Suggesting Email View Filters for Triage and Search

Mark Dredze*
Human Language Technology Center of Excellence
Johns Hopkins University
Baltimore, MD 21211 USA
mdredze@cs.jhu.edu

Bill N. Schilit and **Peter Norvig**
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043 USA
{schilit,pnorvig}@google.com

Abstract

Growing email volumes cause flooded inboxes and swelled email archives, making search and new email processing difficult. While emails have rich metadata, such as recipients and folders, suitable for creating filtered views, it is often difficult to choose appropriate filters for new inbox messages without first examining messages. In this work, we consider a system that automatically suggests relevant view filters to the user for the currently viewed messages. We propose several ranking algorithms for suggesting useful filters. Our work suggests that such systems quickly filter groups of inbox messages and find messages more easily during search.

1 Introduction

Users spend a considerable amount of time organizing and searching their email mailboxes. Increasingly, users rely on fast and effective email search to find messages, forgoing careful message organization. Many users have a single folder containing all of their messages [Fisher *et al.*, 2006]. In fact Gmail 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, users struggle with triaging large inboxes, unable to organize the new information effectively. These two scenarios – email search and email triage – both require a simple way for a user to understand a long list of messages.

Consider a user who opens her mailbox to find 50 new messages. She can begin to skim the list looking for important emails or process the list chronologically. As she reads, she finds a collection of messages from a discussion list. Had she known these emails were present, she could have processed them as a group. Similarly, she searches her mailbox for emails from "john" looking for a message from "John Smith" but search results match many different Johns. In both cases, the user could view a specific group of matching message with a view filter, such as all inbox messages involving a discussion list. Unfortunately, thinking of filters can be difficult, especially without first examining the list of messages.

In this work, we propose automatically generating a list of view filters relevant to the displayed messages. Our filters are implemented as searches, such as a search for all messages in the inbox from a discussion list. We call our task *Search Operator Suggestion*, where search operators are special terms that retrieve emails based on message metadata, such as "from:john smith" and "is:starred." We build a mail filter system for Gmail (Google Mail) using search operators and develop several search operator rankers using features of the user, mailbox and machine learning. We validate our system on data collected from user interactions with our system. Results indicate that our rankers accurately suggest mail filters for supporting an intelligent user interface.

We begin with a discussion of email view filters. We then present the user interface and system for collecting usage data. Several ranking systems and features are evaluated on collected data; our rankers yield high quality suggestions. We include results of a user survey validating view filter suggestion. We conclude with related work and discussion.

2 Email View Filters

Several mail clients have features for displaying collections of messages based on metadata. Microsoft Outlook has search folders that show all messages matching a predefined query. Mozilla Thunderbird filters folders with mail views by read status, recipients, etc. Gmail supports view filters via the search, where users can specify queries over metadata and save these searches for later use.

A similar idea is faceted browsing, which filters objects based on metadata characteristics. Faceted browsing is popular for many e-commerce stores, filtering results based on properties, such as "brand" or "memory type." Facets are meaningful labels that convey different data characteristics to the user, allowing users to easily modify displayed results. When such metadata is available, a hierarchical faceted category system provides the advantage of clarity and consistency as compared to clustering systems [Hearst, 2006]. 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 [Kaki, 2005]. Several studies have shown improved user experiences because of this interaction [Oren *et al.*, 2006]. Such systems are popular and have been applied to web search, blog search, mobile search and images [Hearst, 2006]. Email view filters could provide these

*This work was completed when the first author was at Google.

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

benefits as well. During triage, filters select groups of related messages for processing. For search, filters offer a means of refining the query – a simple way to elicit user feedback for vague queries. Also like facets, view filters can summarize a list of messages, showing important groups that appear in the list of messages. This is similar to the use of social summaries for inbox triage [Neustaedter *et al.*, 2005].

There are several key differences between filters and facets. Facets are typically well defined labels in a category hierarchy. Furthermore, facet browsing typically shows all applicable facets for the data, such as FacetMap by Microsoft Research [Smith *et al.*, 2006]. However, Seek[2], an extension for Thunderbird that extends faceted browsing to email, generates hundreds of facets, too many to show the user.

Instead, we seek to bring the benefits of faceted browsing to email by suggesting view filters relevant to the user's current activity. While these filters indicate the contents of the current view and provide effective tools for navigation, they do not have the burden of defining and displaying a full facet hierarchy. To build these filters, we use the Gmail search system since it already provides robust support for searching email metadata through search operators. In the next section, we describe these operators and our system for generating and displaying them to the user.

