operators. We begin with some baseline systems to measure the difficulty of the task and then proceed to several ranking systems.

6.5.1 Baseline Systems

Random - This ranker randomly orders operators, a baseline to determine task difficulty. Since examples contain 80 operators on average, random ranking should do poorly.

Displayed Order - This ranker measures bias of the display order by ranking operators in the order they were displayed to the user. Operators not shown to the user are randomly ranked at the end of the list. As expected, the data show a bias towards operators that were displayed earlier in the suggested list as evidenced by a histogram of clicks by operator position in the *All* dataset (figure 6.1). Operators in the first 5 positions received twice as many clicks as the last 5 positions. Therefore, this baseline ranker should do well.

Max Count - Operators that match the most threads are ranked first, ie. the data collection heuristic, except without random sampling or permutation. This measures bias of the data collection heuristic.

6.5.2 Ranking Systems

We now consider three systems that rely on user behavior, the threads in the current mail view and properties of the operator values.

Split Results - Since facets are meant to extract groups of messages, useful operators should match a sizable group of messages. However, unlike *Max Count*, we do not want operators that match very large groups. For example, an operator that matches two or 99 threads out of 100 is likely not useful. Useful operators should split the data evenly in two.

This ranker orders operators by how well they split the data, specifically it uses the scoring function:

$$\text{score} = -\text{abs}(\frac{N_v}{2} - N_o) ,$$

where N_v is the number of threads in the current view and N_o is the number of threads matching the operator. If an operator matches all or none of the threads in the view, its score is large and negative. If it matches exactly half of the threads it achieves the maximum score of 0.

Most Popular - Previous user behavior is a good indication of an operator's usefulness. Operators that are frequently selected by a user are probably more useful. Additionally, the context in which an operator was selected may influence its selection. For example, a user may search for the term "john" and then select the operator "from:john smith." In this case, the operator was useful in the context of the query "john" but may not be relevant to other queries.

To capture the popularity of an operator, we build a user model based on previous user searches. The history records the terms used in all user searches. History lookups indicate both how often an operator was used in general and in the context of a query. Using this information we implemented three rankers:

- **By operator** - Score an operator by how many times it has been used in previous queries. This includes every occurrence of the operator in the user's query history.

- **By terms in query** - Given the terms in the current query (if any) how many times has the operator occurred with all of these terms in previous queries? This includes all permutations of the query and any previous queries that contained all these terms as a subset. For example, for the term "from:bill" and the query "to:me", a match would be found in the previous query of "from:bill to:me has:attachment."

- **By exact query** - Given the terms in the current query and the operator, how many times have all of these terms been used together without any additional terms. For example, for the term "from:bill" and the query "to:me", the previous query of "from:bill to:me has:attachment" would not match since it has an additional term "has:attachment."

These three approaches provide a range of flexibility for measuring operator popularity. The first is context insensitive, the second mildly context sensitive and the third requires context. The search history is built independently for each user from training data and used to rank test examples. Ties between operators are broken randomly.

Machine Learning - The previous rankers relied on information from user history and the threads in the view. We now combine these into a single ranker. We developed a statistical machine learning ranker based on a variety of features that fall under three types: features depending on user behavior (popularity rankers,) features depending on the results (split results ranker) and features from intrinsic properties of the operator. For simplicity, all features are binary.

User Behavior Features

The first set of features are based on the search history of the user, similar to the popularity rankers. We included the output of each popularity ranker in our features.

Popularity Features - A template of features binarize output from the three popularity rankers. Each ranker had features corresponding to: pattern never appears, pattern appears in the history, pattern used once, pattern used twice, pattern used three or more times. The pattern is the string matched by each of the three popularity rankers. (15 features)

Relative Popularity Features - While an operator may not be popular, it may be popular relative to other operators in the example, or vice versa. We added features based on the relative popularity ranking of operators in an instance: most used pattern, second most used pattern, third most user pattern, top 5 most used pattern. (15 features)

Result Oriented Features

The next set of features are based on the threads in the current view. As above, we created a feature template to convert the output of the split ranker into features.

Split Features - Features according to ranking order in the *Split Ranker*'s output. The features are: operator is ranked number 1, 2, 3, in top 5, not in top 5. (5 features)

Count Features - The same features but applied to the output of the *Max Count* ranker. (5 features)

Result Order Features - An operator may match many results, but the user typically only looks at the first few. Additionally, since threads are sorted in reverse chronological order, operators matching the top threads indicate relevancy to recent mail. Feature were created as: operator has no matches in first n threads, operator matches less than half of the first n threads, operator matches more than half of the first n threads, for $n = 5, 10, 20, 50$. (12 features)

Intrinsic Features

The final set of features are based on the types (from, to, etc.) of the operators and the values they contain ("bill", "domain.com," etc.). These features capture properties of operators consistent across users and views.

Value Features - Value's for person operators (to, from, cc) are divided into 5 possible categories: person name, domain name, the user ("me"), an email address or contains a hyphen. Each category is a feature. (5 features)

Type+Value Features - Conjunctions of the type of person operators and categories of their values. The possible types for person operators are "from," "to," and "cc." These are conjoined with the value categories above. (15 features)

Address Book Features - Address book membership may indicate familiarity and may influence the relevancy of operators for these people. We added two features: the operator value is a person name in the address book and the operator value is an email address in the address book. (2 features)

Ranker	*All*	*Min10*
Random	0.10	0.09
Displayed Order	0.31	0.30
Max Count	0.19	0.17
Split Results	0.25	0.23
Popular Operator	0.49	0.60
Popular Terms	0.43	0.52
Popular Query	0.39	0.48
Machine Learning	**0.59**	**0.68**

Table 6.2: Mean reciprocal rank (MRR) of the rankers on each dataset. The Machine Learning ranker performs the best on both datasets. All of our rankers improve over the Max Count heuristic.

Person Name Similarity - A query for a specific person may contain part of a person's name that also appears as an operator. For example, a search for "Bill" may indicate that the operator "from:bill smith" is appropriate. For operators that contain people's names, we check to see if each term in the query matches the start of one of the names. If any terms match we add a feature. (1 feature)

Notes on Features

Our features for operator ranking are inspired by the feature methodology we used for reply prediction (section 3.5.1). In this setting, we learn a single ranker for all users trained on a mix of data from different users. We accomplish this through creating a shared feature representation. As with reply prediction, many of the features here are shared features extracted using user specific information. For example, the user history features are shared by all users, but each feature is extracted based on an individual's search history. This enables successful cross user learning in this setting.

Learning Algorithm

For the machine learning ranker we learned a linear ranking function parameterized by weight vector w, where an operator x is scored as $f(w, x) = w \cdot x$. Operators are then ranked according to their assigned score. While there are many ways to learn the weight vector w, we chose a simple one-best perceptron ranker [57, 115, 217]. We found it best to use a simple learning algorithm because of the weak nature of the examples as described above, rather than forcing a margin or a k-best ranking [57]. Initial experiments with k-best learning rankers supported this hypothesis.

The perceptron is an online mistake driven algorithm. Updates to w occur only when the ranker makes a mistake, in this case when the top operator is not correct. An update increases the score of the correct operator and decreases the score of the incorrectly top ranked operator. Additive updates for w are made as

$$w^{i+1} = w^i + y - \hat{y},$$

where w^i is the weight vector after the ith training example, y is the correct operator and $\hat{y}$ is the top operator predicted by the ranker. As is typical batch settings we train the perceptron ranker by making 10 passes over the training data. For more information on online algorithms, see appendix A.1.1.

6.6 Evaluation

We evaluated our rankers on both datasets by creating 10 train/test splits for 10-fold cross validation. For each split, the training set was used to build a user search history and train the machine learning ranker while the test split was used only for evaluation. We investigated creating separate rankers for each view type (inbox, search) but found no change in performance.

We use two common metrics from the information retrieval community to evaluate ranking.

- **Mean Reciprocal Rank** - The mean reciprocal rank (MRR) is the average reciprocal rank at which the correct example appears. Specifically, for N instances, where r_i is the rank of the correct operator in the ith instance: *MRR*$= \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i}$. This metric credits rankers who rank an operator highly even if it is not in the top set of results. It is commonly used for measuring rankers for web and document search.

- **Accuracy at** n - Accuracy at n measures the accuracy of the system when looking at the top n results. For our application, this measure is particularly relevant since it determines what percentage of the time a ranker displaying n results would have shown the correct operator. We report results for all values of n between 1 and 10.

Results for MRR for the *All* and the *Min10* datasets are shown in table 6.2. Results for accuracy at n are shown in figures 6.3 and 6.4. As expected, the baseline method *Random* does very poorly. *Displayed Order* and *Max Count* perform with an MRR of 0.31 and 0.19 respectively, both of which reflect a bias in the data compared with an MRR of 0.10 for *Random*. The high MRR of *Displayed Order* is consistent with the bias observed in clicks discussed above (figure 6.1).

The *Split Results* ranker does poorly, performing slightly better than the baseline *Max Count* heuristic. All three popular rankers do significantly better than the baselines, with *Popular Operator* doing the best. Additionally, the popularity rankers do much better on the smaller dataset since every user has some history (at least 10 clicks), indicating the usefulness of user behavior. The *Machine Learning* system does best overall, with an MRR of 10 points better on *All* and 8 points better on *Min10* than the best popularity ranker. Additionally, its MRR is twice that of the *Displayed Order* baseline on *Min10*, where it has more training data for each user. The *Machine Learning* ranker is also consistently better than all other methods on accuracy for all n. In terms of accuracy on *Min10*, *Popular operator* ranks the correct operator in the top position 49% of the time and 77% of the time in the top 5 positions. *Machine Learning* ranks the correct operator in the top position 55% and 86% in the top five. While finding the correct operator in a list of five suggestions 86% of the time is helpful, it is likely that the actual accuracy from the user's perspective would be higher, since other suggested operators are likely useful as well. These results indicate that the *Popularity* and *Machine Learning* rankers could support a production system.

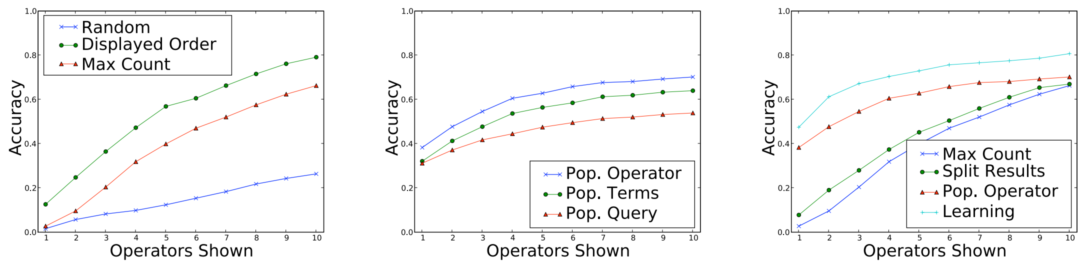


Figure 6.3: Accuracy at N for the *All* dataset. The x-axis shows the increasing n number of positions considered and the y-axis the accuracy of finding the correct operator in one of the n positions.

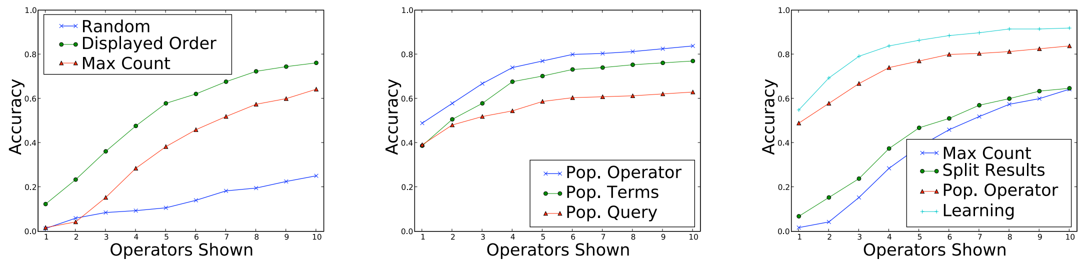


Figure 6.4: Accuracy at N for the *Min10* dataset. The x-axis shows the increasing n number of positions considered and the y-axis the accuracy of finding the correct operator in one of the n positions.

6.6.1 Feature Analysis

The success of the *Popularity* ranker indicates that these features may be beneficial for successful learning. To understand which features most impact performance, we evaluated the feature set by running the *Machine Learning* ranker on both the *All* and *Min10* datasets and removing groups of features. We report the results of evaluations after removing each feature type in table 6.3.

The most effective features are clearly based on user behavior; removing popularity features reduces performance to a third of the full MRR score. Relative popularity also has a significant impact for the *Min10* data, where there is information about previous user behavior. This result is not surprising since the popularity ranker does well; learning from its output clearly helps.

The most significant other features were the order of the results (*All*) and the value

Type	*Features*	*All*	*Min10*
All Features		0.59	0.68
User Behavior	Popularity	0.17	0.25
	Relative Popularity	0.58	0.62
Result Oriented	Split	0.60	0.69
	Count	0.62	0.69
	Result Order	0.57	0.70
Intrinsic	Value	0.60	0.66
	Type+Value	0.60	0.69
	Address Book	0.59	0.69
	Person Name	0.60	0.68

Table 6.3: Leave out feature tests MRR machine learning. The most informative features for learning are based on user behavior.

of the operator (*Min10*). These features are somewhat useful, decreasing performance by about 0.02 and 0.03 respectively. However, the majority of the signal comes from the user behavior.

We next examined the learned weights for the features, an indication as to signals learned by the system. The top positive features all dealt with user behavior, such as "all terms in the query previously used together" and "all terms in the query previously used 3 or more times." The most negative features were mostly based on user behavior, such as "term previously used" and "terms never used together." Additionally, a negative feature was "operator value matches name in query," indicating that refinements that have similar behavior to a search term may not helpful.

6.7 User Surveys

So far we have demonstrated that a ranker can deliver high quality search operators. We now turn towards evaluating the system concept and usefulness based on user feedback.

We polled experiment participants using an online survey to determine their email behaviors, thoughts on the experimental concepts and the deployed system. Users provided general information about mail usage as well as feedback on our experiment using 5 point

Likert agreement items, fixed options and free answer questions. 85 participants completed the survey. A copy of the survey can be found in appendix B and additional survey results in appendix C.

Respondents were power email users with a lot of Gmail experience. Users reported receiving hundreds of emails a day, of which most reported 80% were automatically archived using filters (skipped the inbox.) Nearly all users had been using GMail for at least a year, with 71% having 3 or more years of experience. Our users had an average of 11 labels to manage their email. We noticed different behaviors regarding inbox management. While the median number of unread inbox messages was 10, our users can be sorted into three groups.

- Proponents of the Inbox Zero method[7] who reported 0 messages (17 users.)
- Users that had a reasonable number of unread messages (< 100) (48 users.)
- Users who had inboxes that were out of control, with some users reporting 12,859 and 10,829 unread messages (19 users.)

The four statements and the average user agreement are listed in table 6.4. Users liked the idea of having search operators suggested; 74% of respondents favorably rated the concept of suggesting search operators. Additionally, 77% said they would use a tool if it produced high quality suggestions. In the following sections we consider the two primary uses of our tool: searching and inbox filtering.

6.7.1 Searching

When asked "When did you look at the list of operators?" nearly all users who said they used the system mentioned search (32 respondents.) 59% of respondents agreed with the statement "Suggesting operators for search results helps me find what I am searching for." When asked what they did next after using the operators, 91% of respondents said they

[7]Inbox Zero is an email management style in which the user processes the inbox to completion, removing all messages. For more information see: `http://www.43folders.com/izero`

Statement	*Average*	*StDev*
I like the idea of having search operators suggested.	4	1.78
I would use a search operator suggestion tool (given high quality suggestions and a good UI.)	3.98	1.21
Filtering the inbox by search operators helps me manage my inbox.	3.48	1.32
Suggesting operators for search results helps me find what I am searching for.	3.38	1.31

Table 6.4: Statements included in our user survey and their responses on a 5 point Likert item. For a complete copy of the survey, see appendix B.

read a message, indicating that the primary purpose was to find messages. Specifically, users primarily refined searches using the operators. Several users commented that they looked at the operators "when I didn't find what I was looking for" or "before... changing the query." Users said operators narrowed searches that retrieved too many results: "in a couple instances it really helped me narrow down the search." Another operation mentioned by users was the ability to exclude operators. "My search would include a lot of junk from mailing lists and other high noise sources. The tool made it really easy to filter those out with a click instead of refining the search myself." Finally, operators sometimes preempted a search: "I was about to search my inbox and just so happened to have the search suggested to me.'

6.7.2 Inbox Filtering

While only 6 people mentioned inbox management when asked when they use the tool, 54% agreed with the statement that "Filtering the inbox by search operators helps me manage my inbox." While many users were ambivalent about this use, some users found it very helpful: 28% strongly agreed with the statement and only a quarter of respondents disagreed (1-2 rating.) Additionally, many respondents indicated they labeled (20% of respondents) or archived (33% of respondents) a message after using the operators, indications of inbox management.

Inbox management comments focused on processing groups of messages: email triage.

Users reported selecting operators when “I was either triaging a large number of unread emails,” “my inbox was most full and I needed to drill down quickly” and “when I felt overwhelmed.” Our final interface contained a highlighting tool that highlighted threads matching an operator on mouseover events. Users found that highlighting supported triage behavior: “I’d just mouse over different search suggestions and look at the color change to get a rough idea of the proportion of matching items in my inbox.” Indicating the number of matching threads may also support this usage.

Several users mentioned the ability to group similar threads and select them for processing. “I’d select lots of ‘like’ messages... and archive those” or “I used it for archiving a group of messages.” Many users described how operators supported triage:

- “I mainly use the tool in the morning to process the overnight mail.”
- “It let me zero in on the most important messages first with one click and no typing.”
- “Helped reduce the pile of email to browse through.”

The highlighting was very popular with 63% of respondents agreeing that “Highlighting emails that match an operator (mouseover operator) was helpful.” Microsoft Research’s FacetMap interface [222] also demonstrated that such visualizations could be helpful. The usage of highlighting was twofold. First, highlighting visually segments the inbox, showing the size and location of various groups. One user reported “Often I’d find myself staring at my inbox, knowing that the message I need is there somewhere, but I can’t see it right away. The ability to highlight things... means I can quickly scan a small number of messages and don’t have to scan all 50 messages.” Another user said “often I just want to quickly see which emails match the operator without actually running the search.” This finding is supported by research that indicates facets help users understand search results [126]. Additionally, showing person operators can summarize the social connections in the inbox, which is useful for triage [180]. The second use case was to gauge operator behavior before filtering. “The higlighing also made me more confident that adding a particular filter wouldn’t hide the email was really looking for.”

6.7.3 Filtering with Labels

Users who liked the concept but did not use the suggested operators identified two reasons. First, many users complained about suggestion quality, which is not surprising since the deployed system used a simple heuristic. Deploying our ranking systems should address this concern. Second, many users said they already used filters and labels extensively and therefore did not need operator suggestions. The vast majority of users filter a large percentage of their mail to skip the inbox and many cited this as the reason for not using operators. "I didn't use it... I already have a well defined set of filters to filter my incoming mail." When non-overloaded users were asked how they handle email, nearly all identified filters and labels. Suggesting operators for filtering messages brings some of this benefit automatically to users without filters. Additionally, it was encouraging to find that operators may by able to help users setup these filters, as one user indicated that "there has been at least a case in which the suggested operator led me to create a new filter."

6.7.4 System Improvements

When asked to suggest system improvements, the most popular request was operator customization, such as removing unwanted suggestions or making some suggestions permanent options, such as "is:unread" and "to:me." In particular, "to:me" is a feature available in several popular tools, including Snarf [181] for social email triage and Phlat [64] for personal search. This motivates a design that enables users to create new filters and labels from suggested operators. The next most common suggestion was to show more operators and select multiple operators. These features would support searchers who were unsure how to best narrow a search and were willing to consider more refinement options. Several users asked for the ability to filter by mailing lists (select all mailing list messages.) While we had hoped that suggesting operators would make people more likely to use operators, users were split when asked if they now used more operators in Gmail search (44% agree, 41% disagree.)

Finally, we asked users to suggest new features to help with information overload. Almost all users asked for better tools to manage the inbox, with the most common suggestions involving filter suggestions, something we could integrate with our current tool. Users also asked for better visualization tools, something partially addressed by our highlighting feature. Since our system addresses both of these requests, it is encouraging to see that one user said “this operator suggestion one has been by far the most helpful [triage tool.]”

6.8 Related Work

Faceted browsing and search have been explored in a number of other settings. Its application to email primarily impacts two email behaviors: triage and search. We discuss triage in detail in section 9.1.3.

Email search has recently been explored in the TREC Enterprise track email search and expert finding tasks [59, 223]. Xi *et al.* [254] created an email importance ranking function for search based on features of the thread and authors. Dom *et al.* [77] use a graph based ranking based on social relationships to rank experts. Mock [170] ranked related emails given an email of interest. Email search can also integrate with activity management schemes, such as searching for messages related to an activity [167]. Others have focused on the search interface, such as creating search by example systems for managing email archives [9].

Email is a significant part of a user’s personal information store. The Stuff I’ve Seen project built a prototype for searching personal information based on user interactions specific to this domain [93]. They found that both time and social information are critical to this style of search. Most of our suggested operators are person operators, incorporating a social aspect in faceted browsing. Microsoft Research’s Phlat interface for personal search focused on filtering in a unified interface [64]. They concluded that providing social information for search was especially useful for personal search [63]. Sanderson and Dumais [206] found that searches are repetitive, with people typically issuing the same query multiple times. This lends support to our observation that history based features are the best for ranking operators, since people often perform the same search multiple times.

6.9 Conclusion

We have presented a system that suggests search operators for email, a tool that extends the idea of faceted browsing to email. The tool is useful for both searching and inbox management. We have empirically shown that our ranking systems can effectively select useful search operators from a large set of possible operators, thereby enabling faceted email browsing. User feedback shows that our system helps users refine email searches and supports inbox management. Specifically, our system refines mail searches, selects groups of messages during triage and provides an overview of the messages in the current view.

Additionally, the work in this chapter validates two contributions of this thesis. First, we have shown that shared representations enable cross-user learning for reply prediction. We created these types of features in this chapter for facet ranking and learned a single system for all users. This indicates that the methodology of shared representations created by user specific extractions can be applied to other tasks. Second, we have included a user survey to validate our facet ranking concept. As an example of intelligent email, facet ranking uses an intelligent component to facilitate better email user management. Our survey results indicate that users find such applications helpful, in particular facet ranking, and that intelligent systems can be used to improve email management.

Chapter 7

Confidence-Weighted Learning of Linear Classifiers

7.1 Learning Methods for Intelligent Email

So far we have considered several intelligent email learning problems. In each application, we applied existing machine learning techniques towards supporting our intended interface. The result was novelty in the field of intelligent user interfaces. In this chapter, we turn in the other direction. As we developed these systems, we encountered familiar and new learning challenges. Successful intelligent email systems will require not just applications of existing learning technologies but the development of new algorithms. Additionally, our efforts clarify machine learning and language processing research challenges as well as introduce new settings. We now motivate and evaluate research in these areas through our work in intelligent email.

Several of our intelligent email applications use supervised learning. While we evaluated algorithms in an offline batch style, we expect a deployed system to interact with the user, learning in the wild from user actions. For example, in reply prediction, after the system labels incoming emails the user has the opportunity to correct the system's predictions by relabeling the documents. When given this feedback, the system should use the labeled document as a training example to improve its current hypothesis. This setting calls for an online learning algorithm, whereby the current hypothesis can be updated from a single new example. Since reply prediction, as well several other considered applications, are instances of natural language learning, we now consider the application of online learning to natural language problems. Using intuitions about natural language problems, we develop a new online learning algorithm called Confidence-Weighted learning. Our work in this chapter is based on previously published work in the International Conference on Machine Learning (ICML) [85].

7.2 Confidence-Weighted Learning

Online learning algorithms operate on a single instance at a time, allowing for updates that are fast, simple, make few assumptions about the data, and perform well in wide range of practical settings. Online learning algorithms have become especially popular in natural language processing. Some examples include part-of-speech tagging [51,218], text segmentation [161], noun phrase chunking [51], dependency parsing [162], parsing [32], machine translation [48] and information extraction [160]. For several of these tasks, online methods are considered state-of-the-art.

In this chapter, we revisit the design of linear classifier learning informed by the particularities of natural language tasks. Specifically, feature representations for natural language processing have very high dimension (millions of features derived from words and word combinations are common), and most features are observed on only a small fraction of instances. Nevertheless, those many rare features are important in classifying the instances in which they occur. Therefore, it is worth investigating whether online learning algorithms for linear classifiers could be improved to take advantage of these particularities of natural language data.

We introduce *confidence-weighted* (CW) learning, a new class of online learning methods that maintain a probabilistic measure of confidence in each parameter. Less confident parameters are updated more aggressively than more confident ones. Parameter confidence is formalized with a Gaussian distribution over parameter vectors, which is updated for each new training instance so that the probability of correct classification for that instance under the updated distribution meets a specified confidence. We show superior classification accuracy over state-of-the-art online and batch baselines, faster learning, and new classifier combination methods after parallel training.

We begin with a discussion of the motivating particularities of natural language data. We then derive our algorithm and discuss variants. A series of experiments shows CW learning's empirical benefits. We conclude this section with a discussion of related work.

7.3 Online Algorithms and NLP

In natural language classification tasks, many different features, most of which are binary and are infrequently on, can be weakly indicative of a particular class. Therefore, we have both data sparseness, which demands large training sets, and very high dimensional parameter vectors. For certain types of problems, such as structured prediction in tagging or parsing, the size and processing complexity of individual instances make it difficult to keep more than a small number of instances in main memory. These particularities make online algorithms, which process a single instance at a time, a good match for natural-language tasks. Processing large amounts of data is simple for online methods, which require observing each instance once — though in practice several iterations may be necessary — and update parameter vectors using a single instance at a time. In addition, the simple nature of most online updates make them very fast.

However, while online algorithms do well with large numbers of features and instances, they are not designed for the heavy tailed feature distributions characteristic of natural language tasks. This type of feature distribution can have a detrimental effect on learning. With typical linear classifier training algorithms, such as the perceptron or passive-aggressive (PA) algorithms [55, 202], the parameters of binary features are only updated when the features occur. Therefore, frequent features typically receive more updates. Similarly, features that occur early in the data stream take more responsibility for correct prediction than those observed later. The result is a model that could have good parameter estimates for common features and inaccurate values for rare features. However, no distinction is made between these feature types in most online algorithms.

Consider an illustrative example from the problem of sentiment classification. In this task, a product review is represented as n-grams and the goal is to label the review as being positive or negative about the product. Consider a positive review that simply read *"I liked this author."* An online update would increase the weight of both "liked" and "author." Since both are common words, over several examples the algorithm would converge to the correct values, a positive weight for "liked" and zero weight for "author." Now consider a slightly modified negative example: *"I liked this author, but found the book dull."* Since "dull" is a rare feature, the algorithm has a poor estimate of its weight. An update would decrease the weight of both "liked" and "dull." The algorithm does not know that "dull" is rare and the changed behavior is likely caused by the poorly estimated feature ("dull") instead of the common well estimated feature ("liked.") This update incorrectly modified "liked" and does not attribute enough negative weight to "dull," thereby decreasing the rate of convergence.

