EXPLORING PRIVACY AND PERSONALIZATION IN INFORMATION RETRIEVAL APPLICATIONS

A Dissertation Presented

by

HENRY A. FEILD

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2013

School of Computer Science

EXPLORING PRIVACY AND PERSONALIZATION IN INFORMATION RETRIEVAL APPLICATIONS

A Dissertation Presented

by

HENRY A. FEILD

Approved as to style and content by:

James Allan, Chair

W. Bruce Croft, Member

Michael Lavine, Member

Gerome Miklau, Member

Lori A. Clarke, Chair
School of Computer Science

To my wife, Jacq.

ABSTRACT

EXPLORING PRIVACY AND PERSONALIZATION IN INFORMATION RETRIEVAL APPLICATIONS

SEPTEMBER 2013

HENRY A. FEILD

B.Sc., LOYOLA COLLEGE IN MARYLAND

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor James Allan

A growing number of information retrieval applications rely on search behavior aggregated over many users. If aggregated data such as search query reformulations is not handled properly, it can allow users to be identified and their privacy compromised. Besides leveraging aggregate data, it is also common for applications to make use of user-specific behavior in order to provide a personalized experience for users. Unlike aggregate data, privacy is not an issue in individual personalization since users are the only consumers of their own data.

The goal of this work is to explore the effects of personalization and privacy preservation methods on three information retrieval applications, namely search task identification, task-aware query recommendation, and searcher frustration detection. We pursue this goal by first introducing a novel framework called CrowdLogging for logging and aggregating data privately over a distributed set of users. We then describe several privacy mechanisms for sanitizing global data, including one novel mechanism based on differential privacy. We present a template for describing how local user data and global aggregate data are collected, processed, and used within an application, and apply this template to our three applications.

We find that sanitizing feature vectors aggregated across users has a low impact on performance for classification applications (search task identification and searcher frustration detection). However, sanitizing free-text query reformulations is extremely detrimental to performance for the query recommendation application we consider. Personalization is useful to some degree in all the applications we explore when integrated with global information, achieving gains for search task identification, task-aware query recommendation, and searcher frustration detection.

Finally we introduce an open source system called CrowdLogger that implements the CrowdLogging framework and also serves as a platform for conducting in-situ user studies of search behavior, prototyping and evaluating information retrieval applications, and collecting labeled data.

ACKNOWLEDGMENTS

I would like to take a moment to thank all of the people who encouraged me on my path to completing this work and earning a Ph.D. First and foremost, I thank my beautiful wife, Jacqueline Feild. Six years ago, we embarked together on our journey through graduate school. In the process, we married and have experienced all the aspects of what it means to be graduate students together. Her love and support have been crucial.

I also thank my family. I am not sure that they exactly understood what it meant when I told them I was busy "researching", but they supported me anyway. They allowed me to watch movies while working on my laptop and to otherwise spend most of my time with them half working and half paying attention to them. All the while, they were happy just to have me around, and I am very grateful for that.

Over the last six years, James Allan has served as my advisor. James is kind, funny, and reasonable. He allowed me an immense amount of flexibility to work on what I wanted and he was never overly demanding. He served as a source of sage advice, both academically and in terms of life. I am extremely lucky to have ended up in his academic care and am very thankful for everything he has done for me.

My earliest academic mentors were David Binkley and Dawn Lawrie, my undergraduate research advisors. They guided me and taught me the basic tenants of research. Between watching them teach and perform research, they are the reason I decided to go to graduate school and ultimately to teaching a a liberal arts college.

I engaged in a number of internships and teaching experiences outside of UMass during graduate school. My mentors encouraged me and helped me network well beyond what I could have done alone. I thank Ellen Voorhees from the National Institute of Standards and Technology for introducing me to the core concepts of information retrieval during my summer internship just before entering graduate school. I thank Rosie Jones, Emre Velipasaoglu, and Elizabeth Churchill, who mentored me during my time at Yahoo! Labs. I thank Ryen White, who served as my mentor while interning at Microsoft Research, as well as the rest of the CLUES group that I worked with. I lab instructed for classes taught by Audrey Lee-St. John and Lisa Ballesteros at Mt. Holyoke College, and both provided me with an incredible amount of guidance and ultimately gave me the liberal arts teaching experience necessary to help me make my final decision to teach at such a school. I owe them a great deal of gratitude.

Several people were generous enough to give me advice, guidance, and read over my application materials as I applied for faculty positions. Specifically, I would like to thank James Allan, Dawn Lawrie, David Binkely, Jerod Weinman, Lisa Ballesteros, Audrey Lee-St. John, Tim Wood, and Megan Olson. In addition, James Allan, Lisa Ballesteros, Audrey Lee-St. John, Emre Velipasaoglu, and Ryen White were kind enough to provide letters of recommendation for me. I would like to express my extreme gratitude especially to Audrey, who helped me every step of the way through the process. I'm not sure my faculty search would have been successful without her input.