3 Operator Suggestion System

We built a prototype search operator suggestion system for Gmail to collect data for evaluating rankers. Search operators are a natural way of incorporating filters into Gmail since they can easily retrieve all messages matching a specific metadata field. Gmail has several search operators: users can type "from:john", "to:me", "is:unread" and "has:attachment" to retrieve messages matching those requirements. Each operator has both a type (to, from, attachment, label, etc.) and a value (person name, label name, etc.) Person operators may be especially useful based on research in personal information management [Dumais *et al.*, 2003; Cutrell *et al.*, 2006]. A list of the operators used by our system appears in table 1. While these operator types combined with all possible values created a large number of operators, we favored wide exploration to solicit user feedback.

Gmail is thread (conversation) based. For the first 100 threads that match the current user view, [3] our system enumerates all possible search operators and their values using message metadata, including operators to match email domains (e.g. "from:google.com".) Operators that matched only a single thread were ignored. We then display 10 operators to the user to maximize the options available while keeping the list short enough to read quickly. A production system could select the number of operators to display based on a ranker.

To encourage use of the system for data collection, we desired helpful operators. However, since users click only on displayed operators, the collected data is biased to favor the selection heuristic. Therefore, a heuristic was combined with probabilistic sampling. Operators were scored according to the number of matching threads based on the idea that good operators match multiple messages. To counter the bias of always selecting frequently occurring operators, we randomized operator selection using the scores as a probability distribution. Given the number of threads matched by an operator o as N_o, operators were selected 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 display order.

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

[3] In Gmail, these are also the most recent threads.

Operator	*Definition*
from	Message sender
to	Message recipient
label	Messages with a given label
has:attachment	Messages with attachments
is:starred	Messages that are starred
is:unread	Messages that are unread
cc	Message cc
is:chat	Chat messages
is:draft	Draft messages

Table 1: The types of the search operators used by the search operator suggestion system. The system generates operators of these types paired with a value to create view filters. These operators are documented in the Gmail Help Center (http://mail.google.com/support/).

3.1 User Interface

When a user viewed a list of 10 or more threads in the inbox, search results, or any list of threads, our system displayed 10 search operators below the search box and above the thread list (figure 2.) Our motivation for suggestion placement was visibility to elicit clicks from users. Each operator is displayed as it would be typed in the search box. Each operator is a link that adds the operator to the search box to filter the current view to show only matching threads. For example, when a user selects "is:unread" in the inbox view, the system will show all unread threads in the inbox. Additionally, each operator has a "-" link to exclude threads that match the operator, such as "not to me" (-to:me.) Person operators used the person's name when available and the email address otherwise. We later added the ability to highlight matching threads by mousing over a suggested operator.

This system was deployed to volunteers who read a description of the experiment and the data collected. 183 volunteers joined and left the experiment over an eight week period. We collected data by logging each view request, the time, user, number of matching threads, search query and all search operators generated for the view, recording which were shown to the user, the number and position of matching threads and other properties of the operator useful for ranking. The system also logged user clicks for feedback. No information about message contents was logged.

4 Search Operator Ranking

We study the task of search operator ranking, whereby operators are ranked for display to the user. A ranking instance z is defined by the pair $z = (\mathbf{x}, y)$, where $\mathbf{x}$ is a set of search

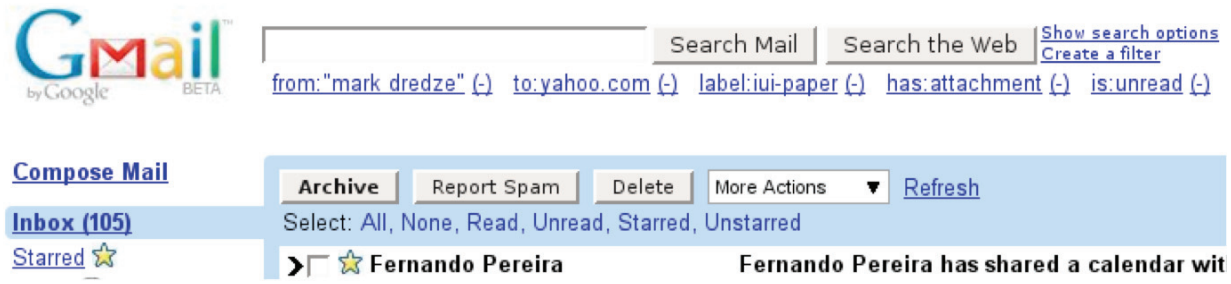

Figure 2: The search operator suggestion user interface for data collection shows suggested operators below the search box.