This example demonstrates how a lack of memory for previous instances — a property that allows online learning — can hurt learning. A simple solution is to augment an online algorithm with additional information, a memory of past examples. Specifically, the algorithm can maintain a confidence parameter for each feature weight. For example, assuming binary features, the algorithm could keep a count of the number of times each feature has been observed, or, for general real-valued features, it could keep the cumulative second moment per feature. The larger the count or second moment, the more confidence in a feature's weight. These estimates are then used to influence parameter updates. Instead of equally updating every feature weight for the features present in an instance, the update favors changing more low-confidence weights than high-confidence ones. At each update, the confidence in all observed features is increased by focusing the update on low confidence features. In the example above, the update would decrease the weight of "dull" but make only a small change to "liked" since the algorithm already has a good estimate of this parameter.

7.4 Online Learning of Linear Classifiers

Online algorithms operate in rounds. On round i the algorithm receives an instance $\boldsymbol{x}_i \in \mathbb{R}^d$ to which it applies its current prediction rule to produce a prediction $\hat{y}_i \in \{-1, +1\}$ (for binary classification.) It then receives the true label $y_i \in \{-1, +1\}$ and suffers a loss $\ell(y_i, \hat{y}_i)$, which in this work will be the zero-one loss $\ell(y_i, \hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$ and $\ell(y_i, \hat{y}_i) = 0$ otherwise. The algorithm then updates its prediction rule and proceeds to the next round.

Just as in many well known algorithms, such as the perceptron and support vector machines, in this work our prediction rules are linear classifiers

$$f_{\boldsymbol{w}}(\boldsymbol{x}) : f_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{x} \cdot \boldsymbol{w}) \ . \tag{7.1}$$

If we fix the norm of $\boldsymbol{w}$, we can identify $f_{\boldsymbol{w}}$ with $\boldsymbol{w}$, and we will use $\boldsymbol{w}$ in the rest of this work.

The *margin* of an example $(\boldsymbol{x}, y)$ with respect to a specific classifier $\boldsymbol{w}$ is given by $y(\boldsymbol{w} \cdot \boldsymbol{x})$. The sign of the margin is positive iff the classifier $\boldsymbol{w}$ predicts correctly the true label y. The absolute value of the margin $|y(\boldsymbol{w} \cdot \boldsymbol{x})| = |\boldsymbol{w} \cdot \boldsymbol{x}|$ is often thought of as the *confidence* in the prediction, with larger positive values corresponding to more confident correct predictions. We denote the margin at round i by $m_i = y_i(\boldsymbol{w}_i \cdot \boldsymbol{x}_i)$.

A variety of linear classifier training algorithms, including the perceptron and linear support vector machines, restrict $\boldsymbol{w}$ to be a linear combination of the input examples. Online algorithms of that kind typically have updates of the form

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_i y_i \boldsymbol{x}_i \ , \tag{7.2}$$

for some non-negative coefficients α_i.

In this chapter we focus on Passive-Aggressive (PA) updates [55] for linear classifiers. After predicting with $\boldsymbol{w}_i$ on the ith round and receiving the true label y_i, the algorithm updates the prediction function such that the example $(\boldsymbol{x}_i, y_i)$ will be classified correctly with a fixed margin (which can always be scaled to 1):

$$\begin{aligned} \boldsymbol{w}_{i+1} = \min_{\boldsymbol{w}} \quad & \frac{1}{2}\left\|\boldsymbol{w}_i - \boldsymbol{w}\right\|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i) \geq 1 \ . \end{aligned} \tag{7.3}$$

The dual of (7.3) gives us the round coefficients for (7.2):

$$\alpha_i = \max\left\{\frac{1 - y_i\left(\boldsymbol{w}_i\cdot\boldsymbol{x}_i\right)}{\left\|\boldsymbol{x}_i\right\|^2}, 0\right\}$$

Crammer *et al.* [55] provide a theoretical analysis of algorithms of this form, and they have been shown to work well in a variety of applications.

7.5 Distributions over Classifiers

We model parameter confidence for a linear classifier with a diagonal Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and standard deviation $\boldsymbol{\sigma} \in \mathbb{R}^d$. The values μ_j and σ_j represent our knowledge of and confidence in the parameter for feature j. The smaller σ_j, the more confidence we have in the mean parameter value μ_j. For simplicity of presentation, we use a covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ for the distribution, with diagonal $\boldsymbol{\sigma}$ and zero for off-diagonal elements. Note that while our motivation assumed sparse binary features, the algorithm does not depend on that assumption.

Conceptually, to classify an input instance $\boldsymbol{x}$, we draw a parameter vector $\boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{\mu}, \Sigma\right)$ and predict the label according to the sign of $\boldsymbol{w}\cdot\boldsymbol{x}$. This multivariate Gaussian distribution over parameter vectors induces a univariate Gaussian distribution over the margin viewed as a random variable:

$$M \sim \mathcal{N}\left(y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i)\ ,\ \boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i\right) \ .$$

The mean of the margin is the margin of the averaged parameter vector and its variance is proportional to the length of the projection of $\boldsymbol{x}_i$ on Σ_i. Since a prediction is correct iff the margin is non-negative, the probability of a correct prediction is

$$\Pr_{\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma)}\left[M \geq 0\right] = \Pr_{\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma)}\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right] \ .$$

When possible, we omit the explicit dependency on the distribution parameters and write $\Pr\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right]$.

7.5.1 Update

On round i, the algorithm adjusts the distribution to ensure that the probability of a correct prediction for training instance i is no smaller than the confidence hyperparameter $\eta \in [0,1]$:

$$\Pr\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right] \geq \eta \ . \tag{7.4}$$

Following the intuition underlying the PA algorithms [55], our algorithm chooses the distribution closest in the KL divergence sense to the current distribution $\mathcal{N}\left(\boldsymbol{\mu}_i, \Sigma_i\right)$. Thus, on round i, the algorithm sets the parameters of the distribution by solving the following optimization problem:

$$\left(\boldsymbol{\mu}_{i+1}, \Sigma_{i+1}\right) = \min \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}\left(\boldsymbol{\mu}, \Sigma\right) \parallel \mathcal{N}\left(\boldsymbol{\mu}_i, \Sigma_i\right)\right) \tag{7.5}$$

$$\text{s.t. } \Pr\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right] \geq \eta \ . \tag{7.6}$$

We now develop both the objective and the constraint of this optimization problem following Boyd and Vandenberghe [24, page 158]. We start with the constraint (7.6). As noted above, under the distribution $\mathcal{N}\left(\boldsymbol{\mu}, \Sigma\right)$, the margin for $\left(\boldsymbol{x}_i, y_i\right)$ has a Gaussian distribution with mean $\mu_M = y_i\left(\boldsymbol{\mu}\cdot\boldsymbol{x}_i\right)$ and variance $\sigma_M^2 = \boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i$. Thus the probability of a *wrong* classification is

$$\Pr\left[M \leq 0\right] = \Pr\left[\frac{M - \mu_M}{\sigma_M} \leq \frac{-\mu_M}{\sigma_M}\right] \ .$$

Since $\left(M-\mu_M\right)/\sigma_M$ is a normally distributed random variable, the above probability equals $\Phi\left(-\mu_M/\sigma_M\right)$, where Φ is the cumulative function of the normal distribution. Thus we can rewrite (7.6) as

$$\frac{-\mu_M}{\sigma_M} \leq \Phi^{-1}\left(1-\eta\right) = -\Phi^{-1}\left(\eta\right) \ .$$

Substituting μ_M and σ_M by their definitions and rearranging terms we obtain:

$$y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i) \geq \phi\sqrt{\boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i}\ ,$$

where $\phi = \Phi^{-1}\left(\eta\right)$.

Unfortunately, this constraint is not convex in Σ, so we linearize it by omitting the square root:

$$y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i) \geq \phi\left(\boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i\right)\ . \tag{7.7}$$

Conceptually, this is a large-margin constraint, where the value of the margin requirement depends on the example $\boldsymbol{x}_i$ via a quadratic form.

We now study the objective (7.5). The KL divergence between two Gaussians is given by

$$\begin{aligned}\mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}\left(\boldsymbol{\mu}_0,\Sigma_0\right)\|\mathcal{N}\left(\boldsymbol{\mu}_1,\Sigma_1\right)\right) = \\ \frac{1}{2}\Bigg(\log\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right) + \mathrm{Tr}\left(\Sigma_1^{-1}\Sigma_0\right) \\ + \left(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0\right)^\top \Sigma_1^{-1}\left(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0\right) - d\Bigg)\ .\end{aligned} \tag{7.8}$$

Using the foregoing equations and omitting irrelevant constants, we obtain the following revised optimization problem:

$$\begin{aligned}\left(\boldsymbol{\mu}_{i+1},\Sigma_{i+1}\right) = \min \frac{1}{2}\log\left(\frac{\det\Sigma_i}{\det\Sigma}\right) + \frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right) \\ + \frac{1}{2}\left(\boldsymbol{\mu}_i-\boldsymbol{\mu}\right)^\top \Sigma_i^{-1}\left(\boldsymbol{\mu}_i-\boldsymbol{\mu}\right) \\ \text{s.t. } y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i) \geq \phi\left(\boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i\right)\ .\end{aligned} \tag{7.9}$$

The optimization objective is convex in $\boldsymbol{\mu}$ and Σ simultaneously and the constraint is linear, so any convex optimization solver could be used to solve this problem. We call the corresponding update *Variance-Exact*. However, for efficiency we prefer a closed-form approximate update that we call *Variance*. In this approximation, we allow the solution for Σ_{i+1} in (7.9) to produce (implicitly) a full matrix, and then project it to a diagonal matrix, where the non-zero off-diagonal entries are dropped. The Lagrangian for this optimization is

$$\begin{aligned}\mathcal{L} \;=\; & \frac{1}{2}\log\left(\frac{\det\Sigma_i}{\det\Sigma}\right)+\frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right)\\ & +\frac{1}{2}\left(\boldsymbol{\mu}_i-\boldsymbol{\mu}\right)^{\top}\Sigma_i^{-1}\left(\boldsymbol{\mu}_i-\boldsymbol{\mu}\right)\\ & +\alpha\left(-y_i\left(\boldsymbol{\mu}\cdot\boldsymbol{x}_i\right)+\phi\left(\boldsymbol{x}_i^{\top}\Sigma\boldsymbol{x}_i\right)\right)\;. \end{aligned} \tag{7.10}$$

At the optimum, we must have

$$\frac{\partial}{\partial\boldsymbol{\mu}}\mathcal{L}=\Sigma_i^{-1}\left(\boldsymbol{\mu}-\boldsymbol{\mu}_i\right)-\alpha y_i\boldsymbol{x}_i=0\;.$$

Assuming Σ_i is non-singular we get,

$$\boldsymbol{\mu}_{i+1}=\boldsymbol{\mu}_i+\alpha y_i\Sigma_i\boldsymbol{x}_i\;. \tag{7.11}$$

At the optimum, we must also have

$$\frac{\partial}{\partial\Sigma}\mathcal{L}=-\frac{1}{2}\Sigma^{-1}+\frac{1}{2}\Sigma_i^{-1}+\phi\alpha\boldsymbol{x}_i\boldsymbol{x}_i^{\top}=0\;.$$

Solving for Σ^{-1} we obtain

$$\Sigma_{i+1}^{-1}=\Sigma_i^{-1}+2\alpha\phi\boldsymbol{x}_i\boldsymbol{x}_i^{\top}\;. \tag{7.12}$$

Finally, we compute the inverse of (7.12) using the Woodbury identity [189, Eq. 135] and get,

$$\begin{aligned}\Sigma_{i+1} &= \left(\Sigma_i^{-1}+2\alpha\phi\boldsymbol{x}_i\boldsymbol{x}_i^{\top}\right)^{-1}\\ &= \Sigma_i-\Sigma_i\boldsymbol{x}_i\left(\frac{1}{2\alpha\phi}+\boldsymbol{x}_i^{\top}\Sigma_i\boldsymbol{x}_i\right)^{-1}\boldsymbol{x}_i^{\top}\Sigma_i\\ &= \Sigma_i-\Sigma_i\boldsymbol{x}_i\frac{2\alpha\phi}{1+2\alpha\phi\boldsymbol{x}_i^{\top}\Sigma_i\boldsymbol{x}_i}\boldsymbol{x}_i^{\top}\Sigma_i\;. \end{aligned} \tag{7.13}$$

The KKT conditions for the optimization imply that the either $\alpha = 0$, and no update is needed, or the constraint (7.7) is an equality after the udpate. Substituting (7.11) and (7.13) into the equality version of (7.7), we obtain:

$$y_i \left(\boldsymbol{x}_i \cdot \left(\boldsymbol{\mu}_i + \alpha y_i \Sigma_i \boldsymbol{x}_i \right) \right) \;=\; \phi \left(\boldsymbol{x}_i^\top \left(\Sigma_i - \Sigma_i \boldsymbol{x}_i \frac{2\alpha\phi}{1 + 2\alpha\phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i} \boldsymbol{x}_i^\top \Sigma_i \right) \boldsymbol{x}_i \right) . \quad (7.14)$$

Rearranging terms we get,

$$y_i \left(\boldsymbol{x}_i \cdot \boldsymbol{\mu}_i \right) \;+\; \alpha \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i \;=\; \phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i \;-\; \phi \left(\boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i \right)^2 \frac{2\alpha\phi}{1 + 2\alpha\phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i} \quad . \quad (7.15)$$

For simplicity, let $M_i = y_i \left(\boldsymbol{x}_i \cdot \boldsymbol{\mu}_i \right)$ be the mean margin and $V_i = \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i$ be the margin variance before the update. Substituting these into (7.15) we get,

$$M_i + \alpha V_i = \phi V_i - \phi V_i^2 \frac{2\alpha\phi}{1 + 2\alpha\phi V_i} \; .$$

It is straightforward to see that this is a quadratic equation in α. Its smaller root is always negative and thus is not a valid Lagrange multiplier. Let γ_i be its larger root:

$$\gamma_i = \frac{-(1{+}2\phi M_i) + \sqrt{(1{+}2\phi M_i)^2 {-} 8\phi \left(M_i {-} \phi V_i \right)}}{4\phi V_i} \; . \quad (7.16)$$

The constraint (7.7) is satisfied before the update if $M_i - \phi V_i \geq 0$. If $1 + 2\phi M_i \leq 0$, then $M_i \leq \phi V_i$ and from (7.16) we have that $\gamma_i > 0$. If, instead, $1 + 2\phi M_i \geq 0$, then, again by (7.16), we have

$$\begin{aligned} &\gamma_i > 0 \\ &\Leftrightarrow \sqrt{(1 + 2\phi M_i)^2 - 8\phi \left(M_i - \phi V_i \right)} > (1 + 2\phi M_i) \\ &\Leftrightarrow M_i < \phi V_i \; . \end{aligned}$$

From the KKT conditions, either $\alpha_i = 0$ or (7.9) is satisfied as an equality. In the later case, (7.14) holds, and thus $\alpha_i = \gamma_i > 0$. To summarize, we have proved the following:

Lemma 1 *The optimal value of the Lagrange multiplier is given by* $\alpha_i = \max \{\gamma_i, 0\}$.

Algorithm 1 Variance Algorithm (Approximate)

```
Input: confidence parameter $\phi = \Phi^{-1}(\eta)$
       initial variance parameter $a > 0$
Initialize: $\boldsymbol{\mu}_1 = \mathbf{0}$ , $\Sigma_1 = aI$
for $i = 1, 2 \ldots$ do
  Receive $\boldsymbol{x}_i \in \mathbb{R}^d$ , $y_i \in \{+1, -1\}$
  Set the following variables:
    $\alpha_i$ as in Lemma 1
    $\boldsymbol{\mu}_{i+1} = \boldsymbol{\mu}_i + \alpha_i y_i \Sigma_i \boldsymbol{x}_i$ (7.11)
    $\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha_i\phi$ diag $(\boldsymbol{x}_i)$ (7.17)
end for
```

The above derivation yields a full covariance matrix. As noted above, we restrict ourself to diagonal matrices and thus we project the solution into the set of diagonal matrices to get our approximation. In practice, it is equivalent to compute α_i as above but update with the following rule instead of (7.12).

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha\phi\text{diag}\left(\boldsymbol{x}_i\right) \ , \tag{7.17}$$

where diag $(\boldsymbol{x}_i)$ is a diagonal matrix with the square of the elements of $\boldsymbol{x}_i$ on the diagonal.

The pesudocode of the algorithm appears in Alg. 1. From the initalization of Σ_1 and the update rule of (7.12), we conclude that the eigenvalues of Σ_i are shirinking, but never set to zero explicltly, and thus the covariance matrices Σ_i are not singular.

7.6 Evaluation

We evaluated our Variance and Variance-Exact algorithms on three popular NLP datasets. Each dataset contains several binary classification tasks from which we selected a total of 12 problems, each contains a balanced mixture of instance labels.

20 Newsgroups The 20 Newsgroups corpus contains approximately 20,000 newsgroup messages, partitioned across 20 different newsgroups.[1] The dataset is a popular choice

[1] http://people.csail.mit.edu/jrennie/20Newsgroups/

comp	`comp.sys.ibm.pc.hardware`
	`comp.sys.mac.hardware`
sci	`sci.electronics`
	`sci.med`
talk	`talk.politics.guns`
	`talk.politics.mideast`

Table 7.1: 20 Newsgroups binary decision tasks.

for binary and multi-class text classification as well as unsupervised clustering. Following common practice, we created binary problems from the dataset by creating binary decision problems of choosing between two similar groups, as shown in Table 7.1. Each message was represented as a binary bag-of-words. For each problem we selected 1800 instances.

Reuters The Reuters Corpus Volume 1 (RCV1-v2/LYRL2004) contains over 800,000 manually categorized newswire stories [147]. Each article contains one or more labels describing its general topic, industry and region. We created the following binary decision tasks from the labeled documents: Insurance: Life (I82002) vs. Non-Life (I82003), Business Services: Banking (I81000) vs. Financial (I83000), and Retail Distribution: Specialist Stores (I65400) vs. Mixed Retail (I65600). These distinctions involve neighboring categories so they are fairly hard to make. Details on document preparation and feature extraction are given by Lewis *et al.* [147]. For each problem we selected 2000 instances using a bag of words representation with binary features.

Sentiment We obtained a larger version of the sentiment multi-domain dataset of Blitzer *et al.* [22] containing product reviews from 6 Amazon domains (book, dvd, electronics, kitchen, music, video). The goal in each domain is to classify a product review as either positive or negative. Feature extraction follows Blitzer *et al.*. For each problem we selected 2000 instances using uni/bi-grams with counts.

	Task	*PA*	*Variance*	*Variance-Exact*	*SVM*	*Maxent*	*SGD*
20 Newsgroups	comp	8.90	†**6.33**	9.63	∗7.67	∗7.62	7.36
	sci	4.22	†**1.78**	3.3	†3.51	†3.55	†4.77
	talk	1.57	1.09	2.21	**0.91**	**0.91**	1.36
Reuters	Business	17.80	17.65	17.70	⋆15.64	⋆**15.10**	⋆15.85
	Insurance	9.76	∗**8.45**	9.49	9.19	8.59	9.05
	Retail	15.41	†**11.05**	14.14	∗12.80	∗12.30	†14.31
Sentiment	books	19.55	∗**17.40**	20.45	†20.45	†19.91	∗19.41
	dvds	19.71	**19.11**	19.91	20.09	19.26	20.20
	electronics	17.40	†**14.10**	17.44	†16.80	†16.21	†16.81
	kitchen	15.64	∗**14.24**	16.35	15.20	14.94	∗15.60
	music	20.05	∗**18.10**	19.66	19.35	19.45	18.81
	videos	19.86	⋆**17.20**	19.85	†20.70	†19.45	⋆19.65
Spam	user 1	1.60	∗**0.90**	2.95	∗1.70	1.15	†2.30
	user 2	2.45	†**1.15**	1.75	†2.90	1.60	⋆1.95
	user 3	2.10	1.80	2.35	1.40	**1.35**	†7.40

Table 7.2: Error on test data using batch training. Statistical significance (McNemar) is measured against PA or the batch method against Variance. (∗ p=.05, ⋆ p=.01, † p=.001)

Spam We include a spam classification problem as a sample problem from the space of email classification problems. We chose spam since it is a widely studied problem with several publicly available datasets. We selected the 2006 ECML/PKDD Discovery Challenge spam dataset [18]. The goal is to classify an email (bag-of-words) as either spam or ham (not-spam.) This corpus contains two datasets: task A, which has three users, and task B, which has 15 users. We use the three users from task A since it has more instances for learning. For each user we select 2000 instances.

Each dataset was randomly divided for 10-fold cross validation experiments. Classifier parameters (ϕ for CW and C for PA) were tuned for each classification task on a single randomized run over the data. Results are reported for each problem as the average accuracy over the 10 folds. Statistical significance is computed using McNemar's test.

7.6.1 Results

We start by examining the performance of the Variance and Variance-Exact versions of our method, discussed in the preceding section, against a PA algorithm. All three algorithms were run on the datasets described above and each training phase consisted of five passes over the training data, which seemed to be enough to yield convergence. The average error on the test set for the three algorithms on all fifteen datasets is shown in table 7.2.

Variance-Exact achieved about the same performance as PA, with each method achieving a lower error on about half of the datasets. In contrast, Variance (approximate) significantly improves over PA, achieving lower error on all fifteen datasets, with statistically significant results on eleven of them.

As discussed above, online algorithms are attractive even for batch learning because of their simplicity and ability to operate on extremely large datasets. In the batch setting, these algorithms are run several times over the training data, which yields slower performance than single pass learning [37]. Our algorithm improves on both accuracy and learning speed by requiring fewer iterations over the training data. Such behavior can be seen on the "talk" dataset and others in Figure 7.1, which shows accuracy on test data after each iteration of the PA baseline and the two variance algorithms. While Variance clearly improves over PA, it converges very quickly, reaching near best performance on the first iteration. In contrast, PA benefits from multiple iterations over the data; its performance changes significantly from the first to fifth iteration. Across the fifteen tasks, Variance yields a 3.02% error reduction while PA gives a 10.24% reduction between the first and fifth iteration, indicating that multiple iterations help PA more. The plot also illustrates Variance-Exact's behavior, which initially beats PA but does not improve. In fact, on twelve of the fifteen datasets, Variance-Exact beats PA on the first iteration. The exact update results in aggressive behavior causing the algorithm to converge very quickly, even more so than Variance. It appears that the approximate update in Variance reduces overtraining and yields the best accuracy.

7.6.2 Batch Learning

While online algorithms are widely used, batch algorithms are still preferred for many tasks. Batch algorithms can make global learning decisions by examining the entire dataset, an ability beyond online algorithms. In general, when batch algorithms can be applied they perform better. We compare our new online algorithm (Variance) against two standard batch algorithms: maxent classification (default configuration of the maxent learner in MALLET [155]) and support vector machines (LibSVM [44]). We also include stochastic gradient descent (SGD) [22], which performs well for NLP tasks. Classifier parameters (Gaussian prior for maxent, C for SVM and the learning rate for SGD) were tuned as for the online methods. SGD was run for 10 iterations over the data.

Results for batch learning are shown in Table 7.2. As expected, the batch methods tend to do better than PA, with SVM doing better 10 times and maxent 14 times. SGD does better 11 times. However, in most cases Variance improves over the batch method, doing better than SVM and maxent 12 out of 15 times (at least 7 statistically significant.) Furthermore, it improves over SGD 14 out of 15 times. These results show that in these tasks, the much faster and simpler online algorithm performs better than the slower more complex batch methods.

We also evaluated the effects of commonly used techniques for online and batch learning, including averaging and TFIDF features, none of which improved accuracy. Although the above datasets are balanced with respect to labels and predictive features, we also evaluated the methods on variant datasets with unbalanced label or feature distributions, and still saw similar benefits from the Variance method.

7.7 Related Work

The idea of using parameter-specific variable learning rates has a long history in neural-network learning [233], although we do not know of a previous model that specifically models confidence in a way that takes into account the frequency of features. The second-order perceptron (SOP) [39] is perhaps the closest to our CW algorithm. Both are online algorithms that maintain a weight vector and some statistics about previous examples. While the SOP models certainty with feature counts, CW learning models uncertainty with a Gaussian distribution. CW algorithms have a probabilistic motivation, while the SOP is based on the geometric idea of replacing a ball around the input examples with a refined ellipsoid. Shivaswamy and Jebara [219] used this intuition in the context of batch learning.

Gaussian process classification (GPC) maintains a Gaussian distribution over weight vectors (primal) or over regressor values (dual). Our algorithm uses a different update criterion than the the standard Bayesian updates used in GPC [196, Ch. 3], avoiding the challenging issues in approximating posteriors in GPC. Bayes point machines [117] maintain a collection of weight vectors consistent with the training data, and use the single linear classifier which best represents the collection. Conceptually, the collection is a non-parametric distribution over the weight vectors. Its online version [114] maintains a finite number of weight-vectors updated simultaneously.

Finally, with the growth of available data there is an increasing need for algorithms that process training data very efficiently. A similar approach to ours is to train classifiers incrementally [23]. The extreme case is to use each example once, without repetitions, as in the multiplicative update method of Carvalho and Cohen [37].

7.8 Conclusion

We have presented confidence-weighted linear classifiers, a new learning method designed for NLP problems based on the notion of parameter confidence. The algorithm maintains a distribution over parameter vectors; online updates both improve the parameter estimates and reduce the distribution's variance. Our method improves over both online and batch methods and learns faster on a fifteen NLP datasets. In addition to improving learning performance, confidence-weighted classifiers have a per-parameter confidence score, as well as a confidence in the predicted margin. In the next chapter, we explore some applications that use this confidence.