Dealing with any large school can be difficult. Luckily for me, the School of Computer Science at UMass Amherst has Leeanne Leclerc. Leeanne makes sure that everyone gets signed up for the right classes, that we have health insurance, and that we get paid. She is the graduate students' primary link to the administrative staff and, in a very real sense, the faculty. In short, she is incredible and I thank her for everything she has done for me these past six years. On a similar note, David Fisher, Jean Joyce, Kate Moruzzi, Dan Parker, and Glenn Stowell have made working in the Center for Intelligent Information retrieval a terrific experience.

Graduate students are a confused lot, and they need each other in order to make sense of all the stages of the graduate school experience. Some of the graduate students who I have relied on most and would like to think are: Elif Aktolga, Niranjan Balasubrimanian, Michael Bendersky, TJ Brunette, Marc Cartright, Van Dang, Jeff Dalton, Sam Houston, Tamsin Maxwell, Megan Olson, Dirk Ruiken, Laura Sevilla, and Tim Wood. Niranjan, Marc, and Sam in particular have provided me with extremely sound and rational advice, usually over a pint. Other former graduate students encouraged me after they graduated. These include Ben Carterette, Matt Lease, and Mark Smucker—many thanks to all of them. I also thank Laura Dietz, one of the postdocs in the lab that handed out advice everyday over lunch.

Finally, I thank my committee: James Allan, Bruce Croft, Michael Lavine, and Gerome Miklau. They gave me incredibly useful feedback and helped me place the finishing touches on this document.

I have had a great over the last six years. It was tiring and trying at times, yes, but I would do it all over again if given the chance. Graduate school is a unique experience, as near as I can tell, and the number of amazing people I have encountered is beyond anything I had ever imagined.

This work was supported in part by the Center for Intelligent Information Retrieval, in part by UMass NEAGAP fellowship, in part by the Defense Advanced Research Projects Agency (DARPA) under contract number HR0011-06-C-0023, in part by NSF CLUE IIS-0844226, in part by NSF grant #IIS-0910884, in part by NSF grant #IIS-11217281, and in part by UpToDate. Any opinions, findings and conclusions or recommendations expressed in this material are the author's and do not necessarily reflect those of the sponsor.

CONTENTS

Page

CHAPTER

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

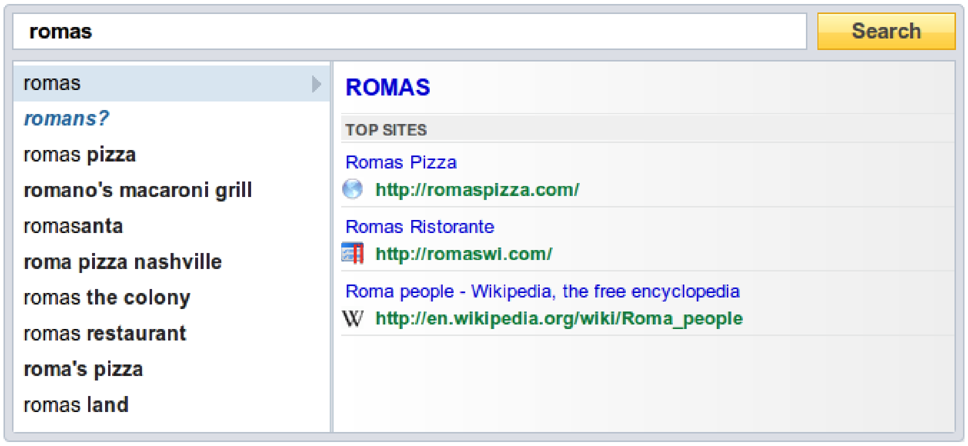
CHAPTER 1
INTRODUCTION

This thesis investigates how privacy and personalization affect three information retrieval (IR) applications: *search task identification*, *task-aware query recommendation*, and *searcher frustration detection*. In pursuing this goal, we outline a framework for collecting and mining data privately, several analyses of the effects of privacy, and a general procedure for combining global data aggregated from many users with local data pertaining to a single individual to provide personalization. We end with a description of an open source system called CrowdLogger that implements the CrowdLogging framework and serves as a platform for conducting in-situ user studies.

1.1 Motivation

Most IR applications that web searchers use on a day-to-day basis are not highly personalized. For example, if a user enters the search "romas" into a commercial search engine, most results that are returned are related to restaurants. This is also true of the suggested query reformulations, as illustrated in Figure 1.1. The one bit of personalization that is widely attempted is the incorporation of location information; with an Amherst, Massachusetts IP address, most commercial search engines show a map at the top of the search page with locations of "Roma's Pizza" near Amherst, MA. If the majority of users who enter the query "romas" are looking for "Roma's Pizza", this is an effective use of personalization. However, consider the scenario where, just prior to this query, the user submitted the following queries:

1. tomato varieties



romas
romans?
romas **pizza**
romano's macaroni grill
romas**anta**
roma pizza nashville
romas **the colony**
romas **restaurant**
roma's pizza
romas **land**

ROMAS

TOP SITES

Romas Pizza
http://romaspizza.com/

Romas Ristorante
http://romaswi.com/

Roma people - Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Roma_people

Figure 1.1. An example of the query suggestions offered by the Yahoo! search engine for the query "romas".