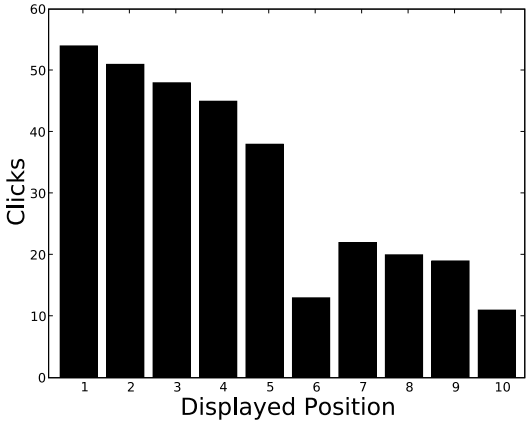

Figure 1: Number of clicks for search operators per display position. Users were twice as likely to click an operator in the first five positions compared to the last five, a strong but weaker bias than for web search where half of all clicks are on the first result and 80% of clicks are on the top 5 [Joachims *et al.*, 2005]. Our weaker bias may be due to the ease by which users could view other options: operators fit in a small space while web search results require scrolling through sentences.

operators and $y \in \mathbf{x}$ is the user selected operator. Operators $x \in \mathbf{x}$ are defined by a set of features. A ranking function $\mathcal{R}$ orders $\mathbf{x}$. 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 rank and $\mathcal{L}(\mathcal{R}, z) = 0$ otherwise.

4.1 Training Data

We created a ranking dataset from user interaction logs, extracting instances where the user clicked on a search operator or its negation or when operators were explicitly entered into the search box. The clicked operator was taken as y and all of the operators (shown and not shown) formed $\mathbf{x}$ to create a ranking instance z. While we collected 436 ranking examples from 183 users, most people were not heavy users and generated few examples. Few examples for a user may indicate initial system experimentation and not actual use. Therefore, we created a second dataset from users with at least 10 examples, with an average of 24 clicks per user and a total of 235 instances. We name the datasets *All* and *Min10* respectively. Since there is a limited amount of data we consider both datasets in our evaluations.

There are several potential issues with our data collection. 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 useful operators but could only select one. Finally, users are biased towards selecting operators earlier in the displayed list. These problems are common to many ranking problems, such as web search [Radlinski and Joachims, 2005]. Joachims et al.carefully studied the use of click through data for ranking using manual relevance judgments and eye tracking behavior, concluding that the relative preferences derived from clicks are reasonably accurate on average [Joachims *et al.*, 2005]. Therefore, we use click through data for our task.

A second problem is the bias introduced by our ranking heuristic. Users are more likely to click on operators that match many threads because they are shown more often. Nevertheless, we found that users selected operators that were not in the top ten of counts a significant portion of the time, indicating that our simple heuristic cannot effectively rank options; learning improves over this heuristic. We use this method of data collection as a compromise between accurate annotations and showing a reasonable number of options.

Our successful results will show that limited data quality and quantity 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 production systems. Furthermore, a production system that obtained more usage could improve due to improved data quality and quantity.

5 Ranking Systems

We develop several ranking algorithms for search operators and begin with some baselines to measure task difficulty.

5.1 Baseline Systems

Random - Ranks operators randomly, 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 their display order followed by a

random ranking of not shown operators. Since clicks were biased by display order (figure 1), this baseline should do well.
Max Count - Operators that match the most threads are ranked first, ie. the data collection heuristic, except without sampling or permutation. This measures data collection bias.

5.2 Ranking Systems

We consider three systems that rely on user behavior, the current mail view threads and properties of the operator values.
Split Results - Since mail filters are meant to extract groups of messages, useful operators should match a sizable group of messages but not a large group, splitting the data evenly (unlike *Max Count*.) For example, an operator that matches two or 99 threads out of 100 is likely not useful. This ranker orders operators by how well they split the data: 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.

Sanderson and Dumais found that user search behavior is repetitive, so previous operator use may be indicative of future use [Sanderson and Dumais, 2007]. 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."