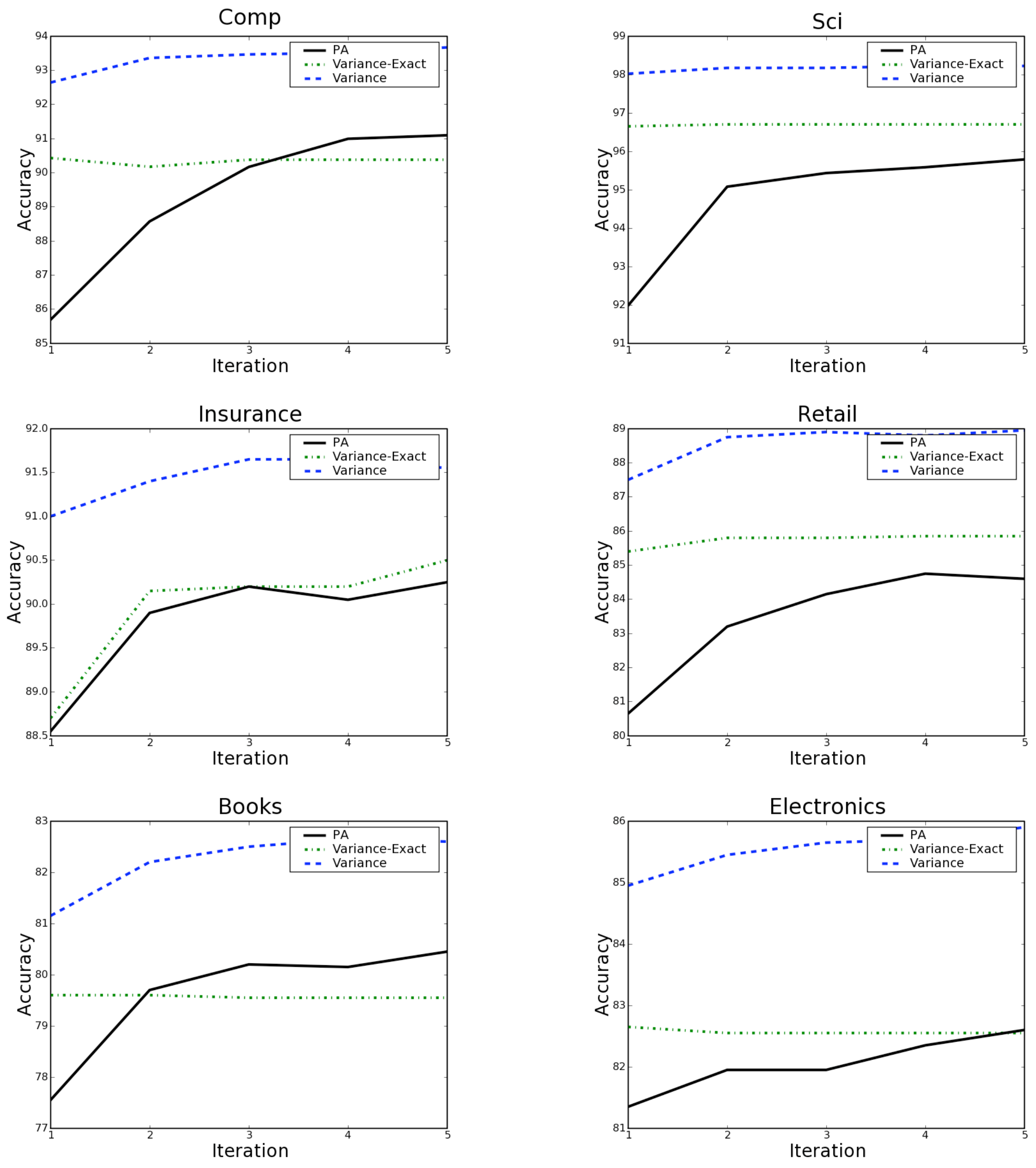


Figure 7.1: Accuracy on test data after each iteration on the several datasets. While PA continues to improve after the first iteration, the CW-Variance classifier tends to converge more quickly. On most datasets, Variance-Exact reaches optimal performance after a single iteration, although it does not perform as well as CW-Variance (approximate.)

Chapter 8

Confidence Based Applications of Confidence-Weighted Classifiers

8.1 Applications

One of the interesting aspects of confidence-weighted classifiers is a per-parameter confidence score, which translates into a confidence in the predicted margin. There are many potential learning applications that could benefit from a notion of confidence. In this chapter, we will explore some learning settings motivated by our intelligent email work and show how the confidence aspect of CW learning can be applied to address these problems.

Online algorithms are especially attractive for training on large amounts of data. Since they do not require access to all training instances, they can operate over a stream of training data, substantially reducing memory requirements and processing time. This property is especially attractive for email data. For large scale email providers with millions of users, the number of training examples for email problems could reach trillions of examples. We begin this chapter with experiments of Confidence-Weighted learning applied to a million training examples. We also consider how training can be spread across multiple processors using the learned confidence parameters. Our work in this section is based on previously published work in the International Conference on Machine Learning

(ICML) [85].

Returning to our example of reply prediction, there are two particular learning challenges to deploying such a system. As we saw, reply prediction requires labeled examples for learning. While our features created a representation useful for out of the box performance on new users, new training examples from a user can still improve the pre-trained classifier. However, obtaining labeled examples is expensive as it requires user interaction. If any deployed system were to solicit labeled examples, it would need to minimize the number of requested labels by using active learning. Active learning is a form of interactive learning, whereby a learning algorithm carefully selects examples from a pool of unlabeled examples to ask the user to label. Instead of labeling random examples, the user works with the learning algorithm to reduce the total labeling cost. We consider an application of active learning with online algorithms. Our work in this section is based on previously published work in the conference of the Association for Computational Linguistics (ACL) [83].

While we showed that our features can learn behaviors common across users for reply prediction, it cannot capture differences in user behaviors. For example, if users reply to email from coworkers with different frequency, then we cannot learn this behavior from a single user's labeled data. However, creating separate classification rules for each user ignores the commonality of behavior across users useful for learning. We formulate this as a new learning setting called multi-domain learning, whereby a learning algorithm receives examples from multiple users or domains and both learns common and domain specific behaviors. Additionally, we consider the case of domain adaptation, where classifiers from multiple users or domains are combined for a new domain. We also extend these methods to a setting with multiple disparate users and consider scaling these applications to a much larger number of domains. Our work in this section is based on previously published work in the conference on Empirical Methods for Natural Language Processing (EMNLP) [84].

8.2 Confidence Properties of CW Classifiers

In the previous chapter, we introduced confidence-weighted learning, where each parameter had an associated confidence value. Parameter confidence was formalized by a Gaussian distribution over weight vectors with mean $\boldsymbol{\mu} \in \mathbb{R}^N$ and diagonal covariance $\Sigma \in \mathbb{R}^{N \times N}$. The values μ_j and $\Sigma_{j,j}$ represent knowledge of and confidence in the parameter for feature j. The smaller $\Sigma_{j,j}$, the more confidence we have in the mean parameter value μ_j. As in the previous chapter, we consider diagonal covariance matrices to scale to NLP data.

A model predicts the highest probability label,

$$\arg\max_{y \in \{\pm 1\}} \Pr_{\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)} \left[y_i (\boldsymbol{w} \cdot \boldsymbol{x}_i) \geq 0 \right] . \tag{8.1}$$

The Gaussian distribution over parameter vectors $\boldsymbol{w}$ induces a univariate Gaussian distribution over the margin $m_i = \boldsymbol{w} \cdot \boldsymbol{x}_i$ parameterized by $\boldsymbol{\mu}$, Σ and the instance $\boldsymbol{x}_i$: $m_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, with mean $\mu_i = \boldsymbol{\mu} \cdot \boldsymbol{x}_i$ and variance $\sigma_i^2 = \boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i$. Normally, we select m_i to be the expectation of the distribution $\mathcal{N}(\mu_i, \sigma_i^2)$, i.e. $m_i = \boldsymbol{w} \cdot \boldsymbol{x}_i$. However, in some of the applications in this chapter, we consider the confidence in this margin as well: the variance of the margin's distribution.

8.2.1 Multi-Classifier Parameter Combination

Several of our applications consider combinations of CW classifiers. We define a combination method as a function that takes M CW classifiers, each parameterized by its own mean and variance parameters $\{(\boldsymbol{\mu}^m, \Sigma^m)\}_{m=1}^M$, and produces a single combined classifier $(\boldsymbol{\mu}^c, \Sigma^c)$. A simple technique would be to average the parameters of classifiers into a new classifier. However, this ignores the difference in feature distributions. Consider for example that the weight associated with some word in a source classifier has a value of 0. This could either mean that the word is very rare or that it is neutral for prediction (like the work "the"). The information captured by the variance parameter allows us to distinguish between the two cases: an high-variance indicates a lack of confidence in the value of the weight vectors because of small number of examples (first case), and vise-versa, small-variance indicates that the value of the weight is based on plenty of evidence. We favor combinations sensitive to this distinction.

Since CW classifiers are Gaussian distributions, we formalize classifier parameter combination as finding a new distribution that minimizes the weighted-divergence to a set of given distributions:

$$(\boldsymbol{\mu}^c, \Sigma^c) = \arg\min \sum_m^M \mathbb{D}((\boldsymbol{\mu}^c, \Sigma^c) || (\boldsymbol{\mu}^m, \Sigma^m) \ ; \ \mathbf{b}^m) \ , \tag{8.2}$$

where (since Σ is diagonal),

$$\mathbb{D}((\boldsymbol{\mu}^c, \Sigma^c) || (\boldsymbol{\mu}, \Sigma) \ ; \ \mathbf{b}) = \textstyle\sum_f^N b_f D((\boldsymbol{\mu}_f^c, \Sigma_{f,f}^c) || (\boldsymbol{\mu}_f, \Sigma_{f,f})) \ .$$

The (classifier specific) importance-weights $\mathbf{b}^m \in \mathbb{R}_+^N$ are used to weigh certain parameters of some classifiers differently in the combination. When $\mathbb{D}$ is the Euclidean distance (L2), we have,

$$D((\boldsymbol{\mu}_f^c, \Sigma_{f,f}^c) || (\boldsymbol{\mu}_f, \Sigma_{f,f})) = (\boldsymbol{\mu}_f^c - \boldsymbol{\mu}_f)^2 + (\Sigma_{f,f}^c - \Sigma_{f,f})^2 \ .$$

and we obtain:

$$\begin{aligned} \boldsymbol{\mu}_f^c &= \frac{1}{\sum_m^M b_f^m} \sum_m^M b_f^m \boldsymbol{\mu}_f^m, \\ \Sigma_{f,f}^c &= \frac{1}{\sum_{m \in M} b_f^m} \sum_m^M b_f^m \Sigma_{f,f}^m \ . \end{aligned} \tag{8.3}$$

Note that this is a (weighted) average of parameters. The other case we consider is when $\mathbb{D}$ is a weighted KL divergence we obtain a weighting of $\boldsymbol{\mu}$ by Σ^{-1}:

$$\begin{aligned} \boldsymbol{\mu}_f^c &= \left(\sum_m^M (\Sigma_{f,f}^m)^{-1} b_f^m \right)^{-1} \sum_m^M (\Sigma_{f,f}^m)^{-1} \boldsymbol{\mu}_f^m b_f^m \\ (\Sigma^c)^{-1} &= \left(M \sum_m^M b_f^m \right)^{-1} \sum_m^M (\Sigma_f^m)^{-1} {b_f}^m \ . \end{aligned} \tag{8.4}$$

While each parameter is weighed by its variance in the KL, we can also explicitly encode this behavior as $b_f^m = a - \Sigma_{f,f}^m \geq 0$, where a is the initialization value for $\Sigma_{f,f}^m$. We call this weighting "variance" as opposed to a uniform weighting of parameters ($b_f^m = 1$). We therefore have two combination methods (L2 and KL) and two weighting methods (uniform and variance).

8.3 Training on Large Datasets

Online algorithms are especially attractive in tasks where training data exceeds available main memory. However, even a single sequential pass over the data can be impractical for extremely large training sets, so we investigate training different models on different amounts of in parallel and combining the learned classifiers into a single classifier. While this often does not perform as well as a single model trained on all of the data, it is a cost effective way of learning from very large training sets.

We consider two large datasets for our evaluations. We obtained a large sample of product reviews from all the sentiment domains used in the previous chapter. We took a combined one million training examples and 10,000 test instances. These were randomized into four train and test sets. Our second dataset comes from the Reuters corpus used in the previous chapter. We created a one vs. all classification task for the *Corporate* topic label, yielding 804,411 instances of which 381,325 are labeled corporate. These were randomly split into 4 train and test sets, with 10K test instances. All parameters were optimized optimized by training on 5K random instances and testing on 10K.

The two datasets use different feature representations. The Reuters data contains unigrams that were thresholded for feature selection (see [147].) It contains 288,062 unique features, for a feature to document ratio of 0.36. In contrast, the sentiment contains raw unigrams and bigrams, so it has many more features. The data contains 13,460,254 unique features, a feature to document ratio of 13.33. This means that Reuters features tend to occur several times during training while many sentiment features occur only once.

We begin by evaluating a single model, both PA and CW-Variance, trained using a single pass over the entire data stream. The average accuracy on the test sets over the four runs are reported in Figure 8.1. On the sentiment data, CW does more than a point better than PA. However, on Reuters, the order is reversed. This may be due to the differences in the feature distributions for the two datasets, where Reuters has a low feature to document ratio.

Next, we evaluate the performance of a classifier trained in parallel across n processors. We randomly divide the data into n streams, where n is set as 10, 50 or 100. On each stream we train a separate model. After training, we combine all trained models into a single linear classifier for prediction on the test set. We evaluate three different combinations:

1. **Average:** Report the average performance of each of the n classifiers on the test data. This indicates the expected performance of taking a single one of the n models.

2. **Uniform:** The combined classifier using the L2 distance and a uniform weighting. We call this uniform since it is a uniform average of the parameters.

3. **Weighted:** The combined classifier using the KL distance and a uniform weighting. We call this weighted since the KL combination weights each classifier by its confidence.

The results for the three combination methods compared to single PA and the CW-Variance models are shown in Figure 8.1. Recall that for Reuters data, the PA single model achieves higher accuracy than Variance. However, combining 10 Variance classifiers achieves the best performance. For sentiment, combining 10 classifiers beats PA but is not as good as a single Variance model. In every case, combining the classifiers using either uniform or weighted improves over each model individually. On sentiment weighted combination improves over uniform combination and in Reuters the models are equivalent. This shows that using the confidence scores of CW classifiers enables better combinations for classifiers trained in parallel across multiple machines.

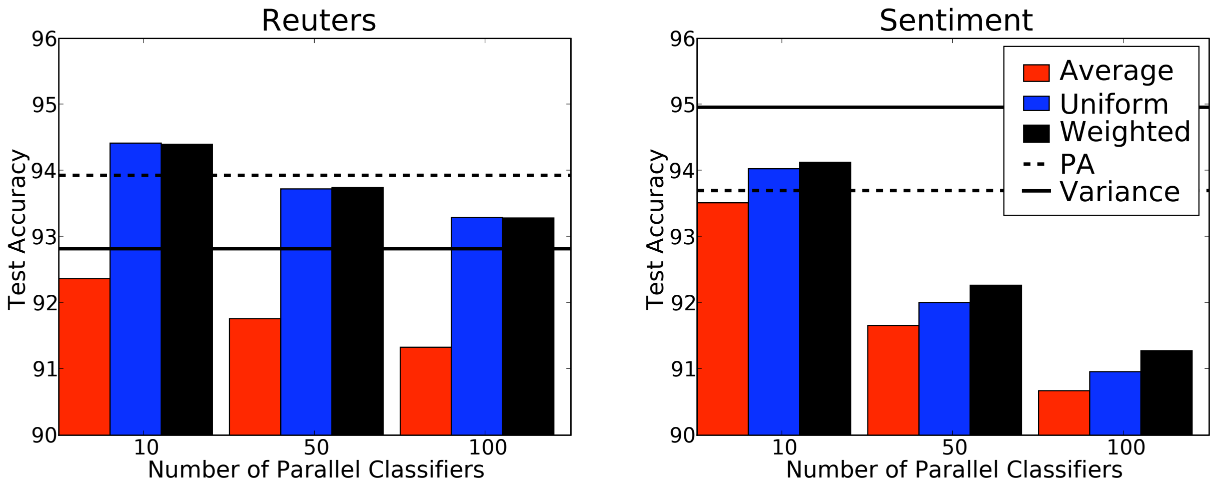


Figure 8.1: Results for Reuters (800k) and Sentiment (1000k) averaged over 4 runs. Horizontal lines show the test accuracy of a model trained on the entire training set. Vertical bars show the performance of n (10, 50, 100) classifiers trained on disjoint sections of the data as the average performance, L2 combination (uniform,) or KL combination (weighted.) All improvements are statistically significant except between uniform and weighted for Reuters.

Finally, we computed the actual run time of both PA and CW-Variance on the large datasets to compare the speed of each model. While CW-Variance is more complex, requiring more computation per instance, the actual speed is comparable to PA; in all tests the run time of the two algorithms was indistinguishable.

8.4 Active Learning for NLP

Successful applications of supervised machine learning to natural language rely on quality labeled training data, but annotation can be costly, slow and difficult. One popular solution is Active Learning, which maximizes learning accuracy while minimizing labeling efforts. In active learning, the learning algorithm itself selects unlabeled examples for annotation. A variety of techniques have been proposed for selecting examples that maximize system performance as compared to selecting instances randomly.

Two learning methodologies dominate NLP applications: probabilistic methods — naive Bayes, logistic regression — and margin methods — support vector machines and passive-aggressive. Active learning for probabilistic methods often uses uncertainty sampling: label the example with the lowest probability prediction (the most "uncertain") [146]. The equivalent technique for margin learning associates the margin with prediction certainty: label the example with the lowest margin [239]. Common intuition equates large margins with high prediction confidence.

However, confidence and margin are two distinct properties. For example, an instance may receive a large margin based on a single feature which has been updated only a small number of times. Another example may receive a small margin, but its features have been learned from a large number of examples. While the first example has a larger margin it has low confidence compared to the second. Both the margin value and confidence should be considered in choosing which example to label.

Confidence-weighted learning can be used in active learning to achieve both margin and confidence. Since the classifier assigns labels according to a Gaussian distribution over margin values instead of a single value, the variance of the distribution represents the confidence in the mean (margin). This can be employed to improve other margin based active learning techniques. Additionally, this is favorable since CW is an online algorithm, which increases the speed of the active learning cycle between user input and instance selection. In this section, we provide empirical support that margin based active learning improves with confidence.

8.4.1 Active Learning with Confidence

We consider pool based active learning. An active learning algorithm is given a pool of unlabeled instances $\mathcal{U} = \{\boldsymbol{x}_i\}_{i=1}^{n}$, a learning algorithm $\mathcal{A}$ and a set of labeled examples initially set to be $\mathcal{L} = \emptyset$. On each round the active learner uses its selection criteria to return a single instance $\boldsymbol{x}_i$ to be labeled by an annotator with $y_i \in \{-1, +1\}$ (for binary classification). The instance and label are added to the labeled set $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\boldsymbol{x}_i, y_i)\}$ and passed to the learning algorithm $\mathcal{A}$, which in turn generates a new model. At the end of labeling the algorithm returns a classifier trained on the final labeled set. Effective active learning minimizes prediction error and the number of labeled examples.

Most active learners for margin based algorithms rely on the magnitude of the margin. Tong and Koller [239] motivate this approach by considering the half-space representation of the hypothesis space for learning. They suggest three margin based active learning methods: Simple margin, MaxMin margin, and Ratio margin. In Simple margin, the algorithm predicts an unsigned margin M for each instance in $\mathcal{U}$ and returns for labeling the instance with the smallest margin. The intuition is that instances for which the classifier is uncertain (small margin) provide the most information for learning. Active learning based on PA algorithms runs in a similar fashion but full SVM retraining on every round is replaced with a single PA update using the new labeled example, greatly increasing learning speed.

Maintaining a distribution over prediction functions makes the CW algorithm attractive for active learning. As we showed above in section 8.2, instead of using a geometrical quantity (margin), it use a probabilistic quantity and picks the example whose label is predicted with the lowest probability. Formally, the margin criteria, $\boldsymbol{x} = \arg\min_{\boldsymbol{z}\in\mathcal{U}}(\boldsymbol{w}\cdot\boldsymbol{z})$, is replaced with a probabilistic criteria $\boldsymbol{x} = \arg\min_{\boldsymbol{z}\in\mathcal{U}} | \left(\Pr_{\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{\mu}_i,\Sigma_i)}\left[\mathrm{sign}(\boldsymbol{w}\cdot\boldsymbol{z}) = 1\right]\right) - \frac{1}{2}|$. The selection criteria naturally captures the notion that we should label the example with the highest uncertainty. Specifically, instead of selecting the example with the smallest margin m_i, we can normalize the margin using the standard deviation σ_i to get a corrected geometric margin as $\mathcal{M} = m_i/\sigma_i$. We then select the example with the smallest corrected margin $\mathcal{M}$. We call this selection criteria Active Confident Learning (ACL).

We can also consider the probabilistic values of the margin. As shown above, the Gaussian distribution over parameter vectors $\boldsymbol{w}$ induces a univariate Gaussian distribution over the margin $m_i = \boldsymbol{w}\cdot\boldsymbol{x}_i$ parameterized by $\boldsymbol{\mu}$, Σ and the instance $\boldsymbol{x}_i$: $m_i \sim \mathcal{N}\left(\mu_i, \sigma_i^2\right)$, with mean $\mu_i = \boldsymbol{\mu}\cdot\boldsymbol{x}_i$ and variance $\sigma_i^2 = \boldsymbol{x}_i^\top\Sigma\boldsymbol{x}_i$. We can select the instance that has the smallest probability associated with its prediction, $\arg\min_{\boldsymbol{x}_i} P(y|\boldsymbol{x}_i)$. The probability associated with a prediction is computed using the cumulative distribution function of the Gaussian $\Phi(x)$. For a specific instance $\boldsymbol{x}_i$, the value of the function is $\Phi(\frac{\boldsymbol{\mu}\cdot\boldsymbol{x}_i}{\sqrt{\boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i}})$. The CDF of the Gaussian $\Phi(x)$ is a monotonic function with $\Phi(0) = \frac{1}{2}$. Selecting the lowest probability prediction is found by minimizing this function, which is achieved by choosing the example $\boldsymbol{x}_i$ such that the ratio $\frac{m_i}{\sigma_i}$ is minimized. This is equivalent to the objective we showed for Active Confident Learning. Therefore, ACL and the least probable prediction are equivalent selection techniques.

8.4.2 Evaluation

To evaluate our active learning methods we used a similar experimental setup to Tong and Koller [239]. Each active learning algorithm was given two labeled examples, one from each class, for initial training of a classifier, and remaining data as unlabeled examples. On each round the algorithm selected a single instance for which it was then given the correct label. The algorithm updated the online classifier and evaluated it on held out test data to measure learning progress.

We used the four binary NLP tasks from the previous chapter for evaluation: 20 Newsgroups, Reuters (used by [239]), sentiment classification and spam. For sentiment we used four domains (book, dvd, electronics, kitchen). Each problem had 2000 instances except for 20 Newsgroups, which used between 1850 and 1971 instances. This created 13 classification problems across four tasks.

Each active learning algorithm was evaluated using a PA (with slack variable $c = 1$) or CW (variance) classifier ($\phi = 1$) using 10-fold cross validation. We evaluated several methods in the Simple margin framework: PA Margin and CW Margin, which select examples with the smallest margin, and active confident learning (ACL). As a baseline we included selecting a random instance. We also evaluated CW and a PA classifier trained on all training instances. Each method was evaluated by labeling up to 500 labels, about 25% of the training data.

The 10 runs on each dataset for each problem appear in the four plots in figure 8.2, where the 13 datasets are grouped by task type (20 Newsgroups, Reuters, Sentiment, Spam.) Each plot shows the test accuracy after each round of active learning. Horizontal lines indicate CW (solid) and PA (dashed) training on all instances. Legend numbers are accuracy after 500 labels.

To achieve 80% of the accuracy of training on all data, a realistic goal for less than 100 labels, PA Margin required 93% the number of labels of PA Random, while CW Margin needed only 73% of the labels of CW Random. By using fewer labels compared to random selection baselines, CW Margin learns faster in the active learning setting as compared with PA. Furthermore, adding confidence reduced labeling cost compared to margin alone. ACL improved over CW Margin on every task and after almost every round; it required 63% of the labels of CW Random to reach the 80% mark.

We computed the fraction of labels CW Margin and ACL required (compared to CW Random) to achieve the 80% accuracy mark of training with all data. The results are summarized in the right panel of figure 8.3, where we plot one point per dataset. Points above the diagonal-line demonstrate the superiority of ACL over CW Margin. ACL required fewer labels than CW margin twice as often as the opposite occurred (8 vs 4). Note that CW Margin used *more* labels than CW Random in three cases, while ACL only once, and this one time only about a dozen labels were needed. To conclude, not only does CW Margin outperforms PA Margin for active-learning, CW maintains additional valuable information (confidence), which further improves performance.

8.4.3 Related Work

Active learning has been widely used for NLP tasks such as part of speech tagging [200], parsing [235] and word sense disambiguation [43]. Many methods rely on entropy-based scores such as uncertainty sampling [146]. Others use margin based methods, such as Kim *et al.* [131], who combined margin scores with corpus diversity, and Sassano [208], who considered SVM active learning for Japanese word segmentation. Our confidence based approach can be used to improve these tasks. Furthermore, margin methods can

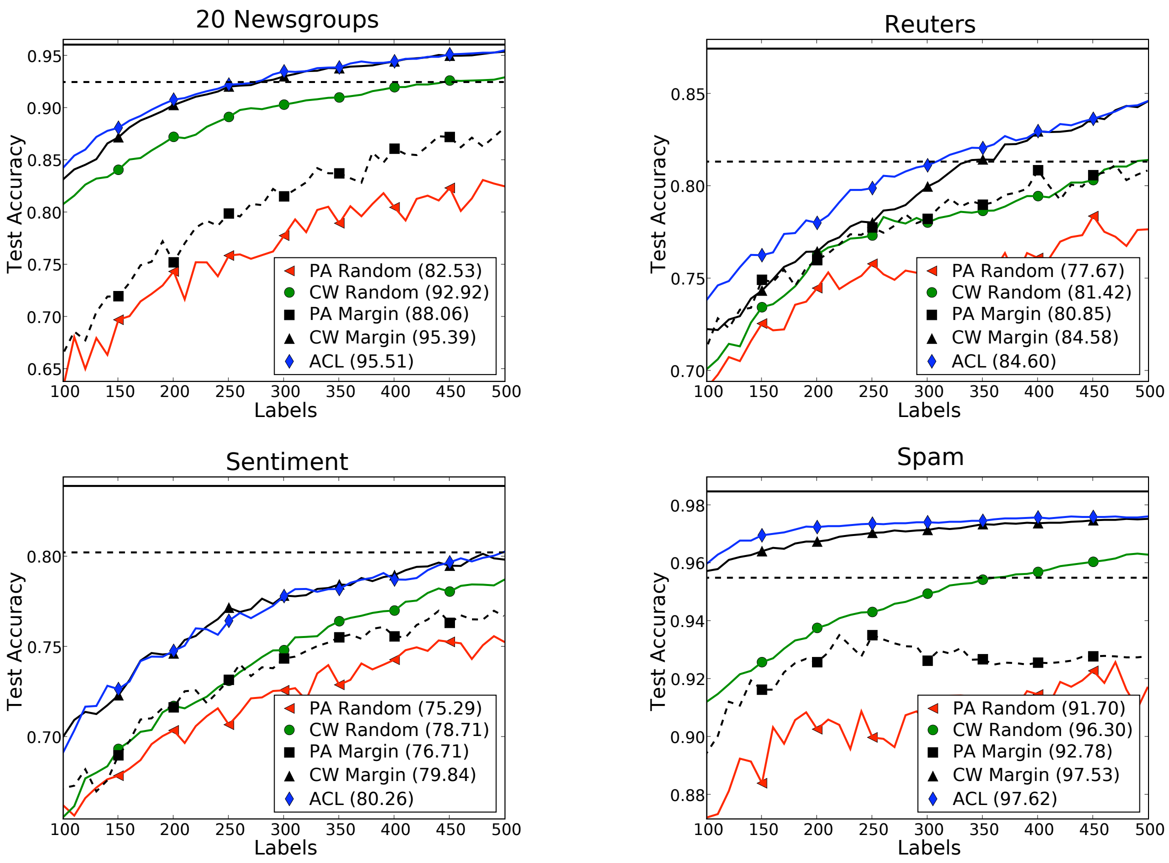


Figure 8.2: Results averaged over each task showing test accuracy over active learning rounds. The numbers in parenthesis in the legends show test accuracy after 500 rounds of learning.

outperform probabilistic methods; CW beats maximum entropy on many NLP tasks as we demonstrated in the previous chapter.

A theoretical analysis of margin based methods selected labels that maximize the reduction of the version space, the hypothesis set consistent with the training data [239]. Another approach selects instances that minimize the future error in probabilistic algorithms [203]. Since we consider an online learning algorithm our techniques can be easily extended to online active learning, which has been theoretically analyzed [42, 69] and evaluated [210].

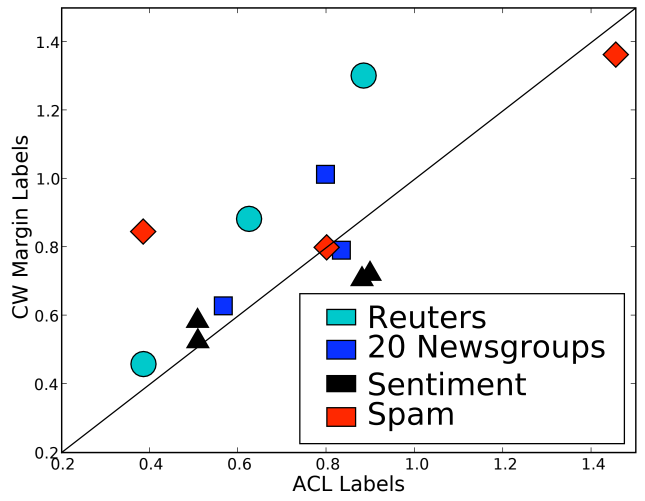


Figure 8.3: A scatter plot showing the amount of labels needed by CW Margin and ACL to achieve 80% of the accuracy of training on all data. Each points refers to one of the twelve dataset.

8.5 Multi-Domain Learning for NLP

Statistical classifiers routinely process millions of websites, emails, blogs and other text every day. Variability across different data sources means that training a single classifier obscures differences and separate classifiers ignore similarities. Similarly, adding new domains to existing systems requires adapting existing classifiers.

In this section we present new online algorithms for three *multi-domain learning* scenarios: adapting existing classifiers to new domains, learning across multiple similar domains and scaling systems to many disparate domains. Multi-domain learning combines characteristics of both multi-task learning and domain adaptation and drawing from both areas, we develop a multi-classifier parameter combination technique for confidence-weighted linear classifiers.

In online multi-domain learning, each instance $\boldsymbol{x}$ is drawn from a domain d specific distribution $\boldsymbol{x} \sim \mathcal{D}_d$ over a vectors space $\mathbb{R}^N$ and labeled with a domain specific function f_d with label $y \in \{-1, +1\}$ (for binary classification.) On round i the classifier receives instance $\boldsymbol{x}_i$ and domain identifier d_i and predicts label $\hat{y}_i \in \{-1, +1\}$. It then receives the true label $y_i \in \{-1, +1\}$ and updates its prediction rule.

As an example, consider a multi-user spam filter, which must give high quality predictions for new users (without new user data), learn on multiple users simultaneously and scale to thousands of accounts. While a single classifier trained on all users would generalize across users and extend to new users, it would fail to learn user-specific preferences. Alternatively, separate classifiers would capture user-specific behaviors but would not generalize across users. The approach we take to solving multi-domain problems is to combine domain-specific classifiers. In the adaptation setting, we combine source domain classifiers for a new target domain. For learning across domains, we combine domain-specific classifiers and a shared classifier learned across all domains. For learning across disparate domains we learn which domain-specific and shared classifiers to combine.

Multi-domain learning combines properties of both multi-task learning and domain adaptation. As in multi-task learning, we consider domains that are labeled with different classification functions. For example, one user may enjoy some emails that another user considers spam: differing in their classification function. The goal of multi-task learning is to generalize across tasks/domains [72, 97]. Furthermore, as in domain adaptation, some examples are draw from different distributions. For example, one user may receive emails about engineering while another about art, differing in their distribution over features. Domain adaptation deals with these feature distribution changes [22,124]. Our work combines these two areas by learning both across distributions and behaviors or functions.

8.5.1 Datasets

For the evaluation of our algorithms in each multi-domain learning setting, we selected two domain adaptation datasets: spam [124] and sentiment [22]. Both of these datasets were used in the previous chapter to evaluate CW learning. As a reminder, the spam data contains two tasks, one with three users (task A) and one with 15 (task B). The goal is to classify an email (bag-of-words) as either spam or ham (not-spam) and each user may have slightly different preferences and features. We used 700 and 100 training messages for each user for task A and B respectively and 300 test emails for each user.

The sentiment data contains product reviews from Amazon for seven product types: books, dvds, electronics, kitchen appliances, apparel, music and videos. We created different datasets by modifying the decision boundary using the ordinal rating of each instance (1-5 stars) and excluding boundary instances. We use four versions of this data:

- **All** - 7 domains, one per product type
- **Books** - 3 domains of books with the binary decision boundary set to 2, 3 and 4 stars
- **DVDs** - Same as *Books* but with DVD reviews
- **Books+DVDs** - Combined *Books* and *DVDs*

The *All* dataset captures the typical domain adaptation scenario, where each domain has the same decision function but different features. *Books* and *DVDs* have the opposite problem: the same features but different classification boundaries. *Books+DVDs* combines both issues. Experiments use 1500 training and 100 test instances per domain.

8.5.2 Multi-Domain Adaptation

We begin by examining the typical domain adaptation scenario, but from an online perspective since learning systems often must adapt to new users or domains quickly and with no training data. For example, a spam filter with separate classifiers trained on each user must also classify mail for a new user. Since other user's training data may have been deleted or be private, the existing classifiers must be combined for the new user.

We combine the existing user-specific classifiers into a single new classifier for a new user. Since nothing is known about the new user (their decision function), each source classifier may be useful. However, feature similarity – possibly measured using unlabeled data – could be used to weigh source domains. Specifically, we combine the parameters of each classifier according to their confidence using the combination methods described above.

We evaluated the four combination strategies – L2 vs. KL, uniform vs. variance – on spam and sentiment data. For each evaluation, a single domain was held out for testing while separate classifiers were trained on each source domain, i.e. no target training. Source classifiers are then combined and the combined classifier is evaluated on the test data (400 instances) of the target domain. Each classifier was trained for 5 iterations over the training data (to ensure convergence) and each experiment was repeated using 10-fold cross validation. The CW parameter ϕ was tuned on a single randomized run for each experiment. We include several baselines:

- **Target:** Train on target data. This would be the best strategy if labeled target data were available.
- **All Src:** Train on all source domains together, a useful strategy if all source data is maintained. This also benefits from the increased data since all training examples are pooled together.
- **Best Src:** Select (with omniscience) the best performing source classifier on target data. This strategy is possible if one knew the similarity between the existing domains and a new domain.
- **Avg Src:** The average performance of the source classifiers. This is the expected real world performance of selecting a classifier at random from existing classifiers to use for the new domain.

While at least one source classifier achieved high performance on the target domain (*Best Src*), the correct source classifier cannot be selected without target data. The realistic strategy, selecting a random classifier (*Avg Src*), yields high error. In contrast, a combined classifier almost always improved over the best source domain classifier (table 8.1). That some of our results improve over the best training scenario is likely caused by increased training data from using multiple domains. Increases over all available training data are very interesting and may be due to a regularization effect of training separate models, similar to the effect we saw above in training classifiers in parallel.

	Target Domain	Train				L2		KL	
		All Src	*Target*	*Best Src*	*Avg Src*	*Uni*	*Var*	*Uni*	*Var*
Spam	user0	3.85	1.80	4.80	8.26	5.25	4.63	4.53	**4.32**
	user1	3.57	3.17	4.28	6.91	4.53	**3.80**	4.23	3.83
	user2	3.30	2.40	3.77	5.75	4.75	**4.60**	4.93	4.67
Sentiment	apparel	12.32	12.02	14.12	21.15	14.03	**13.18**	13.50	13.48
	books	16.85	18.95	22.95	25.76	19.58	**18.63**	19.53	19.05
	dvd	13.65	17.40	17.30	21.89	15.53	**13.73**	14.48	14.15
	kitchen	13.65	14.40	15.52	22.88	16.68	15.10	14.78	**14.02**
	electronics	15.00	14.93	15.52	23.84	18.75	17.37	17.45	**16.82**
	music	18.20	18.30	20.75	24.19	18.38	**17.83**	18.10	18.22
	video	17.00	19.27	19.43	25.78	17.13	**16.25**	16.33	16.42

Table 8.1: Test error on sentiment and spam data for multi-source adaptation using both KL and L2 combinations with uniform (*uni*) and variance (*var*) weightings. Combining classifiers improves over selecting a single classifier a priori (*Avg Src*). The L2 methods performed best and KL improves 7 out of 10 combinations.

The L2 methods performed best and KL improved 7 out of 10 combinations. Classifier parameter combination can clearly yield good classifiers without prior knowledge of the target domain.

8.5.3 Learning Across Domains

In addition to adapting to new domains, multi-domain systems should learn common behaviors across domains. Naively, we can assume that the domains are either sufficiently similar to warrant one classifier or different enough for separate classifiers. The reality is often more complex. Instead, we maintain shared and domain-specific parameters and combine them for learning and prediction.

Multi-task learning aims to learn common behaviors across related problems, a similar goal to multi-domain learning. The primary difference is the nature of the domains/tasks: in our setting each domain is the same task but differs in the types of features in addition to the decision function. A multi-task approach can be adapted to our setting by using our classifier combination techniques.

We seek to learn domain specific parameters guided by shared parameters. Dekel *et al.* [72] followed this approach for an online multi-task algorithm, although they did not have shared parameters and assumed that a training round comprised an example from each task. Evegniou and Pontil [97] achieved a similar goal by using shared parameters for multi-task regularization. Specifically, they assumed that the weight vector for problem d could be represented as $w^c = w^d + w^s$, where w^d are task specific parameters and w^s are shared across all tasks. In this framework, all tasks are close to some underlying mean w^s and each one deviates from this mean by w^d. Their SVM style multi-task objective minimizes the loss of w^c and the norm of w^d and w^s, with a tradeoff parameter allowing for domain deviance from the mean. The simple domain adaptation algorithm of feature splitting used by Daum [70] is a special case of this model where the norms are equally weighted. An analogous CW objective is:

$$\begin{aligned}\min \frac{1}{\lambda_1} \mathrm{D}_{\mathrm{KL}} &\left(\mathcal{N}\left(\boldsymbol{\mu}^d, \Sigma^d\right) \| \mathcal{N}\left(\boldsymbol{\mu}_i^d, \Sigma_i^d\right)\right) \\ &+ \frac{1}{\lambda_2} \mathrm{D}_{\mathrm{KL}} \left(\mathcal{N}\left(\boldsymbol{\mu}^s, \Sigma^s\right) \| \mathcal{N}\left(\boldsymbol{\mu}_i^s, \Sigma_i^s\right)\right) \\ \text{s.t. } \Pr_{\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c)} &\left[y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i\right) \geq 0\right] \geq \eta \,. \end{aligned} \tag{8.5}$$

$\left(\boldsymbol{\mu}^d, \Sigma^d\right)$ are the parameters for domain d, $\left(\boldsymbol{\mu}^s, \Sigma^s\right)$ for the shared classifier and $\left(\boldsymbol{\mu}^c, \Sigma^c\right)$ for the combination of the domain and shared classifiers. The parameters are combined via (8.4) with only two elements summed - one for the shared parameters s and the other for the domain parameters d . This captures the intuition of Evgeniou and Pontil: updates enforce the learning condition on the combined parameters and minimize parameter change. For convenience, we rewrite $\lambda_2 = 2 - 2\lambda_1$, where $\lambda_1 \in [0, 1]$. If classifiers are combined using the sum of the individual weight vectors and $\lambda_1 = 0.5$, this is identical to feature splitting (Daumé) for CW classifiers.

The domain specific and shared classifiers can be updated using the closed form solution to (8.5) as:

$$\begin{aligned}
\boldsymbol{\mu}^s &= \boldsymbol{\mu}_i^s + \lambda_2 \alpha y_i \Sigma_i^c \boldsymbol{x}_i \\
(\Sigma^s)^{-1} &= (\Sigma_i^s)^{-1} + 2\lambda_2 \alpha \phi \boldsymbol{x}_i \boldsymbol{x}_i^T \\
\boldsymbol{\mu}^d &= \boldsymbol{\mu}_i^d + \lambda_1 \alpha y_i \Sigma_i^c \boldsymbol{x}_i \\
(\Sigma^d)^{-1} &= (\Sigma_i^d)^{-1} + 2\lambda_1 \alpha \phi \boldsymbol{x}_i \boldsymbol{x}_i^T
\end{aligned} \tag{8.6}$$

We call this objective Multi-Domain Regularization (MDR).[1] As before, the combined parameters are produced by one of the combination methods. On each round, the algorithm receives instance $\boldsymbol{x}_i$ and domain d_i for which it creates a combined classifier $(\boldsymbol{\mu}^c, \Sigma^c)$ using the shared $(\boldsymbol{\mu}^s, \Sigma^s)$ and domain specific parameters $\left(\boldsymbol{\mu}^d, \Sigma^d\right)$. A prediction is issued using the standard linear classifier prediction rule $\text{sign}(\boldsymbol{\mu}^c \cdot \boldsymbol{x})$ and updates follow (8.6). The effect is that features similar across domains quickly converge in the shared classifier, sharing information across domains. The combined classifier reflects shared and domain specific parameter confidences: weights with low variance (i.e. greater confidence) will contribute more.

We evaluate MDR on a single pass over a stream of instances from multiple domains, simulating a real world setting. Parameters λ_1 and ϕ are iteratively optimized on a single randomized run for each dataset. All experiments use 10-fold CV. In addition to evaluating the four combination methods with MDR, we evaluate several baselines:

- **Single:** A single classifier trained on all domains. This method does best when domains are similar and there are few differences between how domains treat the same features.

[1]Note that this is not the exact solution to the objective cited above. Instead, to obtain a closed form solution we apply the standard CW objective to a combined classifier, minimizing the KL divergence between the combined classifiers before and after the update. Averaging the update to each of the sets of parameters yields the given closed form solution.

	Spam		*Sentiment*			
Method	*Task A*	*Task B*	*Books*	*DVD*	*Books+DVD*	*All*
Single	3.88	8.75	23.7	25.11	23.26	16.57
Separate	5.46	14.53	22.22	21.64	21.23	21.89
Feature Splitting	4.16	8.93	15.65	16.20	14.60	17.45
MDR	4.09	9.18	15.65	15.12	13.76	17.45
MDR+L2	4.27	8.61	**12.70**	14.95	12.73	**17.16**
MDR+L2-Var	**3.75**	**7.52**	12.90	14.21	**12.52**	17.37
MDR+KL	4.32	9.22	13.51	**13.81**	13.32	17.20
MDR+KL-Var	4.02	8.70	14.93	14.03	14.22	18.40

Table 8.2: Online training error for learning across domains.

- **Separate:** Learn a separate classifier for each domain individually. This method can best capture each individual domain's behavior but cannot learn across domains. When domains are mostly similar, this approach will not be able to utilize the increased data from pooling all domains together.

- **Feature Splitting:** The feature splitting method of [70]. Each feature is split into a shared feature and a domain specific feature, one for each domain. This is the same as our method with $\lambda_1 = .5$.

- **MDR:** Conceptually the same as *Feature Splitting* only that updates are biased towards the domain specific or shared features using λ_1. In this case, $\lambda_1 = .5$ is optimized.

Table 8.3 shows results on test data and table 8.2 shows online training error.

In this setting, L2 combinations prove best on 5 of 6 datasets, with the variance weighted combination doing the best. MDR (optimizing λ_1) slightly improves over feature splitting, and the combination methods improve in every case. Our best result is statistically significant compared to *Feature Splitting* using McNemar's test ($p = .001$) for *Task B*, *Books*, *DVD*, *Books+DVD*. While a single or separate classifiers have a different effect on each dataset, MDR gives the best performance overall.

	Spam		*Sentiment*			
Method	*Task A*	*Task B*	*Books*	*DVD*	*Books+DVD*	*All*
Single	2.11	5.60	18.43	18.67	19.08	14.09
Separate	2.43	8.5	18.87	15.97	16.45	17.23
Feature Splitting	1.94	5.51	9.97	9.70	9.05	14.73
MDR	1.94	5.69	9.97	8.33	8.20	14.73
MDR+L2	**1.87**	5.16	6.63	7.97	7.62	**14.20**
MDR+L2-Var	1.90	**4.78**	**6.40**	7.83	**7.30**	14.33
MDR+KL	1.94	5.61	8.37	**7.07**	8.43	14.60
MDR+KL-Var	1.97	5.46	9.40	7.50	8.05	15.50

Table 8.3: Test data error: learning across domains (MDR) improves over the baselines and Daum [70].

8.5.4 Learning in Many Domains

So far we have considered settings with a small number of similar domains. While this is typical of multi-task problems, real world settings present many domains which do not all share the same behaviors. Online algorithms scale to numerous examples and we desire the same behavior for numerous domains. Consider a spam filter used by a large email provider, which filters billions of emails for millions of users. Suppose that spammers control many accounts and maliciously label spam as legitimate. Alternatively, subsets of users may share preferences. Since behaviors are not consistent across domains, shared parameters cannot be learned. We seek algorithms robust to this behavior.

Since subsets of users share behaviors, these can be learned using our MDR framework. For example, discovering spammer and legitimate mail accounts would enable intra-group learning. The challenge is the online discovery of these subsets while learning model parameters. We augment the MDR framework to additionally learn this mapping.

We begin by generalizing MDR to include k shared classifiers instead of a single set of shared parameters. Each set of shared parameters represents a different subset of domains. If the corresponding shared parameters are known for a domain, we could use the same objective (8.5) and update (8.6) as before. If there are many fewer shared parameters than domains ($k \ll D$), we can benefit from multi-domain learning. Next, we augment the learning algorithm to learn a mapping between the domains and shared classifiers. Intuitively, a domain should be mapped to shared parameters that correctly classify that domain. A common technique for learning such experts in the Weighted Majority algorithm [150], which weighs a mixture of experts (classifiers). However, since we require a hard assignment — pick a single shared parameter set s — rather than a mixture, the algorithm reduces to picking the classifier s with the fewest mistakes in predicting domain d. This requires tracking the number of mistakes made by each shared classifier on each domain once a label is revealed. For learning, the shared classifier with the fewest mistakes for a domain is selected for an MDR update. Classifier ties are broken randomly. While we experimented with more complex techniques, this simple method worked well in practice. When a new domain is added to the system, it takes fewer examples to learn which shared classifier to use instead of learning a new model from scratch.

While this approach adds another free parameter (k) that can be set using development data, we observe that k can instead be fixed to a large constant. Since only a single shared classifier is updated each round, the algorithm will favor selecting a previously used classifier as opposed to a new one, using as many classifiers as needed but not scaling up to k. This may not be optimal, but it is a simple.

To evaluate a larger number of domains, we created many varying domains using spam and sentiment data. For spam, 6 email users were created by splitting the 3 task A users into 2 users, and flipping the label of one of these users (a malicious user), yielding 400 train and 100 test emails per user. For sentiment, the book domain was split into 3 groups with binary boundaries at a rating of 2, 3 or 4. Each of these groups was split into 8 groups of which half had their labels flipped, creating 24 domains. The same procedure was repeated for DVD reviews but for a decision boundary of 3, 6 groups were created, and for a boundary of 2 and 4, 3 groups were created with 1 and 2 domains flipped respectively, resulting in 12 DVD domains and 36 total domains with various decision boundaries, features, and inverted decision functions. Each domain used 300 train and 100 test instances. 10-fold cross validation with one training iteration was used to train models on these two

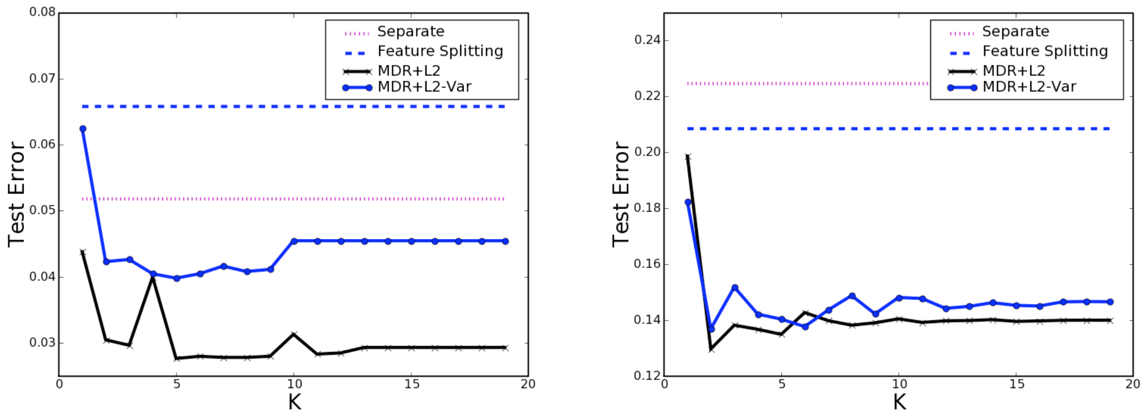


Figure 8.4: Learning across many domains - spam (left) and sentiment (right) - with MDR using k shared classifiers.

datasets. Parameters were optimized as before. Experiments were repeated for various settings of k. Since L2 performed well before, we evaluated MDR+L2 and MDR+L2-Var.

The results are shown in figure 8.4. For both spam and sentiment adding additional shared parameters beyond the single shared classifier significantly reduces error, with further reductions as k increases. This yields a 45% error reduction for spam and a 38% reduction for sentiment over the best baseline. While each task has an optimal k (about 5 for spam, 2 for sentiment), larger values still achieve low error, indicating the flexibility of using large k values.

While adding parameters clearly helps for many domains, it may be impractical to keep domain-specific classifiers for thousands or millions of domains. In this case, we could eliminate the domain-specific classifiers and rely on the k shared classifiers only, learning the domain to classifier mapping. We compare this approach using the best result from MDR above, again varying k. Figure 8.5 shows that losing domain-specific parameters hurts performance, but is still an improvement over baseline methods. Additionally, we can expect better performance as the number of similar domains increases. This may be an attractive alternative to keeping a very large number of parameters.

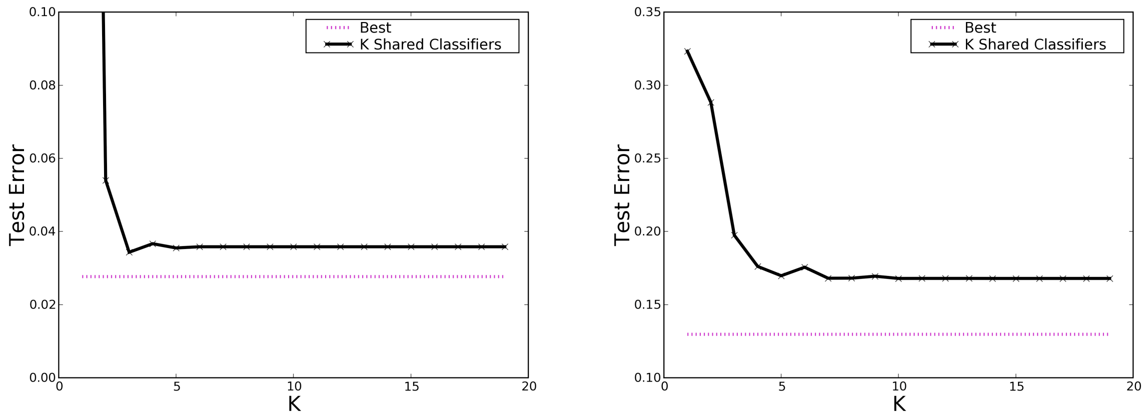


Figure 8.5: Learning across many domains - spam (left) and sentiment (right) - with no domain specific parameters.

8.5.5 Related Work

Multi-domain learning intersects two areas of research: domain adaptation and multi-task learning. In domain adaptation, a classifier trained for a source domain is transfered to a target domain using either unlabeled or a small amount of labeled target data. Blitzer *et al.* [22] used structural correspondence learning to train a classifier on source data with new features induced from target unlabeled data. In a complimentary approach, Jiang and Zhai [124] weighed training instances based on their similarity to unlabeled target domain data. Several approaches utilize source data for training on a limited number of target labels, including feature splitting [70] and adding the source classifier's prediction as a feature [45]. Others have considered transfer learning, in which an existing domain is used to improve learning in a new domain, such as constructing priors [154, 194] and learning parameter functions for text classification from related data [76]. These methods largely require batch learning, unlabeled target data, or available source data at adaptation. In contrast, our algorithms operate purely online and can be applied when no target data is available.

Multi-task algorithms, also known as inductive transfer, learn a set of related problems simultaneously [33]. The most relevant approach is that of Regularized Multi-Task Learning [97], which we use to motivate our online algorithm. Dekel *et al.* [72] gave a similar online approach but did not use shared parameters and assumed multiple instances for each round. We generalize this work to both include an arbitrary classifier combination and many shared classifiers. Some multi-task work has also considered the grouping of tasks similar to our learning of domain subgroups [4, 238].

There are many techniques for combining the output of multiple classifiers for ensemble learning or mixture of experts. Kittler *et al.* [134] provide a theoretical framework for combining classifiers. Some empirical work has considered adding versus multiplying classifier output [236], using local accuracy estimates for combination [253], and applications to NLP tasks [100]. However, these papers consider combining classifier output for prediction. In contrast, we consider parameter combination for both prediction and learning.