We build a user history model based on previous user searches to capture operator popularity. The history records the terms used in all user searches and lookups indicate both how often an operator was used in general and in the context of a query. We implemented three popularity 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 approaches provide a range of flexibility for measuring popularity. The first is context insensitive, the second mildly context sensitive and the third requires context. The history is built independently for each user from training data. Ties between operators are broken randomly.

Machine Learning - We combine information from user history and the threads in the view into a single ranker. We develop 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 of intrinsic properties of the operator. All features are binary for simplicity.

User Behavior Features

The first set of features are based on the user history. 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. Features based on relative operator popularity include: 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, including a feature template for the split ranker.
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 (the most recent.) Features 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)
Person Name Similarity - A query may contain part of a person's name that also appears as an operator. For example, the operator "from:bill smith" may be relevant to the query "Bill." We check for a match between a term in the query and person operators. (1 feature)

Learning Algorithm

We learned a linear ranking function parameterized by weight vector w using a simple one-best perceptron ranker, where an

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 2: Mean reciprocal rank (MRR) of the rankers on each dataset. The learning ranker performs the best and all rankers improve over the Max Count heuristic.

operator x is ranked by its score $w \cdot x$ [Crammer and Singer, 2003; Harrington, 2003].[4] The perceptron is an online mistake driven algorithm; updates occur when the top operator is incorrect. An update increases the correct operator's score and decreases the top operator's score: $w^{i+1} = w^i + y - \hat{y}$, where w^i is the weight vector after the ith training example, y is the correct operator and $\hat{y}$ is the top predicted operator. We train the ranker with 10 passes over the training data.

6 Evaluation

We evaluated our rankers on both datasets by creating 10 train/test splits for cross validation. Each split's training set was used to build a user search history and train the machine learning ranker. We investigated creating separate rankers for each view type (inbox, search) but found no change in performance, so all users and views were grouped together. We use two common information retrieval evaluation metrics: MRR[5] and accuracy at n. [6] For our application, this latter measure is particularly relevant since it indicates how often a ranker displaying n results would have shown the correct operator.

MRR results are shown in table 2 and for accuracy at n in figure 3. *Displayed Order* and *Max Count* perform with a MRR of 0.31 and 0.19 respectively; both reflect a bias compared with a MRR of 0.10 for *Random*. *Displayed Order* results are consistent with click bias (figure 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 and do much better on the smaller dataset since every user has some history. The *Machine Learning* system does best overall, with a MRR of 10 points better on *All* and 8 points better on *Min10* than the best popularity ranker. Its MRR is twice that of *Displayed Order* on *Min10*. The learning ranker has the best 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. The *Popularity* and *Machine Learning* rankers could likely support a production system.

6.1 Feature Analysis

We analyzed feature effectiveness by running the learning ranker with each feature type removed. Removing popularity features reduces performance to a third of the full MRR score (0.59 to 0.17 for *All*.) Relative popularity also has a significant impact for the *Min10* data (0.68 to 0.62,) 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 features for result order and operator value made small differences but the rest of the features made almost no difference to performance.

7 User Survey

We polled 85 experiment participants using an online survey. Complete details appear in [Dredze, 2009]. Respondents were power email users, with many years of Gmail experience and high mail volumes. Users liked the idea of having search operators suggested (74% respondents assigned favorable rating to concept.) Additionally, 77% said they would use a tool if it produced high quality suggestions. Nearly all who used the system mentioned search (32 respondents) and 91% of respondents read a message after selecting an operator, indicating that the primary purpose was to find messages. 59% of respondents agreed with the statement "Suggesting operators for search results helps me find what I am searching for." Several users commented that they looked at the operators "when I didn't find what I was looking for" or "before... changing the query" and that they "really helped me narrow down the search." Clearly, operators helped narrow searches.

The majority (54%) agreed that search operators helped inbox management. Many respondents indicated they labeled (20%) or archived (33%) a message after using the operators. Inbox management comments focused on processing groups of messages, selecting operators when "triaging a large number of unread emails," "my inbox was most full and I needed to drill down quickly" and "when I felt overwhelmed." Users found that highlighting threads that matched an operator on mouseover events 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." Several users described how the tools supported triage, such as using "the tool in the morning to process the overnight mail." 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 results [Kaki, 2005].

8 Conclusion

We have presented a system that suggests email search operators for dynamic mail view filters, useful for both narrowing searches and finding groups of messages in the inbox. Empirical results on user interaction data show that our rankers

[4] We found this simple learning algorithm best because of weak user feedback. Aggressive updates, such as max-margin and k-best ranking, work best with little label noise. Initial experiments with k-best ranking supported this decision.