8.6 Conclusion

This chapter has explored several applications of parameter confidence to different learning tasks. All three of these applications are realistic challenges for deploying NLP systems in the wild. Furthermore, they are all particularly applicable to our intelligent email goals. Each method uses the notion of a distribution over the classifiers, either producing a distribution for the margin or combining classifiers sensitive to feature distributions. In the large data case, we first showed the performance of CW learning on a large amount of data, which is important for scaling an email prediction system to billions of emails (or more.) We then demonstrated that similar performance could be achieved by parallelizing training across processors, then combining the resulting trained classifiers using confidence sensitive combinations. For active learning, we showed improvements in the standard margin based instance selection methods by normalizing the margins based on their distributions. These results reduce user labeling cost when systems interact with users to obtain a minimum amount of training data for a supervised learning task. Finally, we considered several multi-domain learning settings using CW classifiers and a combination method, a setting inspired by our work on reply prediction. An email system designed for multiple users must pool data to learn across users but still capture user specific behaviors. Our approach creates a better classifier for a new target domain than selecting a random source classifier a priori, reduces learning error on multiple domains compared to baseline approaches, can handle many disparate domains by using many shared classifiers, and scales to a very large number of domains with a small performance reduction.

Chapter 9

Related Work

9.1 Email Overload

Email usage patterns have been widely studied in the human-computer interaction community. Work on email overload can be divided into email organization, email activity management and email triage, which motivate the applications considered in this thesis. For a detailed survey of these efforts see the special issue of the Journal of Human-Computer Interaction on Revisiting and Reinventing Email [250].

9.1.1 Email Organization

Email is a valuable source of information. Emails contain project information, contact information, announcements, procedures and a host of other information that may be valuable at a later time. Many users carefully sort this mail into folders or apply labels or tags to messages so they can find them in the future. User studies have found that a significant amount of time is spent organizing email into folders [113,251]. Some users meticulously file email, creating a wide array of carefully managed folders. Others ignore filing and keep large mailboxes, relying on search to find needed email. A study a decade after that of Whittaker and Sidner [251] revisited some of their observations and found that the challenges remained. While inbox sizes remained the same, the number of folders, daily incoming messages and total email had increased significantly [99]. More recently, a large scale study of a corporate user population found folder usage wide-spread, some with an intricate foldering system [234]. These users also tagged email in a (BlueMail) prototype client to further organize email. Many users found tagging to be complementary to foldering and noted that tagging helped as a search tool. Additionally, several studies found mail organizational behaviors followed the user's job type. Whittaker and Sidner found differences in foldering behaviors between managers and assistants. Danis *et al.* [68] compared the organizational style of managers to non-managers, finding that managers create more categories and a more differentiated category structure than non-managers. Additionally, managers create relationship-oriented categories more often.

Users who spend considerable time organizing email could be assisted by automated systems. Segal and Kephart [214] introduced MailCat (later SwiftFile), an automated email foldering system. MailCat created profiles for each user defined folder based on messages in the folder and used these to offer suggested folders for incoming messages. Users were presented with quick access buttons at the top of each message to move the email into the recommended folder. They later extended this work to the incremental (online) learning setting [213]. While Segal and Kephart developed their tool as a user aid, others have taken email foldering as a document classification task. Bekkerman *et al.* [10] produced a comparative survey of these efforts.

9.1.2 Email Activity Management

Collaborative and email based task management has become the reality in many work places. Several research efforts have studied how people manage tasks in this environment and developed interfaces for this setting. Initial work studied how users manage tasks in their existing applications. Ducheneaut and Bellotti [90] described email as a habitat in which work is received and delegated. Users co-opted email as a personal information management (PIM) tool, following a common tendency to embed PIM directly into familiar workspaces. Their initial work studied consequences of this behavior and suggested new designs to support users. Further work showed how users managed activities as objects using email applications [91]. Similarly, Gwizdka and colleagues studied email habits of users and observed the tendency to manage activities with the email client instead of moving them to other software [112, 113].

These studies led to the development of new software for email activity management. Bellotti and colleagues [12, 13] developed Taskmaster, which casted email as a task management tool and task-centric resources were embedded directly into the email client. A user study showed the advantages for email overload by making email a natural place to manage activities. In further work, they conducted a user study that revealed the inherit structures users employ to measure task progress and to prioritize activities [11]. While Taskmaster casted the email client as a task management tool, their later work embedded the most useful of these tools into a production email client. TV-ACTA (TaskVista and activity-centered task assistant) supported structured activities created within Microsoft Outlook, providing templates based on a user's commonly conducted tasks [15, 16]. Users found this integration useful as it structured task management efforts within the email client. Other efforts yielded similar task management systems, such as TaskView [111]. TaskView allowed users to quickly find information relating to task dates, time and task overviews. This task perspective complemented the inbox, which allowed fast access based on subject lines and senders.

One of the challenges of task management in email is the collaborative nature of tasks. Bellotti *et al.* [14] found that users were overloaded when tasks involved multiple participants since task update became interleaved in the mailbox. A task's state was only identifiable by compiling a series of messages spread throughout the mailbox. Similarly, Muller and Gruen [173] studied the way in which emails and inboxes were shared in a collaborative fashion between users. Along these lines, some task management systems have assumed an entirely collaborative orientation. Researchers at IBM have developed a collaborative activity management system. Their ActivityExplorer [174] enabled people to collaborate around shared artifacts to create "lightweight, informally structured, opportunistic activities featuring heterogeneous threads of shared objects with dynamic membership." These shared activities housed all the necessary information for activity management, although they were as yet unconnected with email. Shared activities are based on the notion of "object-centric sharing," where users share lightweight objects that aggregate and organize different types of shared artifacts, including the ability to impose structure and hierarchy on the activity objects [106]. A later version of this system, called Activities, included templates that captured ad hoc business processes for later reuse [171]. Other collaborative activity management systems include the Unified Activity Manager, also from IBM [53, 172].

Email Activity Management has also been the focus of some machine learning research. The TaskTracer system infers from a user's actions his or her current activity and provides help in managing that activity [78, 216]. Both the CALO and RADAR projects discussed below contain elements of task management and integrations with email. Kushmerick and colleagues have developed several systems towards supporting the types of interfaces described above, of which this thesis is a part. Kushmerick and Lau [138] examined email activities that represented structured business processes such as purchasing books online, and reverse-engineered the structure of processes by learning them from representative email transactions. In contrast, our work in chapter 5 is focused on less-structured activities that are not necessarily generated by formal business processes. Khoussainov and Kushmerick [130] inferred relationships among messages in less-structured activities using a one time batch processing of activities. A summary of these efforts can be found in Kushmerick *et al.* [139].

Others have used machine learning to discover activities in a user's mailbox. Huang *et al.* [121] apply clustering algorithms to discover groups of emails that could correspond to activities. Mitchell *et al.* [169] extends this effort to also generate a description of the discovered task. Dredze and Wallach [87] use a generative model – inspired by some of our work on email activity management – to cluster emails according to activity [246] (Chapter 5). Another approach is to use latent concept models to infer topics in a mailbox

[232]. Cselle *et al.* [61] track topics as they evolve in email to support an intelligent interface. Minkov *et al.* [167] propose several activity-centric search scenarios for email, including finding all emails that correspond to a given activity reference.

Finally, in addition to our machine learning work to support the interfaces described above, some learning systems have been developed as part of IBM Research's collaborative task management system. Li *et al.* [148] used support vector machines to learn a user specific ranking of activities in a collaborative system. Additionally, Shen *et al.* [215] built a recommendation system to find and suggest relevant resources for activities created in the collaborative system.

9.1.3 Email Triage

The volume of email has prompted users to engage in email triage, where users must quickly decide which messages to read and how to act. The constant arrival of new emails interrupts users and harms productivity. Venolia *et al.* [243] listed a variety of factors that users suggested made an email important, such as whether it was from a supervisor, a reply from a previous email, from co-workers and if it was addressed directly to the user. We use some of these factors as features for our reply prediction system. Dabbish *et al.* [65] studied how users make these decisions, finding that factors such as social information predicted a user's action on a message. Neustaedter *et al.* [180] studied triage patterns among users. Users choose either a single pass or a multi-pass strategy. In a single pass strategy, each email is examined once in the inbox. This approach favors simplicity over the ability to quickly find important emails, which is especially problematic when there are many emails and insufficient time to read all of them. The multi-pass strategy was effective in finding important emails quickly, although the strategy was overall more time consuming since it involves multiple passes over the inbox. Tyler and Tang [242] found that users were careful in how and when they replied to emails. The time to respond conveys a "responsiveness image" that depends on the social relationship with the sender.

Several systems emerged from these studies to support inbox organization and triage.

Venolia *et al.* [243] proposed several ways to reduce email distractions, giving users better tools to understand and quickly process incoming email. They also favored a thread based view of the inbox, grouping together messages in the same conversation. IBM Research's ReMail system [129,201] used ThreadArcs [128], which visually showed an email's place in the thread's progression. GrandCentral [244] explored both a linear chronological view and a tree view of email threads. Their visualization simultaneously presented sequence and reply relationships in an email thread. These ideas have been popularized by production email systems, such as Gmail's conversation view.

Building on the observations about the social importance of emails, several systems have extended the traditional sorting by sender feature. Bälter and Sidner's Bifrost [6] clusters emails into groups based on the sender, including "VIP" (pre-specified list), sent to the user directly and sent to mailing lists. ReMail also included an indicator to show if a message was sent from a known contact. Lockerd and Selker's [152] DriftCatcher displays social meta-data about emails. They use social network analysis to identify the relationship strength between senders and recipients and the average time to respond to an email. They also use an SVM to classify emails by interaction process analysis, which labels emails with eight types of interactions, such as "informing," "inquiring," "planning" and "keepInTouch." Neustaedter and colleagues built on their study of email triage to develop Snarf, a tool that displays a social overview of an inbox to support triage decisions [181]. Snarf organizes correspondent into groups (family, manager, close friends, discussion lists) and shows the number of emails currently in the inbox from each group.

Finally, there has been some work on prioritizing email messages automatically. The Priorities system developed by Horvitz *et al.* [120] uses machine learning to rank emails by importance. Their features include social meta-data about emails and previous user behaviors.

9.2 Natural Language Processing Applications

Computational processing of language has been investigated in many domains, such as news and biomedical text and speech and many popular tasks have applied to the email domain. Additionally, new tasks have been inspired by properties of email. In this section we survey some popular NLP applications in email: email understanding, document classification and email summarization.

9.2.1 Email Understanding

Several studies have looked at modeling the intent and meaning of email messages. The most popular example is the classification of emails into Email Speech Acts. This taxonomy is based on Speech Act Theory [3, 211] and characteristics of email. Speech acts include *request for information* and *propose a meeting*. Carvalho and colleagues [35, 50] use a variety of techniques to classify email into speech acts, including collective classification techniques based on multiple emails in a conversation. Speech acts in email have been used in other applications, such as Khoussainov and Kushmerick [130] who define an iterative algorithm whereby speech acts improve task identification, and the identified tasks are then used to improve speech acts. Leusky [145] defined a similar speech acts classification problem to identify a user's role based on speech act patterns.

There are other classification systems based on email intentions. Lockerd and Selker's [152] DriftCatcher, described above, classifies emails according to eight type of interactions. Goldstein and Sabin [204] learned speech acts and genre classification of email. Lampert and colleagues have studied the nature of requests and commitments in email. They defined a Verbal Response Modes (VRM) taxonomy of speech acts, which categorizes emails along two dimensions: literal meaning and pragmatic meaning [141]. They presented a careful definition of requests and commitments in email to guide annotations and measure the confidence of classifications [142]. They built an automated system to classify workplace emails as requests for actions or commitments to act [143]. Other work by Carvalho has focused on modeling the intent of the email sender, such as determining appropriate recipients for an email [36] and alerting the user when an email is sent to an inappropriate recipient (leak detection) [34].

Another perspective has been to modify email to convey additional meta-data about the semantic contents of the message. The Semantic Email project at the University of Washington defines several email patterns that can be annotated with semantic information, allowing for richer interfaces for these processes [163, 164]. Examples include scheduling meetings, RSVPing for events and first come first served giveaways.

9.2.2 Document Classification

Document classification, a common task in natural language processing, has many applications in email. The most well known example of document classification is that of spam filtering, for which naive Bayes models are the most popular [205]. However, there are many other classification settings, many of which are unique to email.

One popular application has been automated foldering, the task of automatically predicting the appropriate folder for a given email message. One of the first such applications of this learning task to a real email client was by Segal and Kephart [214]. Crawford *et al.* [60] showed predicted labels to users in the inbox. Many others have evaluated different learning algorithms on email foldering, such as rule induction and k-nearest neighbors [187], RIPPER (a rule induction classifier) [49], Naive-Bayes [191, 199], SVMs [132], pattern detection from graphs [1] and other models [98]. Bekkerman *et al.* [10] conducted a comparison of numerous techniques on the Enron corpus.

Another document classification task is call center classification, whereby incoming emails to support centers are automatically routed to necessary specialists or replied to automatically [137]. Nenkova and Bagga [178] first identify emails to a call center that seek a response, then filter these by stage of the conversation. Busemann *et al.* [28] used features from shallow text processing, such as chunking, to classify call center emails.

Brutlag and Meek [25] identify some challenges of document classification in the email domain. In particular, they recognize the need to adapt to changing behavior, handle heterogeneity of classes and speed/memory requirements needed for a desktop system.

9.2.3 Email Summarization

Automatic document summarization is an important application in natural language processing. As email volume has increased, so has the desire to apply summarization techniques to the email domain. However, email poses significant new challenges for summarization systems, such as threads instead of single documents, multiple authors, informal writing styles and different types of emails (announcements, discussions, question-answer exchanges, etc.)

There are also different types of summaries that can be generated for email: informative and indicative. An informative summary replaces the contents of the message and summarizes the important points, allowing users to quickly read a long email thread or recall previous threads in preparation for meetings. In contrast, an indicative summary for a document indicates the contents without providing the information presented in the document. Other types of summaries include task oriented summaries, where the summary is a list of tasks mentioned in the email [52]. In this section, we focus on summaries of the first two types.

Informative Summaries

Sentence extraction is a popular approach to informative summarization. There have been several applications of sentence extraction summarization techniques to creating informative summaries for email threads. Lam *et al.* [140] built an early system for evaluating the effectiveness of various summarization techniques. Fasil is a full summarization system used to generate summaries for mediums with limited interaction capabilities, such as automated phone systems [67]. However, the two dominant approaches to email summarization have relied on detecting question-answer pairs and using clue words.

The work of McKeown and colleagues use both standard summarization techniques and question-answer extraction to summarize email threads. They assume that the first email in the thread introduces the topic of discussion and that subsequent emails address this topic without changing subjects. Based on this structure, they use centroids in vector space to identify key sentences in the initial email and related sentences in followup messages [248]. Additional features relevant to this task are explored by Rambow *et al.* [195]. Sentences from each email are then tiled together to form a summary.

Shrestha and McKeown [220] extend this approach to identifying and extracting question-answer pairs directly, a goal similar to that of Murakoshi *et al.* [175]. Machine learning classification is used to identify which sentences in the original email are questions and which sentences in subsequent emails answer the questions. These extracted question-answer pairs and the extracted sentences from discussions are interleaved to form a single summary of the email thread [165].

Carenini *et al.* offer a second approach to email thread summarization based on conversational elements in email [30]. They observe that followup emails will quote previous messages when responding, often layering the response with the original quotation. From this structure they construct a quotation graph linking fragments of emails with other messages where they are quoted. The summary contains the most cohesive fragments, selected by word repetition in fragments. Followup work compared this approach favorably to graph methods based on semantic similarity [31].

Indicative Summaries

The purpose of an indicative summary is to give the reader an idea about what is discussed in a document without replacing the need to read the document. It should aid the user in deciding whether or not the document is relevant to the current task. Nenkova and Bagga [179] identify sentences in the first email of the thread that contain the main issue by finding the shortest sentence in the message that has the largest overlap of nouns, verbs, adjectives or adverbs with the message's subject. Muresan *et al.* [176] extract high quality noun phrases to capture the gist of an email. A third approach is that of Newman [182,183] for understanding the contents of an email collection. Newman clusters groups of topically related messages and then extracts a summary common to the whole group.

9.3 Email Analysis

The shift of businesses to electronic communications has created a large repository of documents in the form of email archives. Many applications require understanding these archives, such as corporate reviews and legal discovery. After the collapse of Enron in 2001, federal investigators obtained hundreds of thousands of emails from the company to study by hand. Since then, new tools have emerged from the fields of natural language processing, machine learning and social network analysis to analyze email archives. These tools can learn important relationships in the data, create organizational structures, disambiguate named references and find suspicious or interesting patterns of communication. For an overview of these tools see section 3.5.1.

9.4 Smart Desktops

Much of the work in artificial intelligence in email in recent years has grown out of the Personalized Agent that Learns (PAL) project [1]. The goal of the PAL project is the development of cognitive assistants that improve the way computers support human information management. This project includes two large efforts: CALO and RADAR.

CALO (Cognitive Assistant that Learns and Organizes)[2] has developed the IRIS client, which handles a wide variety of rich applications supported by intelligent assistants [47]. Work on CALO has lead to hundreds of published papers in machine learning, natural language processing, systems and human-computer interaction. CALO includes systems that learn calendar preferences [105], extract information about user activities [169] and facilitate collaborative planning and interactions between users [8]. CALO also has extensive

[1] `http://www.darpa.mil/IPTO/programs/pal/pal.asp`

[2] `http://caloproject.sri.com/`

task learning capabilities based on the programming by example paradigm. One such tool is LAPDOG, which acquires fully or partially automated procedures by observing a user's interaction with PIM applications [104]. These procedures can span multiple applications and includes tasks like setting up a meeting or tracking a requisition.

The RADAR project[3] has focused on supporting users engaged in managing tasks like organizing a conference [101]. RADAR learns in the wild, observing how a user manages a task and then learns to support these behaviors. For example, a user scheduling a conference may get numerous emails for a number of tasks, like scheduling rooms, ordering food and arranging a conference program. RADAR can learn to identify each one of these tasks and automate them, for example, finding a free room of the right size when a room request email arrives. RADAR also manages the long term goals of the project, scheduling important tasks first and making sure tasks are completed before deadlines. Studies have demonstrated improved user efficiency when using RADAR as compared to acting alone or even with a human assistant [224].

A final example of a smart desktop system is TaskTracer, a learning system integrated with the user's desktop environment that infers the current task of the user [78,230]. TaskTracer observes user actions and learns how to support a user conducting an activity. TaskTracer integrates with the user's file system, document processing application and email client. Once TaskTracer learns user activities it can support user actions by suggesting relevant webpages [144] or appropriate folders for finding and saving documents [7].

[3]`http://www.radar.cs.cmu.edu/`

Chapter 10

Conclusion

This thesis has made two major contributions:

1. **Intelligent Email:** We introduced the concept of intelligent email as supporting smarter user actions through user oriented learning techniques for user supporting interfaces. Most work using machine learning in email focuses on email analysis and standard learning problems. Alternatively, some build intelligent systems to automate user actions. Instead, intelligent email learns information to support user email management and enhance user interfaces. In building intelligent email systems, we use techniques such as shared representations with user specific feature extractions and unsupervised learning, to learn systems specific to each user. We have demonstrated this approach with several novel applications: attachment prediction, reply prediction, user-specific representations for learning, email activity classification and email facet ranking. Additionally, a user survey of our facet ranking system indicated that users benefited from using an intelligent email system. The main contributions of this work have been to the field of Intelligent User Interfaces.

2. **Confidence-Weighted Learning:** We have developed a new online learning algorithm for linear classifiers that incorporates confidence in parameters into the learning process. Confidence-Weighted learning is sensitive to feature distributions in

data, forcing stronger updates for features that have been observed less often. To achieve this effect, we replace the linear classifier with a Gaussian distribution over linear classifiers. As the algorithm encounters more examples, the distribution gradually becomes peaked, representing an increasing confidence in the hypothesis. We have also demonstrated that parameter confidence is useful for other learning settings, especially some that are motivated by intelligent email. We have demonstrated improvements in learning using many examples, active learning, and multi-domain learning. The main contributions of this work have been to the fields of machine learning and natural language processing.

Towards the goal of intelligent email, we have developed and evaluated several learning applications.

Chapter 2 A common problem in email today is the forgotten attachment, where users send emails and forget to attach a necessary file. We introduced an attachment prediction system that uses supervised classification to predict missing attachments based on email contents and relationship features. Since we first published this work, attachment detectors have been built for several mail clients, including Microsoft Outlook,[1] Apple Mail, [2] Mozilla Thunderbird [3] and Google's Gmail. [4] All of these systems are rule based, looking for the word "attach" or "attachment." Our work shows that learning systems are much more effective than rule based approaches for this task.

Chapter 3 When a user quickly reads his or her inbox, the lack of information makes informed decisions difficult. We developed the reply predictor, an intelligent system that automatically labels messages that need a reply. We presented a new user interface that marks messages that need a reply and allows the user to manage reply commitments. Our

[1]http://mark.bird.googlepages.com/

[2]http://www.hawkwings.net/2006/04/06/plugin-to-scan-for-missing-attachments/

[3]https://addons.mozilla.org/en-US/thunderbird/addon/2281

[4]http://gmailblog.blogspot.com/2008/09/new-in-labs-handy-intern-tweaks.html

main contribution in this work is the application of reply prediction and the cross-user learning setting for email. We address the latter through a shared feature space extracted in a user specific way. We show that principles from social network and email analysis can be used as deictic features to capture user behavior and social relationships. These features enable cross-user learning.

Chapter 4 Many email learning tasks rely on the content of messages. We evaluated several content representations based on latent concept models such as latent semantic analysis and latent Dirichlet allocation. These dimensionality reduction techniques allow user specific representations by relating words in the email to user specific concepts. Our main contribution is content summarization with keywords based on these models. We show how these models relate emails to larger concepts in the mailbox and evaluate these representations for two learning tasks. Furthermore, our results are promising for email summarization, which may benefit from better representations of message content.

Chapter 5 The HCI community has developed several email activity management systems to facilitate better email management. These systems often require grouping messages into activities. We presented a system for automatically classifying emails into activities in an online setting. Our system creates representations of activities in terms of content, social and threading information. This work was the first to support an intelligent email activity assistant by interacting with the user to create activity groups as new messages arrive. More recent work on email activity management in the AI community has used this approach as a model, seeking ways to support these types of interactions or cover other aspects of activity management.

Chapter 6 Faceted browsing and search has become a popular tool in many domains, but email generates too many facets to display to the user. We introduce the task of facet ranking, which uses machine learning to select the most useful facets based on the current collection of messages. A ranker can select only the most useful facets to display to the user. This work is one of the first systems to explore how facets can help email search and triage. Additionally, there have been few user studies of intelligent email systems [5], and our user survey is one of the first to explore how email users can interact with an intelligent interface. Furthermore, while other work has used machine learning to create facets or facet hierarchies [66, 136, 227], to our knowledge, our work is the first to use machine learning to generate faceted browsing interfaces.

In addition to our work in intelligent email, we have also developed new learning algorithms based on our experiences in applying learning to email problems. We have applied the resulting algorithm to learning settings relevant to email applications.

Chapter 7 We introduced Confidence-Weighted learning, a new learning algorithm sensitive to feature distributions common to NLP applications. Our new algorithm performs better than many common learning algorithms on a range of NLP tasks. We plan to extend these algorithms to new settings, such as multi-class and structured prediction.

Chapter 8 While parameter confidence improves learning, it is also useful for other learning tasks. We have demonstrated how confidence can be used to improve important learning settings: active learning, large scale learning and parallel training. We have also introduced a new learning setting: multi-domain learning. We show how parameter confidence can be used to generalize common features across domains and learn domain specific feature behaviors.

Appendix A

Machine Learning Methods

The applications of machine learning in this thesis are of two types: supervised and unsupervised learning. While all machine learning uses examples, supervised learning relies on examples labeled for a specific task and unsupervised learning discovers latent patterns in the data. In this section, we will explore both types of learning as they are applied in this thesis. Our explanations will have an NLP focus to follow the applications in this thesis.

A.1 Supervised Learning

In supervised learning, the algorithm is provided a set of examples labeled according to a guideline for a task of interest. For example, for binary classification each document may be an email message labeled as either *needs reply* or *does not need reply*. In this case, the goal of a supervised learning algorithm is to train a classifier to distinguish between instances of each label type so as to label new unseen instances.

Each document, or email, is called an instance, or example, of the learning task. The example show how the algorithm should learn to label a particular document. The instance is represented by a feature vector $\boldsymbol{x} \in \mathbb{R}^d$, where each of the d dimensions in $\boldsymbol{x}$ is a real valued number for a feature in the instance. For instance, each word in a corpus may be a feature, so d are the number of words in the vocabulary and the dth position of $\boldsymbol{x}$ is the number of times the word occurs in the document. For batch algorithms, we assume these instances are drawn from a distribution $\boldsymbol{x} \sim \mathcal{D}^d$. Each document is labeled by some unknown function and we are provided the label y_i from this function for each document $\boldsymbol{x}_i$. For binary classification, the label will be given as $y_i \in \{0, 1\}$ and in multiclass learning the label will come from a larger set of labels $y_i \in \mathcal{Y}$.

Sometimes we express features of a document explicitly in our notation and treat $\boldsymbol{x}$ as an indicator of the document. In this case, $\Phi_f(\boldsymbol{x}, y)$ is a feature function that examines $\boldsymbol{x}$ and y to produce a value for the fth feature in document $\boldsymbol{x}$ when considering label y. For example, the fth feature may correspond to the presence of the word "dog" in the document, so $\Phi_f(\boldsymbol{x}_i, y_i)$ returns the number of times "dog" appears in the ith document. In many NLP applications, features are assumed to be binary (i.e. $\Phi_f(\boldsymbol{x}, y) \in \{0, 1\}$.

In supervised learning, we are provided a collection of N examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. The goal of a supervised learning algorithm is to learn a function from the observed examples to accurately label new examples drawn from $\mathcal{D}^d$.

In this thesis, we apply two machine learning methods for learning functions from labeled data. Each one corresponds to two popular learning principles: maximum entropy and maximum margin. We also consider two learning settings: batch and online learning. We first explain these two learning setting and then give an example of each below.

A.1.1 Batch and Online Learning

In batch learning, an algorithm is given the N training examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ from which it learns a function. Since the algorithm has access to all examples, it can define a global objective over all of the training data. In contrast, online algorithms operate in rounds. On round i the algorithm receives an instance $\boldsymbol{x}_i$ to which it applies its current prediction rule to produce a prediction $\hat{y}_i$. It then receives the true label y_i and suffers a loss $\ell(y_i, \hat{y}_i)$, such as the zero-one loss ($\ell(y_i, \hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$ otherwise $\ell(y_i, \hat{y}_i) = 0$.) The algorithm then updates its prediction rule and proceeds to the next round. This models an interactive learning environment, where outputs from the system are corrected by an oracle after each round.

There are advantages and disadvantages to each learning setting. In general, batch algorithms outperform online algorithms since they can examine all available training data at once. In contrast, online algorithm make local decisions based on single instances. However, online algorithms are much faster and easier to implement. Additionally, online algorithms can more easily scale to larger datasets since they need only process a single instance at a time. In this thesis, we typically use batch methods. However, in a setting meant to simulate a real interactive learning environment, such as automatic foldering, we consider online algorithms. A detailed treatment of online learning can be found in Cesa-Bianchi and Lugosi [41].

A.1.2 Maximum Entropy Learning

Maximum Entropy learning is a probabilistic learning method that uses empirical distributions over training data to learn a log-linear classifier. We begin by assuming that the labeled examples are evidence of some optimal labeling function. Using these example, we can measure the expectations of each feature with respect to each label. For example, if we observe a feature frequently occurring with a certain label, we expect the optimal model to indicate that the feature is predictive of the label. We term the empirical distribution of the data as $\hat{p}$.

Using the computed expectations for each feature and label, we seek a probabilistic model that matches these expectations. Defining each expectation as a constraint, we can then find a probabilistic model of the data that respects all of the constraints. However, even with numerous model constraints there are still many models that produce the desired feature expectations. How are we to choose a single model from many possible models?

Intuitively, we seek a model that respects the observed constraints without imposing any additional bias. This desire can be expressed by the selecting the most uniform distribution subject to the constraints defined by the training data. A common measure of uniformity of a conditional distribution $p(y|\boldsymbol{x})$ is the conditional entropy The conditional entropy $H(p)$ of the distribution is given by

$$H(p) = -\sum_{\boldsymbol{x},y} \hat{p}(\boldsymbol{x}) p(y|\boldsymbol{x}) \log p(y|\boldsymbol{x}) \; , \tag{A.1}$$

where $\hat{p}$ is the empirical distribution of the data and the argument p indicates the observed variables $\boldsymbol{x}$ and y. Using this notion of entropy, we define the maximum entropy learning objective as:

- **Learning Objective**: Select the distribution p with the maximum entropy subject to the constraints of the training data.

It can be shown that this learning objective has a dual formulation of finding a model p that maximizes the likelihood of the training data [17]. Specifically, we can find a parametric representation of p that is the maximum entropy model by finding the model p that maximizes the log-likelihood $L_{\hat{p}}(p)$ of the empirical distribution:

$$L_{\hat{p}}(p) = \log \prod_{\boldsymbol{x},y} p(y|\boldsymbol{x})^{\hat{p}(\boldsymbol{x},y)} = \sum_{\boldsymbol{x},y} \hat{p}(\boldsymbol{x}, y) \log(p(y|\boldsymbol{x})) \; . \tag{A.2}$$

A model that maximizes the log-likelihood is the maximum entropy model. Therefore, the objective of learning in the maximum entropy setting can be formulated in terms of the dual as:

- **Learning Objective**: Select parameters of the model so as to maximize the likelihood of given training data.

In this formulation, we seek to learn a set of parameters λ_f for each feature function $\Phi_f(\boldsymbol{x}, y)$ in the parametric representation of p. The probability of a label given these parameters is given as

$$p_\lambda(y|\boldsymbol{x}) = \frac{1}{Z_\lambda(\boldsymbol{x})} \exp(\sum_f \lambda_f \Phi_f(\boldsymbol{x}, y)) \; , \tag{A.3}$$

where $Z_\lambda(\boldsymbol{x})$ is the normaization constant to ensure a valid probability distribution, i.e. $\sum_y p_\lambda(y|\boldsymbol{x}) = 1$ for all $\boldsymbol{x}$:

$$Z_\lambda(\boldsymbol{x}) = \sum_y \exp(\sum_f \lambda_f \Phi_f(\boldsymbol{x}, y)) \ . \tag{A.4}$$

For predictions, we need only compute the numerator as $Z_\lambda(\boldsymbol{x})$ does not depend on y.

In practice, most maximum entropy algorithms deal with the likelihood representation. Since the likelihood function is convex in the parameters λ, we can use any convex optimization routine, such as those found in Boyd and Vandenberghe [24]. Additionally, for binary classification, maximum-entropy learning reduces to simple logistic regression.

Early work on training maximum entropy models favored techniques such as iterative scaling. Newer approaches use gradient based methods, such as gradient ascent and L-BFGS [46, 151]. Additionally, we can add other terms to the objective function to balance likelihood maximization with other requirements. For example, the optimum solution to the objective often yields poor generalization performance when tested on held out data. This effect is caused by overfitting, where parameters of the model are tuned to fit every observation of the training data, even when such data contains statistically insignificant artificats. Therefore, we relax the requirement of fitting the training data by augmenting the objective with penalty terms: regularizers.

While there has been significant work on regularization techniques, we use a simple Guassian prior over the model parameters [46]. In Guassian prior smoothing, a Gaussian prior is placed over the model parameters with mean 0 and diagonal covariance. The maximum-likelihood objective is augmented with this regularizer to yield:

$$L_{\hat{p}}(p) + \sum_i \log \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{\lambda_i^2}{2\sigma_i^2}\right) \ . \tag{A.5}$$

Since the prior is convex, finding the optimal solution to the objective is still a convex optimization problem. For a detailed derivation and explanation of the maximum entropy framework, see Berger *et al.* [17], Ratnaparkhi [197] and Ratnaparkhi [198].

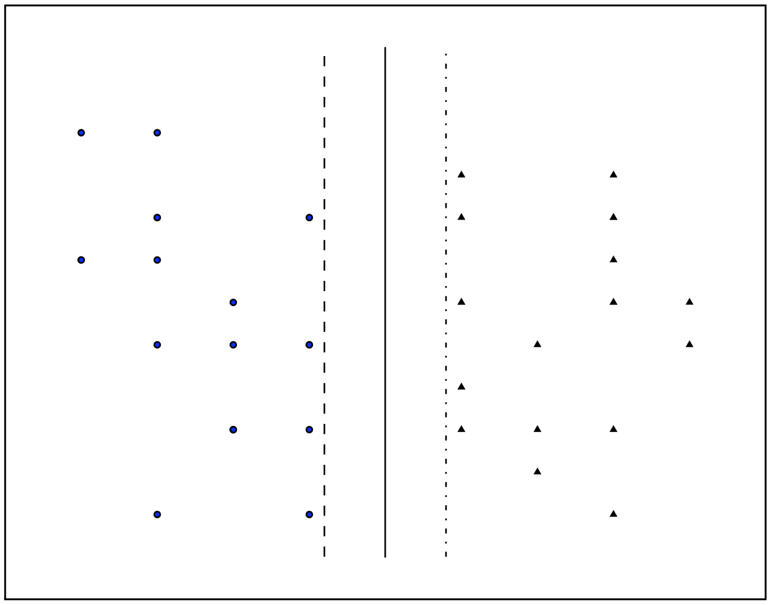


Figure A.1: A two dimensional learning example where points are classified as either blue circles (left) or black triangles (right). The three lines are all possible separators for the points. The center solid line is the max-margin separator.

A.1.3 Max-Margin Learning

Another approach to supervised learning takes a geometric view of the problem. Consider each input example $\boldsymbol{x}$ as representing a point in a d dimensional space, where each point has label y. For simplicity, we consider binary classification, i.e. $y \in \{0, 1\}$. We assume that the given points are linearly separable: there is a separating hyperplane (linear) $\boldsymbol{w}^*$ that accurately divides all points in the space according to their binary label. The goal of a learning algorithm will be to find a linear separator $\boldsymbol{w} \in \mathbb{R}^d$ based on the training data to label new examples.

By looking at all of the training examples, it is straightforward to find a separator that divides the space correctly. However, like in the probabilistic setting, there are many possible hyperplanes that separate the data. As before, we seek a learning objective so that we can choose a single separator.

Consider the two dimensional example shown in figure A.1. In this example, the points are classified as either blue circles (left) or black triangles (right). There are clearly many lines that separate these points accurately; three are shown in the figure. The distance between an example (point) and the separator (line) is called the margin, and the signed margin indicates accuracy as well as distance (positive margins are correct classifications.)

While all three lines are acceptable, the dashed ones seem undesirable since they are close to the points (small margins.) It is likely that a new point may cross one of these lines. In contrast, the solid black line in the middle has the highest margin, i.e. it is the furthest from any point. Given the observed examples, this seems like the best choice for a separator and it is called the max-margin separator since it has the maximal margin from all known points. We can now state a max-margin learning objective:

- **Learning Objective**: Select the max-margin separator such that all training examples are classified correctly with the greatest margin.

Using this learning objective, we can define a quadratic programming optimization problem to find the max-margin separator:

$$\min_{\boldsymbol{w}} \tfrac{1}{2}\|\boldsymbol{w}\|^2 \tag{A.6}$$

$$\text{such that} \quad \forall \boldsymbol{x}_i, y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) \geq \gamma \; . \tag{A.7}$$

A prediction using the model $\boldsymbol{w}$ is defined as the sign of the inner product with the example i.e. $\text{sign}(\boldsymbol{w} \cdot \boldsymbol{x})$. b determines the offset of the hyperplane from the origin and γ is the margin (usually $\gamma = 1$.) The objective ensures that each point is classified correctly with a margin of γ and that the norm of the weight vector $\boldsymbol{w}$ remains small. Algorithms with this objective are called Support Vector Machines [27].

One of the assumptions we made was that the training data was linearly separable. In many domains, this is not the case and there is no separator that accurately divides the data. There are two common approaches to this problem. First, we can relax our condition to allow some examples to violate the constraint. Slack variables are typically used to determine how example are permitted to violate their constraint. A second approach is to use kernel learning, whereby the examples are projected into a higher dimensional space in which there exists a linear separator [209].

SVM classifiers can also be extended to multiclass and structured problems. For more information on such extensions see Crammer and Singer [56] and Tsochantaridis *et al.* [240].

A.1.4 Passive-Aggressive Learning

We now introduce an online algorithm for max-margin learning. We begin by considering the popular Perceptron algorithm [202].

The Perceptron algorithm maintains a weight vector $\boldsymbol{w} \in \mathbb{R}^d$ and labels an instance $\boldsymbol{x}$ by taking the sign of the product of these vectors: $\hat{y} = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x})$. The algorithm is mistake driven (conservative or passive,) meaning that the weight vector is only updated when an example is classified incorrectly. After an incorrect classification, the algorithm moves the hyperplane closer to the instance in accordance with the magnitude of the example:

$$\begin{array}{lll} \text{if} & \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}) \neq y_i & \boldsymbol{w}_{i+1} = \boldsymbol{w}_i + y_i \boldsymbol{x}_i \\ \text{else} & & \boldsymbol{w}_{i+1} = \boldsymbol{w}_i \end{array}$$

where $\boldsymbol{w}_{i+1}$ is the weight vector after training on the ith example.

In the limit, the Perceptron algorithm is guaranteed to find a linear separator for linearly separable data [29, 40]. However, there is no guarantee on the performance of the algorithm after an update. If an example is incorrectly classified, the new weight vector may still label it incorrectly.

Another approach builds on the max-margin framework to introduce an aggressive online algorithm, where the weight vector resulting from an update will correctly classify and example with some margin. One such family of algorithms are Passive-Aggressive (PA) algorithms, which follow the form of the Perceptron algorithm but with aggressive learning conditions. One such PA algorithm, MIRA, updates the weight vector to guarantee correct classification with a given margin on the new example. Specifically, the new weight vector $\boldsymbol{w}_{i+1}$ is given by the solution to the following objective:

$$\begin{aligned} \boldsymbol{w}_{i+1} = & \quad \arg\min_{\boldsymbol{w}} \tfrac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}_i\|^2 \\ \text{such that} & \qquad y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i) \geq \gamma \ . \end{aligned}$$

Solving this optimization problem yields the following update rule:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_i y_i \boldsymbol{x}_i \ , \tag{A.8}$$

where

$$\alpha_i = \frac{\ell_\gamma(\boldsymbol{w}_i; (\boldsymbol{x}_i, y_i))}{\|\boldsymbol{x}_i\|^2} \ . \tag{A.9}$$

$\ell_\gamma(\boldsymbol{w}; (\boldsymbol{x}, y))$ is the hinge loss function with margin γ. This function returns 0 when the signed margin of $\boldsymbol{w}$ on example $\boldsymbol{x}$ is at least γ.

As with SVMs, MIRA can be augmented with slack variables for non-separable data. For a detailed development and analysis of Passive-Aggressive learning, see Crammer [54] and Crammer *et al.* [55].

Multiclass Extensions

Some of the applications in this thesis require multiclass methods. We now extend the PA framework to multiclass problems. Instead of a binary label, we now have a set of many possible labels $\mathcal{Y}$. For each instance, we can predict one of these labels with $\boldsymbol{w}$ by selecting the label that maximizes the score of the model:

$$h(\boldsymbol{x}) = \arg\max_{z \in \mathcal{Y}} \boldsymbol{w} \cdot \Psi(\boldsymbol{x}, z) \ . \tag{A.10}$$

We call the inner product $\boldsymbol{w} \cdot \Psi(\boldsymbol{x}, z)$ the score of label z. Following the PA model, we seek an update that ensures that the correct label is favored over other possible labels with a given margin γ. Specifically, we would like to add constraints of the form $\boldsymbol{w} \cdot \Psi(\boldsymbol{x}, y) - \boldsymbol{w} \cdot \Psi(\boldsymbol{x}, z) \geq \gamma$ for all labels $z \in \mathcal{Y}$ where $z \neq y$. In practice, it may not always be possible to add constraints for every possible label z, especially when z is a structured label, which can lead to exponentially many labels. Instead, we can enforce this condition for the top k highest scoring labels for an example. We define the set $best_k(\boldsymbol{x})$ to be the k highest scoring labels according to $h(\boldsymbol{x})$. Adding these constraints yields a multiclass algorithm called k-best MIRA:

$$\begin{aligned} \boldsymbol{w}_{i+1} = & \quad \arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}_i\|^2 \\ \text{such that} \quad \forall z \in best_k(\boldsymbol{x}_i), & \quad \boldsymbol{w} \cdot \Psi(\boldsymbol{x}_i, y_i) - \boldsymbol{w} \cdot \Psi(\boldsymbol{x}_i, z) \geq \gamma \ . \end{aligned}$$

This same extension can be applied to structured prediction problems as well, where the label $\boldsymbol{y}$ is now a complex structure of labels for the input $\boldsymbol{x}$. For a full treatment of multiclass and structured prediction problems, see McDonald *et al.* [160] Crammer *et al.* [55] and Crammer and Singer [58].

A.2 Unsupervised Learning

The goal of unsupervised learning is to uncover useful patterns in the data. The selected model encodes a belief about the data and the algorithm attempts to fit the data to the given model. In this thesis we use latent concept models, which are designed to group words in a collection of topics according to topics or concepts [21, 109, 118, 157, 245]. These models treat documents as having an underlying latent semantic structure, which may be inferred from word-document co-occurrences. The latent structure provides a low-dimensional representation that relates words to concepts and concepts to documents.

There are two popular approaches to designing such models. Latent semantic analysis (LSA) measures word co-occurrences and a singular value decomposition (SVD) to identify how words and documents relate to latent concepts. Latent Dirichlet allocation (LDA) uses Bayesian statistics to infer word to topic relationships from a corpus assumed to be generated by a latent model. In general, LDA has the benefit of having a clear probabilistic interpretation and has been shown to improve over LSA in a wide range of applications [21, 249]. We describe both models in this section.

A.2.1 Latent Semantic Analysis

Latent semantic analysis, introduced by Deerwester *et al.* [71], assumes some underlying latent structure in the pattern of word usage across documents in the corpus. The goal of LSA is to learn this structure through word co-occurrence. The learned structure can then be used in place of the given words and documents to represent the data. For example, words with similar meaning and usage patterns, such as "canine" and "dog," will be strongly associated with the same latent factors, while dissimilar words will not. We can represent both of these terms using their latent concepts.

LSA constructs a word-document co-occurrence matrix X from a text corpus. Each row of X corresponds to a word in the vocabulary and each column corresponds to a document. Assuming D documents and W words in a text corpus, we construct a $W \times D$ matrix X, where the element X_{wd} indicates the number of times word w occurs in document d. While the simplest approach constructs the matrix using raw word counts, other techniques rely on more complex measures, such as TF-IDF scores.

LSA uses singular-value decomposition (SVD) to create a low dimensional representation of the matrix. A SVD of a matrix decomposes it into orthogonal factors, of which we select the top k to approximate the original matrix X. A $W \times D$ matrix X can be decomposed into three matrices as $X = USV^T$, where U is a $W \times R$ matrix, S a square $R \times R$ matrix and V a $D \times R$ matrix. U and V have orthonormal columns and S is diagonal matrix with rank R. The decomposition of X into these three matrices is known as the SVD of X. Instead of maintaining S as a rank R matrix, LSA keeps only the K largest singular values of S along with the corresponding columns in U and T. The result is a new matrix $\hat{X}$ that is the unique matrix of rank K closest to X in the least squares sense. Since we maintain only the first K independent linear components of X, the matrix $\hat{X}$ captures the major associations in the matrix, hopefully ignoring the noise. These associations are often considered as semantic associations, hence the name latent semantic analysis.

In applications of LSA, the reduced matrices $\hat{U}$ (size $W \times K$) and $\hat{V}$ (size $D \times K$) represent relationships between words and documents to the underlying concepts. While the original word-document matrix X contains counts, $\hat{U}$ and $\hat{V}$ are real-valued and indicate the positive or negative association between each word and document and a particular latent factor. For example, if we want to measure the similarity between two words, we can compute the dot product of their entries in the reduced matrix $\hat{U}$. Returning to our example above, we'd expect a high score for the words "canine" and "dog" since they likely share the same underlying concepts. In applications to information retrieval problems, LSA is often called latent semantic indexing since it is an extension of the vector space retrieval model and can be used to index documents in a corpus. The most appropriate number of latent factors K depends on the corpus. In smaller corpora in this thesis, lower values, such as $K = 50$ are appropriate. Larger values, typically between 100 to 300, are appropriate for much larger corpora. For a detailed explanation of LSA, see Dumais [92].

A.2.2 Latent Dirichlet Allocation

Latent Dirichlet allocation is a Bayesian generative model of a corpus and underlying topics. The model assumes that a corpus of documents was generated by a Bayesian generative process and infers the underlying model from the observed documents. LDA is an example of one of many probabilistic topic models. Since most topic models are based on LDA, we present the basic model in this section.

LDA's generative process assumes that each document was generated from a series of latent topics. Each latent topic is defined as a distribution over words in the vocabulary. Highly probable words in a topic represent the semantic concepts in that topic. Each document contains a distribution over latent topics. For each word in the document, a topic is chosen from this distribution and then a word is drawn from the topic according to its distribution over words. This sampling procedure is repeated until all words in the document have been generated. For example, a corpus of newspaper articles might contain latent topics that correspond to concepts such as "politics," "finance," "sports" and "entertainment." Each article has a different distribution over these topics: an article about government spending might give equal probability to the first two topics, while an article about the World Cup might give equal probabilities to the last two. An article about government spending will then contain words from the "politics" and "finance" topic, such as "bill," "budget" and "congress."

Model Definition

Given a corpus containing D documents, the ith document d_i is generated using a distribution θ^d over T topics. The mixture of topics $\boldsymbol{z}_d$ in document d has probability $P(\boldsymbol{z}_d)$. This probability decomposes into the probability of each topic being assigned to each word: the probability of the ith word being assigned topic j is $P(z_{d,i} = j)$. The probability of the ith word in document d given its topic is j is given as $P(w_{d,i}|z_{d,i} = j)$. LDA gives a document's probability as the product of each of the N words marginalized over all topics:

$$P(\boldsymbol{w}_d) = \prod_{i=1}^{N} P(w_{d,i}) = \prod_{i=1}^{N} \sum_{j=1}^{T} P(w_{d,i}|z_{d,i} = j) P(z_{d,i} = j) \ . \qquad \text{(A.11)}$$

The probability of a word being produced by a topic ($P(w|z = j)$) is written as ϕ^j: a multi-nomial distribution over words for topic j. The document specific distribution over topics is written θ^d. An LDA model is defined by T topic specific distributions ϕ^j over words and D document specific distributions over topics θ^d.

LDA uses a Dirichlet prior on θ, which is a conjugate prior for the multinomial. The T dimensional Dirichlet distribution allows the model to generate document specific distributions over topics, subject to given concentration parameters, which define the variance of the multinomial distributions. A T dimensional Dirichlet distribution is given as:

$$\mathrm{Dir}(\alpha_1, \ldots, \alpha_T) = \frac{\Gamma\left(\sum_j \alpha_j\right)}{\prod_j \Gamma\left(\alpha_j\right)} \prod_{j=1}^{T} p_j^{\alpha_j - 1} \ . \qquad \text{(A.12)}$$

The concentration parameters $\alpha_1, \ldots, \alpha_T$ control smoothness of drawn distributions. Most topic models use a symmetric Dirichlet distribution and set all hyperparaters to α. An additional hyperparameter of β is used to control smoothing on the multinomials ϕ.

We can now define the generative process for document d of length n:

1. Choose $\theta^d \sim Dir(\alpha)$.

2. For each of the n words in document d:

   Choose a topic $z_{d,i} \sim$ Multinomial(θ^d).

   Choose a word $w_{d,i}$ from $P(w_{d,i}|z_{d,i}, \phi^z)$, a multinomial distribution conditioned on the topic $z_{d,i}$.

Inference

Learning a topic model means inferring the distributions θ and ϕ from a corpus. Specifically, we seek to compute the posterior distribution over the hidden variables, the topic assignments and document specific topic distributions:

$$P(\theta, \boldsymbol{z}|\boldsymbol{w}, \alpha, \beta) = \frac{P(\theta, \boldsymbol{z}, \boldsymbol{w}|\alpha, \beta)}{P(\boldsymbol{w}|\alpha, \beta)} \ . \tag{A.13}$$

Unfortunately, this distribution is intractable due to a coupling between θ and β when the distribution is marginalized out over hidden variables. Instead, there are two popular ways to estimate these distributions. The first is variational inference, where a set of variational parameters are estimated to find the tightest lower bound on the log likelihood of the distribution [21]. The second approach, which we use in this work, is based on Markov chain Monte Carlo (MCMC). A Gibbs-EM sampling (or alternating conditional sampling) is used to simulate a high-dimensional distribution by sampling lower-dimensional subsets of variables, each conditioned on the value of the other variables. Additionally, our Gibbs-EM algorithm optimizes the Dirichlet parameters α and β. Gibbs-EM alternates between optimizing α and β and sampling a topic assignment for each word in the corpus from the distribution over topics for that word, conditioned on all other variables. For a derivation of this procedure, see Steyvers and Griffiths [225].

Note that LDA makes no assumptions about the order of words in the document, relying on the common bag-of-words representation of documents. Additionally, there is no correlation between topics as any two topics are equally likely to co-occur. Later models address both of these points by using previous words [245] or syntax [110]. Other models are sensitive to topic interactions and measure pair-wise topic correlations [20] or learn a topic hierarchy [19, 149, 166, 237].

We note that while there are more advanced topic models that obviate the need for setting the number of topics T manually, such as the Hierarchical Dirichlet Process [149]. For more information on topic models, see Blei *et al.* [21] Griffiths and Steyvers [109] and Steyvers and Griffiths [225].

Appendix B

User Survey

The following is a copy of the user survey used to collect feedback for the search operator suggest experiment in chapter 6. For results of the survey, see section 6.7 and appendix C.

General

1. About how many emails do you receive a day?

   Text box

2. About what percentage of incoming emails skip your inbox (filter archives it)?

   Text box

3. How many labels do you regularly use?

   Text box

4. How long have you used Gmail?

   Less than 6 months

   1 year

   2 years

   3 years

   4 years or more

5. How many unread messages are in your inbox right now?

   Text box

Concept

6. I like the idea of having search operators suggested.

   Five-point Likert item

7. Filtering the inbox by search operators helps me manage my inbox.

   Five-point Likert item

8. Suggesting operators for search results helps me find what I am searching for.

   Five-point Likert item

9. I would use a search operator suggestion tool (given high quality suggestions and a good UI.)

   Five-point Likert item

Implementation

10. When did you look at the list of operators?

    Text box

11. I typically did the following after selecting an operator:

    Read a message

    Archived

    Forward/Reply

    Labeled messages

12. Highlighting emails that match an operator (mouseover operator) was helpful.

    Five-point Likert item

13. I have learned to use more operators in my Gmail searches.

    Five-point Likert item

14. If you found the tool useful, please describe how you used it.

    Text box

15. Did you find the tool useable? What can we do to improve usability?

    Text box

16. How can we improve Gmail search?

    Text box

17. What features would help you deal with email overload.

    Text box

18. What is your #1 frustration with Gmail?

    Text box

Appendix C

User Survey Results

This appendix contains additional results for the search operator suggest experiment survey in chapter 6. A copy of the survey can be found in appendix B and results are discussed in section 6.7. Figures are histograms showing the number of responses of each type for the indicated question. The survey had 85 respondents, although not every respondent answered every questions. The 5 point Likert agreement items were labeled *agree* (5) and *disagree* (1).

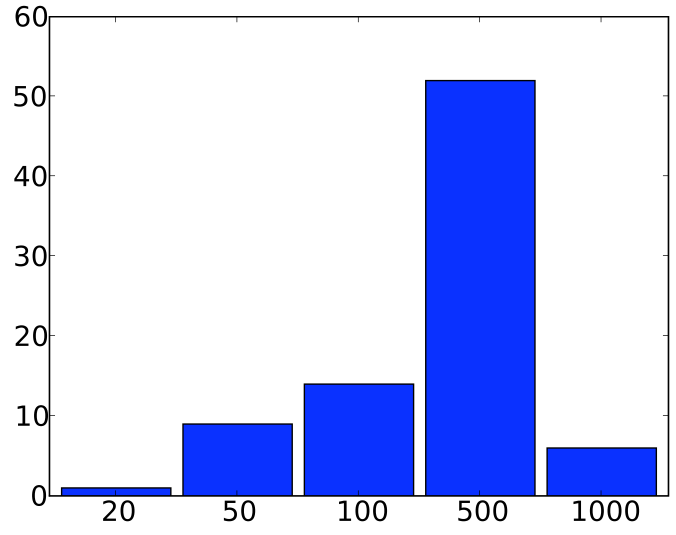


Figure C.1: *About how many emails do you receive a day?*
Users reported receiving 245 messages on average (200 median.) This is a much higher mail volume than was reported by Fisher *et al.* [99] in 2006 (87 average, 58 median) and Whittaker and Sidner [251] in 1996 (49 average.)

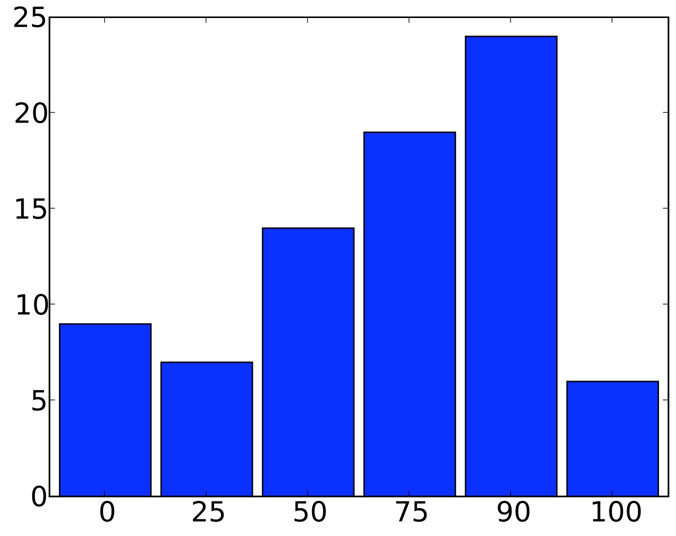


Figure C.2: *About what percentage of incoming emails skip your inbox (filter archives it)?*
Most mail users relied heavily on filters to remove mail from their inbox automatically. About 80% of users automatically filtered nearly the majority of their mail.