[5] The mean reciprocal rank at which the correct example appears. For N instances, where r_i is the rank of the correct operator in the ith instance: *MRR*= $\frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i}$.

[6] Accuracy at n credits a ranker when the correct operator appears in the top n results.

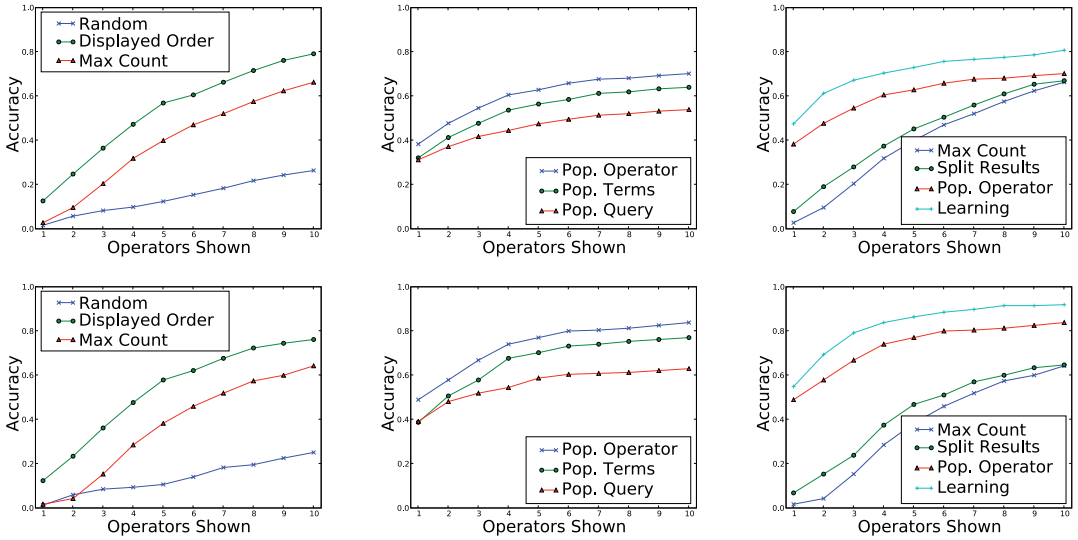

Figure 3: Accuracy at N for the *All* (top) and *Min10* (bottom) datasets. The x-axis shows the increasing n number of positions considered and the y-axis the accuracy of finding the correct operator in one of the n positions.

effectively select useful search operators, showing the user a few useful operators for currently displayed messages.

9 Acknowledgments

The authors thank Yang Li, Fernando Pereira and Arun Swami for suggestions, the Gmail team for implementation assistance and users who participated in our experiment.

References

[Crammer and Singer, 2003] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. In *Journal of Machine Learning Research (JMLR)*, 2003.

[Cutrell *et al.*, 2006] Edward Cutrell, Daniel C. Robbins, Susan T. Dumais, and Raman Sarin. Fast, flexible filtering with phlat - personal search and organization made easy. In *Computer-Human Interaction (CHI)*, 2006.

[Dredze, 2009] Mark Dredze. *Intelligent Email: Aiding Users with AI*. PhD thesis, University of Pennsylvania, 2009.

[Dumais *et al.*, 2003] 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.

[Fisher *et al.*, 2006] 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.

[Harrington, 2003] Edward F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *International Conference on Machine Learning (ICML)*, 2003.

[Hearst, 2006] Marti Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4), 2006.

[Joachims *et al.*, 2005] 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)*, 2005.

[Kaki, 2005] Mika Kaki. Findex: Search result categories help users when document rankings fail. In *Computer-Human Interaction (CHI)*, 2005.

[Neustaedter *et al.*, 2005] Carman Neustaedter, A.J. 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)*, 2005.

[Oren *et al.*, 2006] Eyal Oren, Renaud Delbru, and Stefan Decker. Extending faceted navigation for rdf data. In *International Semantic Web Conference (ISWC)*, 2006.

[Radlinski and Joachims, 2005] Filip Radlinski and Thorsten Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *American National Conference on Artificial Intelligence (AAAI)*, 2005.

[Sanderson and Dumais, 2007] Mark Sanderson and Susan Dumais. Examining repetition in user search behavior. In *European Conference on Information Retrieval (ECIR)*, 2007.

[Smith *et al.*, 2006] 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, 2006.