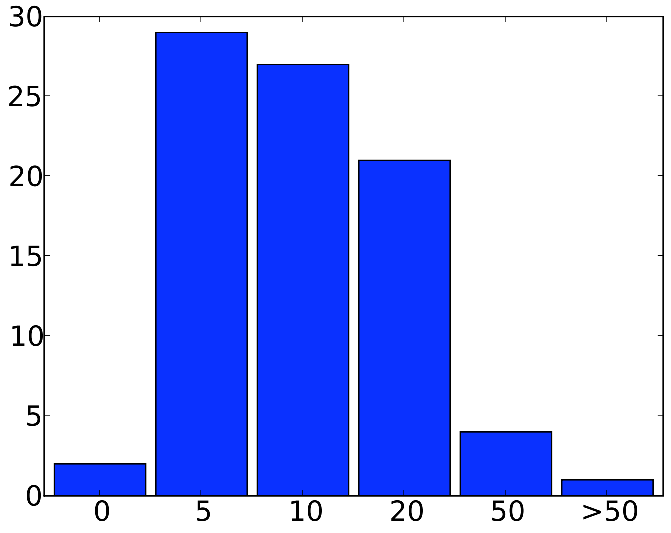


Figure C.3: *How many labels do you regularly use?*
Almost all users relied on labels in some way, with most users frequently using a handful to a few dozen labels.

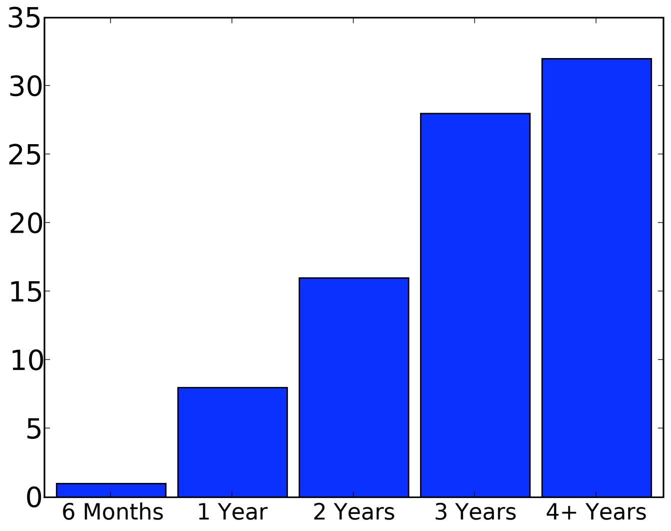


Figure C.4: *How long have you used Gmail?*
We had a very experienced user population as almost every user had at least one year of experience using their client.

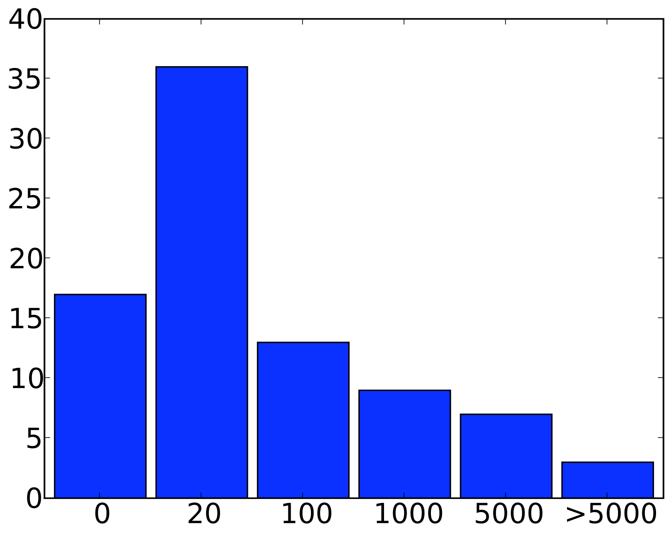


Figure C.5: *How many unread messages are in your inbox right now?*
There were three basic types of users. Those who worked to keep a very small inbox (close to 0,) those who kept moderate inboxes, and those with very large inboxes. We emphasize that these totals are *unread* messages and not total messages. Our users reported an average unread count of 554 messages (10 median.) Compared to Fisher *et al.* [99] we seem to have the same spread of users (they report median size of 7) but our large inbox users are significantly larger (they report average size of 153.)

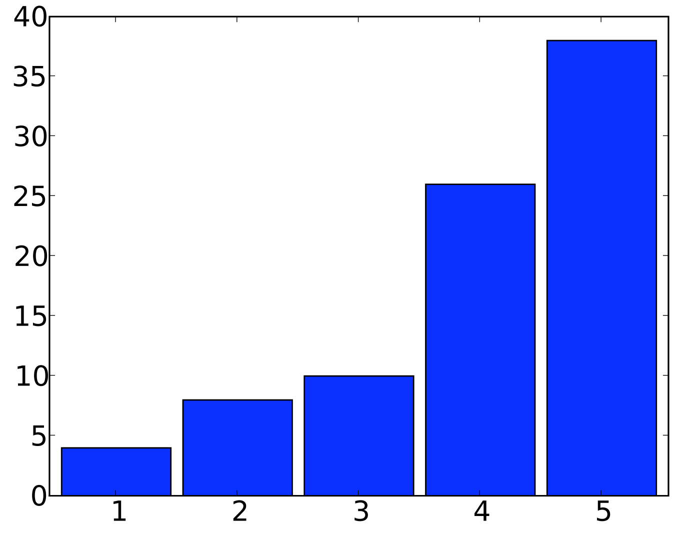


Figure C.6: *I like the idea of having search operators suggested.*
Answers given on a five-point Likert item.

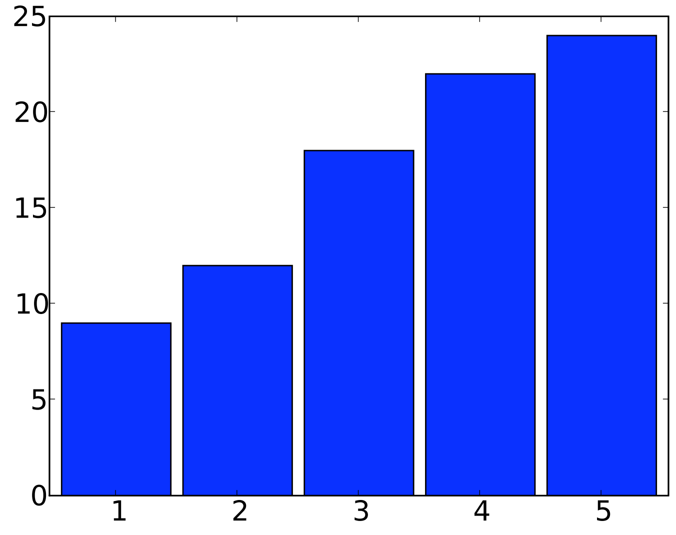


Figure C.7: *Filtering the inbox by search operators helps me manage my inbox.*
Answers given on a five-point Likert item.

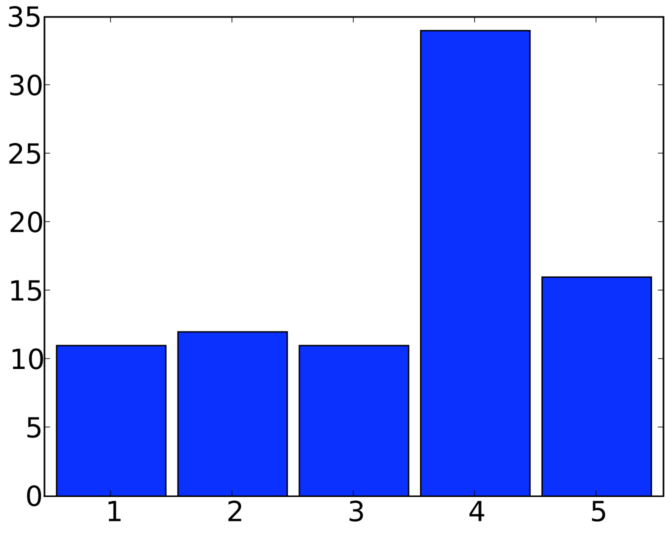


Figure C.8: *Suggesting operators for search results helps me find what I am searching for.* Answers given on a five-point Likert item.

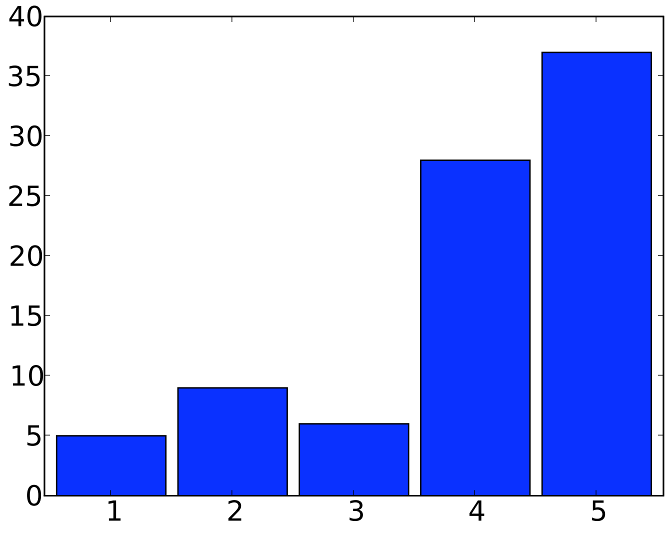


Figure C.9: *I would use a search operator suggestion tool (given high quality suggestions and a good UI.)*
Answers given on a five-point Likert item.

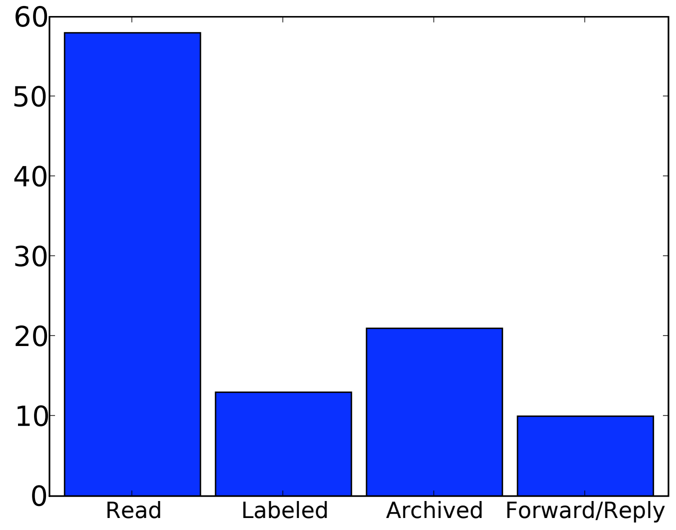


Figure C.10: *I typically did the following after selecting an operator: Read a message, Archived, Forward/Reply, Labeled messages*
Users selected zero or more applicable options. 64 users answered this question.

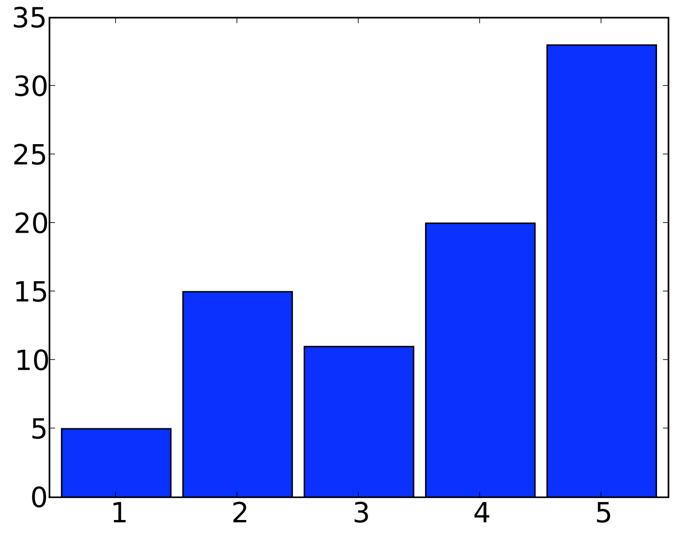


Figure C.11: *Highlighting emails that match an operator (mouseover operator) was helpful.*
Answers given on a five-point Likert item.

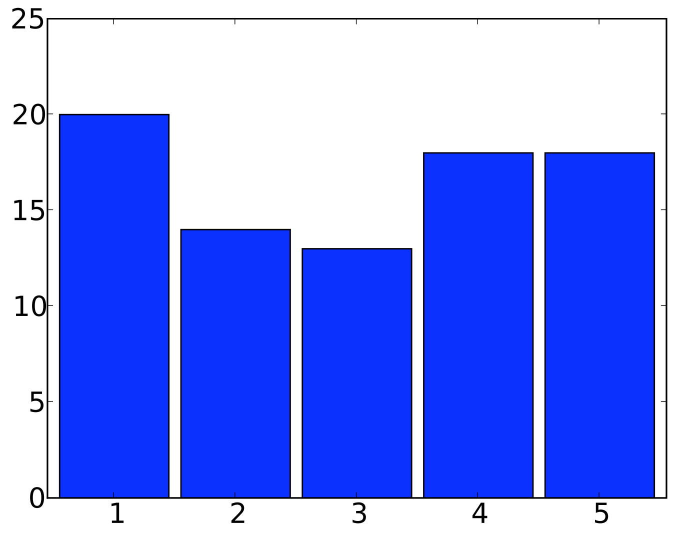


Figure C.12: *I have learned to use more operators in my Gmail searches.* Answers given on a five-point Likert item.

Bibliography

[1] Manu Aery and Sharma Chakravarthy. eMailSift: mining-based approaches to email classification. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, pages 580–581. ACM Press, 2004.

[2] Rie Kubota Ando and Lillian Lee. Iterative residual rescaling. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, pages 154–162. ACM Press, 2001.

[3] John Langshaw Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.

[4] Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multi–task learning. *Journal of Machine Learning Research (JMLR)*, 4:83–99, 2003.

[5] Ramnath Balasubramanyan, Vitor Carvalho, and William Cohen. Cutonce - recipient recommendation and leak detection in action. In *Conference on Email and Anti-Spam (CEAS)*, 2008.

[6] Olle Bälter and Candace L. Sidner. Bifrost inbox organizer: giving users control over the inbox. In *Nordic conference on Human-computer interaction (NordiCHI)*, pages 111–118, New York, NY, USA, 2002. ACM.

[7] Xinlong Bao, Jonathan L. Herlocker, and Thomas G. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *Intelligent User Interfaces (IUI)*, 2006.

[8] Paulo Barthelmess, Edward Kaiser, Rebecca Lunsford, David McGee, Philip Cohen, and Sharon Oviatt. Human-centered collaborative interaction. In *Workshop on Human-Centered Multimedia (HCM)*, 2006.

[9] Karin Becker, Michelle O. Cardoso, Caren M. Nichele, and Michele Frighetto. Mail-by-example: a visual query interface for email management. In *Brazilian Symposium on Databases*, 2003.

[10] Ron Bekkerman, Andrew McCallum, and Gary Huang. Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora. CIIR Technical Report IR-418, Department of Computer Science, University of Massachusetts, Amherst, 2004.

[11] Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel Bobrow, and Nicolas Ducheneaut. What a to-do: studies of task management towards the design of a personal task list manager. In *Conference on Human Factors in Computing Systems (CHI)*, 2004.

[12] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. Taskmaster: recasting email as task management. In *CSCW Workshop on Re-designing E-mail for the 21st Century*, 2002.

[13] 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 *Conference on Human Factors in Computing Systems (CHI)*, pages 345–352, 2003.

[14] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, Ian Smith, and Rebecca E. Grinter. Quality versus quantity: e-mail-centric task management and its relation with overload. *Human-Computer Interaction*, 20(1-2), 2005.

[15] Victoria Bellotti and James Thornton. Managing activities with tv-acta: Taskvista and activity-centered task assistant. In *SIGIR Workshop on Personal Information Management*, 2006.

[16] Victoria Bellotti, Jim Thornton, Alvin Chin, Diane Schiano, and Nathan Good. TV-ACTA: embedding an activity-centered interface for task management in email. In *Conference on Email and Anti-Spam (CEAS)*, 2007.

[17] Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[18] Steffen Bickel. Ecml-pkdd discovery challenge overview. In *The ECML-PKDD Discovery Challenge Workshop*, 2006.

[19] David Blei, Tom Griffiths, Michael Jordan, and Josh Tenebaum. Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[20] David Blei and John Lafferty. A correlated topic model of science. *Annals of Applied Statistics*, 1(1):17–35, 2007.

[21] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research (JMLR)*, 3:993–1022, 2003.

[22] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *Association of Computational Linguistics (ACL)*, 2007.

[23] Antoine Bordes and Leon Bottou. The huller: a simple and efficient online svm. In *European Conference on Machine Learning (ECML)*, 2005.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[25] Jake D. Brutlag and Christopher Meek. Challenges of the Email Domain for Text Classification. In *International Conference on Machine Learning (ICML)*, pages 103–110, 2000.

[26] Wray Buntine, Jaakko Löfström, Jukka Perkiö, Sami Perttu, Vladimir Poroshin, Tomi Silander, Henry Tirri, Antti Tuominen, and Ville Tuulos. A scalable topic-based open source search engine. In *IEEE/WIC/ACM Conference on Web Intelligence*, pages 228–234, 2004.

[27] Christopher Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[28] Stephan Busemann, Sven Schmeier, and Roman G. Arens. Message classification in the call center. In *Conference on Applied Natural Language Processing*, pages 158–165. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2000.

[29] Tom Bylander. A worst-case analysis of the perceptron and exponentiated update algorithms. *Artificial Intelligence*, 106(335–352), 1998.

[30] Giuseppe Carenini, Raymond Ng, and Xiaodong Zhou. Summarizing email conversations with clue words. In *World Wide Web (WWW)*, 2007.

[31] Giuseppe Carenini, Raymond T. Ng, and Xiaodong Zhou. Summarizing emails with conversational cohesion and subjectivity. In *Association for Computational Linguistics (ACL)*, pages 353–361, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[32] Xavier Carreras, Michael Collins, and Terry Koo. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Conference on Natural Language Learning (CONLL)*, 2008.

[33] Rich Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[34] Vitor Carvalho and William Cohen. Preventing information leaks in email. In *International Conference on Data Mining (SDM)*, 2007.

[35] Vitor Carvalho and William W. Cohen. On the collective classification of email speech acts. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2005.

[36] Vitor R. Carvalho and William Cohen. Recommending recipients in the enron email corpus. Technical Report CMU-LTI-07-005, Carnegie Mellon University, Language Technologies Institute, 2007.

[37] Vitor R. Carvalho and William W. Cohen. Single-pass online learning: Performance, voting schemes and online feature selection. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2006.

[38] Vitor R. Carvalho, Wen Wu, and William W. Cohen. Discovering leadership roles in email workgroups. In *Conference on Email and Anti-Spam (CEAS)*, 2007.

[39] Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. A second-order perceptron algorithm. *SIAM Journal on Computing*, 34(3):640 – 668, 2005.

[40] Nicolò Cesa-Bianchi, Phil M. Long, and Manfred K. Warmuth. Worst-case quadratic loss bounds for a generalization of the widrow-hoff rule. *IEEE Transactions on Neural Networks*, 7:604–619, 1996.

[41] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[42] Nicolò Cesa-Bianchi, Gábor Lugosi, and Gilles Stolt. Minimizing regret with label efficient prediction. *IEEE Transactions on Information Theory*, 51(6), June 2005.

[43] Yee Seng Chan and Hwee Tou Ng. Domain adaptation with active learning for word sense disambiguation. In *Association for Computational Linguistics (ACL)*, 2007.

[44] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[45] Ciprian Chelba and Alex Acero. Adaptation of max- imum entropy classifier: Little data can help a lot. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[46] Stanley F. Chen and Ronald Rosenfeld. A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99- 108, Carnegie Mellon University, 1999.

[47] Adam Cheyer, Jack Park, and Richard Giuli. IRIS: Integrate. Relate. Infer. Share. In *International Semantic Web Conference Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure*, 2005.

[48] David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[49] William W. Cohen. Learning rules that classify e-mail. In *AAAI Spring Symposium on ML and IR*, 1996.

[50] William W. Cohen, Vitor R. Carvalho, and Tom M. Mitchell. Learning to classify email into "speech acts". In *Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[51] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[52] Simon Corston-Oliver, Eric Ringger, Michael Gamon, and Richard Campbell. Task-focused summarization of email. In Stan Szpakowicz Marie-Francine Moens, editor, *ACL Workshop on Text Summarization Branches Out*, pages 43–50, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[53] Alex Cozzi, Tom Moran, and Clemens Drews. The shared checklist: Reorganizing the user experience around unified activities. In *International Conference on Human-Computer Interaction (INTERACT)*, Sept. 2005.

[54] Koby Crammer. *Online Learning of Complex Categorial Problems*. PhD thesis, Hebrew Univeristy of Jerusalem, 2004.

[55] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research (JMLR)*, 2006.

[56] Koby Crammer and Yoram Singer. Algorithmic implementation of multi-class svms. *Journal of Machine Learning Research (JMLR)*, 2001.

[57] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. In *Journal of Machine Learning Research (JMLR)*, 2003.

[58] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research (JMLR)*, 3:951–991, 2003.

[59] Nick Craswell, Arjen P. de Vries, and Ian Soboroff. Overview of the TREC 2005 enterprise track. In *Text REtrieval Conference (TREC)*, 2005.

[60] Elisabeth Crawford, Judy Kay, and Eric McCreath. An intelligent interface for sorting electronic mail. In *Intelligent User Interfaces (IUI)*, 2002.

[61] Gabor Cselle, Keno Albrecht, and Roger Wattenhofer. Buzztrack: topic detection and tracking in email. In *Intelligent User Interfaces (IUI)*, 2007.

[62] Aron Culotta, Ron Bekkerman, and Andrew McCallum. Extracting social networks and contact information from email and the web. In *Conference on Email and Anti-Spam (CEAS)*, 2004.

[63] Edward Cutrell, Susan T. Dumais, and Jaime Teevan. Searching to reduce the need for personal information management. *Communications of the ACM*, 49(1):58–64, 2006.

[64] Edward Cutrell, Daniel C. Robbins, Susan T. Dumais, and Raman Sarin. Fast, flexible filtering with Phlat - personal search and organization made easy. In *Conference on Human Factors in Computing Systems (CHI)*, 2006.

[65] Laura Dabbish, Robert Kraut, Susan Fussell, and Sara Kiesler. Understanding email usage: Predicting action on a message. In *Conference on Human Factors in Computing Systems (CHI)*, 2005.

[66] Wisam Dakka, Rishabh Dayal, and Panagiotis G. Ipeirotis. Automatic discovery of useful facet terms. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2006.

[67] Angelo Dalli, Yunqing Xia, and Yorick Wilks. Fasil email summarisation system. In *Conference on Computational Linguistics (COLING)*, 2004.

[68] Catalina Danis, Wendy Kellogg, Tessa Lau, Mark Dredze, Jeffrey Stylos, and Nicholas Kushmerick. Managers email: Beyond tasks and to-dos. In *Conference on Human Factors in Computing Systems (CHI)*, 2005.

[69] Sanjoy Dasgupta, Adam Tauma Kalai, and Claire Monteleoni. Analysis of perceptron-based active learning. In *Conference on Learning Theory (COLT)*, 2005.

[70] Hal Daumé. Frustratingly easy domain adaptation. In *Association for Computational Linguistics (ACL)*, 2007.

[71] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[72] Ofer Dekel, Philip M. Long, and Yoram Singer. Online multitask learning. In *Conference on Learning Theory (COLT)*, 2006.

[73] Christopher Diehl, Lise Getoor, and Galileo Mark Namata. Name reference resolution in organizational email archives. In *International Conference on Data Mining (SDM)*, 2006.

[74] Christopher Diehl, Galileo Mark Namata, and Lise Getoor. Relationship identification for social network discovery. In *American National Conference on Artificial Intelligence (AAAI)*, July 2007.

[75] Jana Diesner and Kathleen M. Carley. Exploration of communication networks from the enron email corpus. In *SIAM Workshop on Link Analysis, Counterterrorism and Security*, pages 3–14, 2005.

[76] Chuong B. Do and Andrew Ng. Transfer learning for text classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[77] Byron Dom, Iris Eiron, Alex Cozzi, and Yi Zhang. Graph-based ranking algorithms for e-mail expertise analysis. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.

[78] Anton Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jon L. Herlocker. TaskTracer: a desktop environment to support multi-tasking knowledge workers. In *Intelligent User Interfaces (IUI)*, pages 75–82. ACM Press, 2005.

[79] Mark Dredze, John Blitzer, Koby Crammer, and Fernando Pereira. Feature design for transfer learning. In *North East Student Colloquium on Artificial Intelligence (NESCAI)*, 2006.

[80] Mark Dredze, John Blitzer, and Fernando Pereira. Reply expectation prediction for email management. In *Conference on Email and Anti-Spam (CEAS)*, 2005.

[81] Mark Dredze, John Blitzer, and Fernando Pereira. "Sorry, I forgot the attachment:" Email attachment prediction forgot the attachment:" email attachment prediction. In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[82] Mark Dredze, Tova Brooks, Josh Carroll, Joshua Magarick, John Blitzer, and Fernando Pereira. Intelligent email: Reply and attachment prediction. In *Intelligent User Interfaces (IUI)*, 2008.

[83] Mark Dredze and Koby Crammer. Active learning with confidence. In *Association for Computational Linguistics (ACL)*, 2008.

[84] Mark Dredze and Koby Crammer. Online methods for multi-domain learning and adaptation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[85] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *International Conference on Machine Learning (ICML)*, 2008.

[86] Mark Dredze, Tessa Lau, and Nicholas Kushmerick. Automatically classifying emails into activities. In *Intelligent User Interfaces (IUI)*, 2006.

[87] Mark Dredze and Hanna Wallach. User models for email activity management. In *IUI Workshop on Ubiquitous User Modeling*, 2008.

[88] Mark Dredze, Hanna Wallach, Danny Puller, Tova Brooks, Josh Carroll, Joshua Magarick, John Blitzer, and Fernando Pereira. Intelligent email: Aiding users with AI. In *American National Conference on Artificial Intelligence (AAAI), NECTAR Paper*, 2008.

[89] Mark Dredze, Hanna Wallach, Danny Puller, and Fernando Pereira. Generating summary keywords for emails using topics. In *Intelligent User Interfaces (IUI)*, 2008.

[90] Nicolas Ducheneaut and Victoria Bellotti. Email as habitat: An exploration of embedded personal information management. *ACM Interactions*, 8(5):30–38, September-October 2001.

[91] Nicolas Ducheneaut and Victoria Bellotti. Ceci n'est pas un objet? talking about objects in email. *Human Computer Interaction*, 18(1–2):85–110, 2003.

[92] Susan T. Dumais. LSI meets TREC: A status report. In *Text REtrieval Conference (TREC)*, pages 137–152, 1992.

[93] Susan T. Dumais, Edward Cutrell, J. J. Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. Stuff I've Seen: A system for personal information retrieval and re-use. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2003.

[94] Tamer Elsayed and Douglas W. Oard. Modeling identity in archival collections of email: A preliminary study. In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[95] Tamer Elsayed, Douglas W. Oard, and Galileo Namata. Resolving personal names in email using context expansion. In *Association for Computational Linguistics (ACL)*, 2008.

[96] Jennifer English, Marti Hearst, Rashmi Sinha, Kirsten Swearingen, and Ka-Ping Yee. Flexible search and navigation using faceted metadata. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2002.

[97] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[98] Michael Fink, Shai Shalev-Shwartz, Yoram Singer, and Shimon Ullman. Online multiclass learning by interclass hypothesis sharing. In *International Conference on Machine Learning (ICML)*, 2006.

[99] Danyel Fisher, A.J. Brush, Eric Gleave, and Marc A. Smith. Revisiting Whittaker & Sidner's "email overload" ten years later. In *Computer Supported Cooperative Work (CSCW)*, 2006.

[100] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Conference on Computational Natural Language Learning (CONLL)*, 2003.

[101] Michael Freed, Jaime G. Carbonell, Geoff Gordon, Jordan Hayes, Brad A. Myers, Daniel P. Siewiorek, Stephen Smith, Aaron Steinfeld, and Anthony Tomasic. Radar: A personal assistant that learns to reduce email overload. In *American National Conference on Artificial Intelligence (AAAI)*, 2008.

[102] Ko Fujimura, Hiroyuki Toda, Takafumi Inoue, Nobuaki Hiroshima, Ryoji Kataoka, and Masayuki Sugizaki. BLOGRANGER - a multi-faceted blog search engine. In *World Wide Web (WWW)*, 2006.

[103] Alexander Genkin, David D. Lewis, and David Madigan. Large-scale Bayesian logistic regression for text categorization. Technical report, Department of Statistics, Rutgers University, 2004. `http://www.stat.rutgers.edu/~madigan/PAPERS/techno-06-09-18.pdf`.

[104] Melinda Gervasio, Thomas J. Lee, and Steven Eker. Learning email procedures for the desktop. In *AAAI Workshop on Enhanced Messaging*, 2008.

[105] Melinda T. Gervasio, Michael D. Moffitt, Martha E. Pollack, Joseph M. Taylor, and Tomas E. Uribe. Active preference learning for personalized calendar scheduling assistance. In *Intelligent User Interfaces (IUI)*, 2005.

[106] Werner Geyer, Juergen Vogel, Li-Te Cheng, and Michael Muller. Supporting activity-centric collaboration through peer-to-peer shared objects. In *International Conference on Supporting Group Work*, 2003.

[107] Alyssa Glass, Deborah McGuinness, and Michael Wolverton. Toward establishing trust in adaptive agents. In *Intelligent User Interfaces (IUI)*, 2008.

[108] Yihong Gong and Xin Liu. Generic text summarization using relevance measure and latent semantic analysis. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, pages 19–25, 2001.

[109] Tom Griffiths and Mark Steyvers. A probabilistic approach to semantic representation. In *Conference of the Cognitive Society*, 2002.

[110] Tom Griffiths, Mark Steyvers, David Blei, and Josh Tenebaum. Integrating topics and syntax. In *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[111] Jacek Gwizdka. TaskView–Design and evaluation of a task-based email interface. In *IBM Centers for Advanced Studies Conference (CASCON)*, 2002.

[112] Jacek Gwizdka. Email task management styles: the cleaners and the keepers. In *Conference on Human Factors in Computing Systems (CHI)*, pages 1235–1238, 2004.

[113] Jacek Gwizdka and Mark Chignell. Individual differences and task-based user interface evaluation: A case study of pending tasks in email. *Interacting with Computers*, 16(4):769–797, 2004.

[114] Edward Harrington, Ralf Herbrich, Jyrki Kivinen, John Platt, and Robert C. Williamson. Online bayes point machines. In *7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2003.

[115] Edward F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *International Conference on Machine Learning (ICML)*, 2003.

[116] Marti Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4), 2006.

[117] Ralf Herbrich and Thore Graepel. Bayes point machines. *Journal of Machine Learning Research (JMLR)*, 1:245–279, 2001.

[118] Thomas Hoffman. Probabilistic latent semantic analysis. In *Uncertainty in Artificial Intelligence (UAI)*, 1999.

[119] Ralf Hölzer, Bradley Malin, and Latanya Sweeney. Email alias detection using social network analysis. In *KDD Workshop on Link discovery (LinkKDD)*, pages 52–57, New York, NY, USA, 2005. ACM.

[120] Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive alerting. In *Uncertainty in Artificial Intelligence (UAI)*, pages 305–313. Morgan Kaufmann, 1999.

[121] Yifen Huang, Dinesh Govindaraju, Tom Mitchell, Vitor Carvalho, and William Cohen. Inferring ongoing activities of workstation users by clustering email. In *Conference on Email and Anti-Spam (CEAS)*, July 2004.

[122] Jason Rennie Tommi Jaakkola. Automatic Feature Induction for Text Classification. MIT Artificial Intelligence Laboratory Abstract Book, 2002.

[123] Aleks Jakulin. *Machine Learning Based on Attribute Interactions*. PhD thesis, Columbia University, 2005.

[124] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *Association for Computational Linguistics (ACL)*, 2007.

[125] Thorsten Joachims, Laura Granka, Bin Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, pages 154–161. ACM New York, NY, USA, 2005.

[126] Mika Kaki. Findex: Search result categories help users when document rankings fail. In *Conference on Human Factors in Computing Systems (CHI)*, 2005.

[127] Amy K. Karlson, George G. Robertson, Daniel C. Robbins, Mary P. Czerwinski, and Greg R. Smith. FaThumb: a facet-based interface for mobile search. In *Conference on Human Factors in Computing Systems (CHI)*, 2006.

[128] Bernard Kerr. Thread arcs: An email thread visualization. In *INFOVIZ*, 2003.

[129] Bernard Kerr and Eric Wilcox. Designing remail: reinventing the email client through innovation and integration. In *Conference on Human Factors in Computing Systems (CHI)*, pages 837–852, New York, NY, USA, 2004. ACM.

[130] Rinat Khoussainov and Nicholas Kushmerick. Email task management: An iterative relational learning approach. In *Conference on Email and Anti-Spam (CEAS)*, 2005.

[131] Seokhwan Kim, Yu Song, Kyungduk Kim, Jeong-Won Cha, and Gary Geunbae Lee. Mmr-based active machine learning for bio named entity recognition. In *North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, 2006.

[132] Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. In *Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 192–201. IBM Press, 2001.

[133] Svetlana Kiritchenko, Stan Matwin, and Suhayya Abu-Hakima. Email classification with temporal features. In *Intelligent Information Systems, New Trends in Intelligent Information Processing and Web Mining (IIPWM)*, pages 523–534. Springer Verlag, 2004.

[134] Josef Kittler, Mohamad Hatef, Rober P.W. Duin, and Jiri Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, Mar 1998.

[135] Bryan Klimt and Yiming Yang. The Enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning (ECML)*, 2004.

[136] Christian Kohlschutter, Paul-Alexandru Chirita, and Wolfgang Nejdl. Using link analysis to identify aspects in faceted web search. In *SIGIR Workshop on Faceted Search*, 2006.

[137] Leila Kosseim, Stéphane Beauregard, and Guy Lapalme. Using information extraction and natural language generation to answer e-mail. *Data & Knowledge Engineering*, 38, 2001.

[138] Nicholas Kushmerick and Tessa Lau. Automated email activity management: an unsupervised learning approach. In *Intelligent User Interfaces (IUI)*, pages 67–74. ACM Press, 2005.

[139] Nicholas Kushmerick, Tessa Lau, Mark Dredze, and Rinat Khoussainov. Activity-centric email: A machine learning approach. In *American National Conference on Artificial Intelligence (AAAI)*, 2006.

[140] Derek Lam, Steven L. Rohall, Chris Schmandt, and Mia K. Stern. Exploiting email structure to improve summarization. In *Computer Supported Cooperative Work (CSCW), Poster*, 2002.

[141] Andrew Lampert, Robert Dale, and Cécile Paris. Classifying speech acts using verbal response modes. In *Australasian Language Technology Workshop*, 2006.

[142] Andrew Lampert, Robert Dale, and Cécile Paris. The nature of requests and commitments in email messages. In *AAAI Workshop on Enhanced Messaging*, 2008.

[143] Andrew Lampert, Cécile Paris, and Robert Dale. Can requests-for-action and commitments-to-act be reliably identified in email messages? In *Australasian Document Computing Symposium*, pages 48–55, 2007.

[144] A. Twinkle Lettkeman, Simone Stumpf, Jed Irvine, and Jon Herlocker. Predicting task-specific webpages for revisiting. In *American National Conference on Artificial Intelligence (AAAI)*, 2006.

[145] Anton Leuski. Email is a stage: discovering people roles from email archives. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, pages 502–503. ACM New York, NY, USA, 2004.

[146] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 1994.

[147] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research (JMLR)*, 5:361–397, 2004.

[148] Lida Li, Michael J. Muller, Werner Geyer, Casey Dugan, Beth Brownholtz, and David R. Millen. Predicting individual priorities of shared activities using support vector machines. In *Conference on information and knowledge management (CIKM)*, pages 515–524, New York, NY, USA, 2007. ACM.

[149] Wei Li and Andrew McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *International Conference on Machine Learning (ICML)*, 2006.

[150] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

[151] Dong C. Liu and Jorge Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming B*, 45(3):503–528, 1989.

[152] Andrea Lockerd and Ted Selker. Driftcatcher: The implicit social context of email. In *International Conference on Human-Computer Interaction (INTERACT)*, pages 813–816, 2003.

[153] Tiago C. Marques, Vasco Furtado, and Deborah L. McGuinness. Explaining social relationships. In *AAAI Workshop on Enhanced Messaging*, 2008.

[154] Zvika Marx, Michael T. Rosenstein, Thomas G. Dietterich, and Leslie Pack Kaelbling. Two algorithms for transfer learning. In *Inductive Transfer: 10 years later*. 2008.

[155] Andrew McCallum. MALLET: A machine learning for language toolkit. `http://mallet.cs.umass.edu`, 2002.

[156] Andrew McCallum. Efficiently inducing features of conditional random fields. In *Uncertainty in Artificial Intelligence (UAI)*, 2003.

[157] Andrew McCallum, Andres Corrada-Emmanuel, and Xuerui Wang. Topic and role discovery in social networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[158] Andrew McCallum, Xuerui Wang, and Andres Corrada-Emmanuel. Topic and role discovery in social networks with experiments on Enron and academic email. In *Journal of Artificial Intelligence Research (JAIR)*, 2007.

[159] Ryan McDonald, Koby Crammer, Kuzman Ganchev, Surya Prakash Bachoti, and Mark Dredze. Penn StructLearn. http://www.seas.upenn.edu/ strctlrn/StructLearn/StructLearn.html, 2006.

[160] Ryan McDonald, Koby Crammer, and Fernando Pereira. Large margin online learning algorithms for scalable structured classification. In *NIPS Workshop on Structured Outputs*, 2004.

[161] Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2005.

[162] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Association for Computational Linguistics (ACL)*, 2005.

[163] Luke McDowell, Oren Etzioni, and Alon Halevy. Semantic email: Theory and applications. *Journal of Web Semantics*, 2(2):153–183, 2004.

[164] Luke McDowell, Oren Etzioni, Alon Halevy, and Henry Levy. Semantic email. In *World Wide Web (WWW)*, pages 244–254. ACM Press, 2004.

[165] Kathleen McKeown, Lokesh Shrestha, and Owen Rambow. Using question-answer pairs in extractive summarization of email conversations. In *Conference on Intelligent Text Processing and Computational Linguistics (CICLING)*, 2007.

[166] David Mimno, Wei Li, and Andrew McCallum. Mixtures of hierarchical topics with pachinko allocation. In *International Conference on Machine Learning (ICML)*, 2007.

[167] Einat Minkov, Ramnath Balasubramanyan, and William W. Cohen. Activity-centric search in email. In *AAAI Workshop on Enhanced Messaging*, 2008.

[168] Einat Minkov, William W. Cohen, and Andrew Y. Ng. Contextual search and name disambiguation in email using graphs. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2006.

[169] Tom Mitchell, Sophie Wang, Yifen Huang, and Adam Cheyer. Extracting knowledge about users' activities from raw workstation contents. In *American National Conference on Artificial Intelligence (AAAI)*, 2006.

[170] Kenrick Mock. An experimental framework for email categorization and management. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2001.

[171] Marty Moore, Miguel Estrada, Timothy Finley, Michael Muller, and Werner Geyer. Next generation activity-centric computing. In *Computer Supported Cooperative Work (CSCW), Poster*, 2006.

[172] Thomas P. Moran, Alex Cozzi, and Stephen P. Farrell. Unified Activity Management: Supporting People in eBusiness. *Communications of the ACM*, 2005.

[173] Michael Muller and Dan Gruen. Collaborating Within–not Through–Email: Users Reinvent a Familiar Technology. *Computer Supported Cooperative Work (CSCW), Poster*, 2002.

[174] 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 *Conference on Human Factors in Computing Systems (CHI)*, pages 375–382. ACM Press, 2004.

[175] Hiroyuki Murakoshi, Akira Shimazu, and Koichiro Ochimizu. Construction of deliberation structure in email communication. In *Pacific Association for Computational Linguistics (PACLING)*, 1999.

[176] Smaranda Muresan, Evelyne Tzoukermann, and Judith L. Klavans. Combining linguistic and machine learning techniques for email summarization. In *Conference on Natural Language Learning (CONLL)*, 2001.

[177] Galileo Mark Namata, Lise Getoor, and Christopher Diehl. Inferring organizational titles in online communications. In *ICML Workshop on Statistical Network Analysis*, 2006.

[178] Ani Nenkova and Amit Bagga. Email classification for contact centers. In *ACM Symposium on Applied Computing*, pages 789–792, 2003.

[179] Ani Nenkova and Amit Bagga. Facilitating email thread access by extractive summary generation. In *Recent Advances in Natural Language Processing (RANLP)*, 2003.

[180] Carman Neustaedter, A.J. Brush, and Marc Smith. Beyond ”from” and ”received”: Exploring the dynamics of email triage. In *Conference on Human Factors in Computing Systems (CHI)*, 2005.

[181] Carman Neustaedter, A.J. Bernheim Brush, Marc A. Smith, and Danyel Fisher. The social network and relationship finder: Social sorting for email triage. In *Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, 2005.

[182] Paula S. Newman. Email archive overviews using subject indexes. In *Conference on Human Factors in Computing Systems (CHI)*, 2002.

[183] Paula S. Newman and John C. Blitzer. Summarizing archived discussions: a beginning. In *Intelligent User Interfaces (IUI)*, 2003.

[184] Eyal Oren, Renaud Delbru, and Stefan Decker. Extending faceted navigation for RDF data. In *International Semantic Web Conference (ISWC)*, 2006.

[185] Chris Pal and Andrew McCallum. CC prediction with graphical models. In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[186] Kayur Patel, James Fogarty, James A. Landay, and Beverly Harrison. Investigating statistical machine learning as a tool for software development. In *Conference on Human Factors in Computing Systems (CHI)*, 2008.

[187] Terry R. Payne and Peter Edwards. Interface agents that learn: An investigation of learning issues in a mail agent interface. *Applied Artificial Intelligence*, 11(1):1–32, 1997.

[188] Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 167–176. ACM New York, NY, USA, 2003.

[189] Kaare Brandt Petersen and Michael Syskind Pedersen. *The Matrix Cookbook*, 2007.

[190] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[191] Jefferson Provost. Naive-Bayes vs. Rule-learning in classification of Email [R]. Texas: The University of Texas at Austin. Technical Report TR-99-284, University of Texas at Austin, Artificial Intelligence Lab, 1999.

[192] Dragomir R. Radev, Hongyan Jing, Malgorzata Stys, and Daniel Tam. Centroid-based summarization of multiple documents. *Information Processing and Management*, 40:919–938, December 2004.

[193] Filip Radlinski and Thorsten Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *American National Conference on Artificial Intelligence (AAAI)*, 2005.

[194] Rajat Raina, Andrew Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *International Conference on Machine Learning (ICML)*, 2006.

[195] Owen Rambow, Lokesh Shrestha, John Chen, and Chirsty Lauridsen. Summarizing email threads. In *North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, 2004.

[196] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[197] Adwait Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing. Technical Report 97–08, Institute for Research in Cognitive Science, 1997.

[198] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998.

[199] Jason Rennie. IFile: An Application of Machine Learning to E-Mail Filtering. In *KDD Workshop on Text Mining*, pages 95–98. New York: Association for Computering Machinery, 2000.

[200] Eric Ringger, Peter McClanahan, Robbie Haertel, George Busby, Marc Carmen, James Carroll, Kevin Seppi, and Deryle Lonsdale. Active learning for part-of-speech tagging: Accelerating corpus annotation. In *ACL Linguistic Annotation Workshop*, 2007.

[201] 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 *Conference on Human Factors in Computing Systems (CHI)*, pages 791–792, New York, NY, USA, 2004. ACM.

[202] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Cornell Aeronautical Laboratory, Psychological Review*, 65(6):386–408, 1958.

[203] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *International Conference on Machine Learning (ICML)*, 2001.

[204] Jade Goldstein Roberta Evans Sabin. Using Speech Acts to Categorize Email and Identify Email Genres. In *HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*, volume 39, page 50. IEEE, 2006.

[205] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[206] Mark Sanderson and Susan Dumais. Examining repetition in user search behavior. In *European Conference on Information Retrieval (ECIR)*, 2007.

[207] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. *Advances in Neural Information Processing Systems (NIPS)*, 17:1185–1192, 2005.

[208] Manabu Sassano. An empirical study of active learning with support vector machines for japanese word segmentation. In *Association for Computational Linguistics (ACL)*, 2002.

[209] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[210] D. Sculley. Online active learning methods for fast label-efficient spam filtering. In *Conference on Email and Anti-Spam (CEAS)*, 2007.

[211] John. R. Searle. *Speech Acts*. Cambridge University Press, London, 1969.

[212] Richard Segal, Jason Crawford, Jeff Kephart, and Barry Leiba. SpamGuru: An Enterprise Anti-Spam Filtering System. In *Conference on Email and Anti-Spam (CEAS)*, July 2004.

[213] Richard Segal and Jeff Kephart. Incremental Learning in SwiftFile. In *International Conference on Machine Learning (ICML)*, pages 863–870, San Francisco, CA, 2000.

[214] Richard B. Segal and Jeffrey O. Kephart. MailCat: An intelligent assistant for organizing e-mail. In *American National Conference on Artificial Intelligence (AAAI)*, 1999.

[215] Jianqiang Shen, Werner Geyer, Michael Muller, Casey Dugan, Beth Brownholtz, and David R Millen. Automatically finding and recommending resources to support

knowledge workers' activities. In *Intelligent User Interfaces (IUI)*, pages 207–216, New York, NY, USA, 2008. ACM.

[216] Jianqiang Shen, Lida Li, Thomas G. Dietterich, and Jonathan L. Herlocker. A hybrid learning system for recognizing user tasks from desk activities and email messages. In *Intelligent User Interfaces (IUI)*, 2006.

[217] Libin Shen and Aravind Joshi. Ranking and reranking with perceptron. *Machine Learning*, 60, 2005.

[218] Libin Shen, Giorgio Satta, and Aravind K. Joshi. Guided learning for bidirectional sequence classification. In *Association for Computational Linguistics (ACL)*, 2007.

[219] Pannagadatta K. Shivaswamy and Tony Jebara. Ellipsoidal kernel machines. In *Artificial Intelligence and Statistics (AISTATS)*, 2007.

[220] Lokesh Shrestha and Kathleen McKeown. Detection of question-answer pairs in email conversations. In *Conference on Computational Linguistics (COLING)*, 2004.

[221] H. Gregory Silber and Kathleen F. McCoy. Efficiently Computed Lexical Chains as an Intermediate Representation for Automatic Text Summarization. *Computational Linguistics*, 28(4):487–496, 2002.

[222] Greg Smith, Mary Czerwinski, Brian Meyers, Daniel Robbins, George Robertson, and Desney S. Tan. FacetMap: A Scalable Search and Browse Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):797–804, September/October 2006.

[223] Ian Soboroff, Arjen P. de Vries, and Nick Craswell. Overview of the TREC 2006 enterprise track. In *Text REtrieval Conference (TREC)*, 2006.

[224] Aaron Steinfeld, S. Rachael Bennett, Kyle Cunningham, Matt Lahut, Pablo-Alejandro Quinones, Django Wexler, Daniel Siewiorek, Jordan Hayes, Paul Cohen,

Julie Fitzgerald, Othar Hansson, Mike Pool, and Mark Drummond. Evaluation of an integrated multi-task machine learning system with humans in the loop. In *NIST Performance Metrics for Intelligent Systems Workshop (PerMIS)*, 2007.

[225] Mark Steyvers and Tom Griffiths. Probabilistic topic models. In D McNamara, S Dennis, and W Kintsch, editors, *Latent Semantic Analysis: A Road to Meaning*. Laurence Erlbaum, in press.

[226] Emilia Stoica and Marti Hearst. Nearly-automated metadata hierarchy creation. In *North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, 2004.

[227] Emilia Stoica, Marti A. Hearst, and Megan Richardson. Automating creation of hierarchical faceted metadata structures. In *North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, 2007.

[228] Salvatore J. Stolfo, German Creamer, and Shlomo Hershkop. A temporal based forensic analysis of electronic communication. In *Digital Government Proceedings*, 2006.

[229] Salvatore J. Stolfo, Shlomo Hershkop, Chia-Wei Hu, Wei-Jen Li, Olivier Nimeskern, and Ke Wang. Behavior-based modeling and its application to email analysis. *ACM Transactions on Internet Technology*, 6(2):187–221, May 2006.

[230] Simone Stumpf, Xinlong Bao, Anton Dragunov, Thomas G. Dietterich, Jon Herlocker, Kevin Johnsrude, Lida Li, and JianQiang Shen. The tasktracer system. In *American National Conference on Artificial Intelligence (AAAI), Intelligent Systems Demonstrations*, 2005.

[231] Simone Stumpf, Erin Sullivan, Erin Fitzhenry, Ian Oberst, Weng-Keen Wong, and Margaret Burnett. Integrating rich user feedback into intelligent user interfaces. In *Intelligent User Interfaces (IUI)*, 2008.

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

[233] Richard S. Sutton. Adapting bias by gradient descent: an incremental version of delta-bar-delta. In *American National Conference on Artificial Intelligence (AAAI)*, pages 171–176. MIT Press, 1992.

[234] John Tang, Eric Wilcox, Julian A. Cerruti, Hernan Badenes, Stefan Nusser, and Jerald Schoudt. Tag-it, snag-it, or bag-it: Combining tags, threads, and folders in e-mail. In *Conference on Human Factors in Computing Systems (CHI)*, 2008.

[235] Min Tang, Xiaoqiang Luo, and Salim Roukos. Active learning for statistical natural language parsing. In *Association for Computational Linguistics (ACL)*, 2002.

[236] David M. J. Tax, Martijn van Breukelen, Robert P. W. Duina, and Josef Kittler. Combining multiple classifiers by averaging or by multiplying? *Pattern Recognition*, 33(9):1475–1485, September 2000.

[237] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101, 2004.

[238] S. Thrun and J. O'Sullivan. Clustering learning tasks and the selective cross–task transfer of knowledge. In S. Thrun and L.Y. Pratt, editors, *Learning To Learn*. Kluwer Academic Publishers, 1998.

[239] Simon Tong and Daphne Koller. Supprt vector machine active learning with applications to text classification. *Journal of Machine Learning Research (JMLR)*, 2001.

[240] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, pages 823–830, 2004.

[241] Joshua Tyler, Dennis M. Wilkinson, and Bernardo A. Huberman. Email as spectroscopy: Automated discovery of community structure within organizations. In *Communities and Technologies (C&T)*. Kluwer Academic Publishers, 2003.

[242] Joshua R. Tyler and John C. Tang. When can i expect an email response? a study of rhythms in email usage. In *European Conference on Computer-Supported Cooperative Work (ECSCW)*, 2003.

[243] Gina Danielle Venolia, Laura Dabbish, J.J. Cadiz, and Anoop Gupta. Supporting email workflow. Technical Report MSR-TR-2001-88, Microsoft Research, 2001.

[244] Gina Danielle Venolia and Carman Neustaedter. Understanding sequence and reply relationships within email conversations: a mixed-model visualization. In *Conference on Human Factors in Computing Systems (CHI)*, pages 361–368, New York, NY, USA, 2003. ACM.

[245] Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *International Conference on Machine Learning (ICML)*, Pittsburgh, Pennsylvania, 2006.

[246] Hanna M. Wallacha. *Structured Topic Models for Language*. PhD thesis, University of Cambridge, 2008.

[247] Stephen Wan, Mark Dras, Cécile Paris, and Robert Dale. Using Thematic Information in Statistical Headline Generation. In *ACL Workshop on Multilingual Summarization and Question Answering*, 2003.

[248] Stephen Wan and Kathy McKeown. Generating overview summaries of ongoing email thread discussions. In *Conference on Computational Linguistics (COLING)*, 2004.

[249] Xing Wei and W. Bruce Croft. LDA-based document models for *Ad-hoc* retrieval. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2006.

[250] Steve Whittaker, Victoria Bellotti, and Paul Moody. Introduction to this special issue on revisiting and reinventing e-mail. *Human-Computer Interaction*, 20(1-2):1–9, 2005.

[251] Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *Conference on Human Factors in Computing Systems (CHI)*, 1996.

[252] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques, 2nd ed.* Morgan Kaufmann, 2005.

[253] Kevin Woods, W. Philip Kegelmeyer Jr., and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, 1997.

[254] Wensi Xi, Jesper Lind, and Eric Brill. Learning effective ranking functions for newsgroup search. In *ACM SIGIR Conference on Information Retrieval (SIGIR)*, 2004.

[255] Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Conference on Human Factors in Computing Systems (CHI)*, 2003.

[256] Klaus Zechner and Alon Lavie. Increasing the coherence of spoken dialogue summaries by cross-speaker information linking. In *NAACL Workshop on Automatic Summarization*, 2001.

[257] Dong Zhang, Daniel Gatica-Perez, Deb Roy, and Samy Bengio. Modeling interactions from email communications. In *IEEE International Conference on Multimedia and Expo*, 2006.

[258] Ding Zhou, Eren Manavoglu, Jia Li, C. Lee Giles, and Hongyuan Zha. Probabilistic models for discovering e-communities. In *World Wide Web (WWW)*, pages 173–182. ACM New York, NY, USA, 2006.