2. cherry tomatoes

3. grape tomatoes

The user's search history clearly indicates that the most likely intended target of "romas" is "roma tomatoes". If this information were taken into account, the query suggestions and results could be re-ranked to reflect this intention. This example illustrates *personalization*. Because the results are tailored to this particular user's context, the expected outcome is higher relevance relative to the user, less time to success, and a better user experience.

Our focus is on IR applications that 1) can benefit from personalization and 2) rely on *global data* collected from a large number of individuals. We already saw how query recommendation could benefit from personalization by taking into account a user's previous searches. State-of-the-art query recommendation systems also rely on global data, namely common query reformulations entered by other users. There are a number of reasons why global data may be required by an IR application. For example, when a user's data is too sparse or when an element of popularity that cannot be captured by an individual's data is required.

The reliance on global data raises an import issue that is a growing concern on the web: privacy. Collecting data without considering user privacy can have devastating consequences. For example, identity theft, public humiliation, and loss of user trust. Take, for example, the query suggestion algorithm discussed above. If we blindly collect *all* query reformulations from contributing users, then some very sensitive query suggestions may be presented to another user. Suppose user X submits the query *stolen credit card* followed by another query consisting of his credit card number—*222333444888*—perhaps in an effort to see if his credit card number has appeared somewhere on the web. This query reformulation is collected and stored with the global data. Now another user, Y, comes along and enters *stolen credit card.* Unless some care is taken, it is reasonable to expect that *222333444888* may come up as a suggestion, especially if *stolen credit card* is not a popular query. Since this is clearly a credit card number, Y has information that could be used to take advantage of X and possibly even identify X.[1] This example demonstrates that preserving user privacy must be a primary consideration in collecting and using global data.

The goal of this work is to explore two important research questions: first, how can useful global data be collected but *sanitized* to maintain user privacy, and second, how can this cleaned-up global data and rich local user data be combined to provide personalization. We look specifically at three IR applications: *search task identification*, *task-aware query recommendation*, and *searcher frustration detection.* For each, we analyze the effects of certain types of personalization and privacy-preservation mechanisms on their performance.

To address these questions, we divide this work as follows. Chapter 3 details CrowdLogging, a framework for logging, mining, and aggregating search activity over a distributed network of users in a privacy-preserving manner. We also explore some

[1] Yes, it is true that a credit card number alone would not be enough information to use the card, but it illustrates the point that sensitive information can be revealed.

of the privacy mechanisms that can be used with CrowdLogging to sanitize search logs. In Chapter 4, we describe a general framework for combining local user data with global aggregate data to produce useful privacy preserving, personalized IR applications. In Chapters 5, 6, and 7, we introduce personalized models for each of the IR applications and analyze the effects of both privacy and personalization on performance. In Chapter 8, we describe a web browser extension called CrowdLogger, which implements the CrowdLogging framework and provides many functionalities for researchers to release prototypes of IR applications or conduct in-situ user studies. Finally, we wrap up with conclusions and some final thoughts in Chapter 9.

1.2 Contributions

The following is a summary the contributions of this thesis.

1. **CrowdLogging** – We describe a new distributed framework for logging and mining data on users' computers and a mechanism to upload and aggregate user data in a sanitized form in Chapter 3.

2. **$\mathbf{DP_u}$** – In Chapter 3.4, we introduce a novel extension to previous work, creating a differentially private mechanism, DP_u, that considers the number of users that contribute a search artifact rather than the number of instances of that artifact. We use DP_u as a differentially private parallel for another privacy mechanism we formalize, called FT_u. We demonstrate that DP_u provides coverage similar to the privacy mechanism from which it was derived, DP_a.

3. **Empirical examples of privacy leakage** – In Chapter 3.4, we describe three attacks that demonstrate the vulnerability of two privacy policies that we formalize, FT_a and FT_u. One attack allows an attacker to attribute released information to "power users", another allows an attacker to infer information that is not released, and a third allows an attacker with knowledge that a piece

of data from a given user was included during data collection to infer additional data from that user. We demonstrate the practicality of the second attack using query sequences extracted from the 2006 AOL query logs, showing that a small number of infrequent—and therefore unreleased—query triples can be inferred using released query and query pair counts.

4. **Artifact coverage comparison** – One class of privacy mechanisms analyzed in this thesis, frequency thresholding, releases considerably more information than differentially private mechanisms using parameter settings typical in the literature. As differentially private mechanisms have many desirable theoretical properties, we consider the settings required to achieve similar coverage to that produced with frequency thresholding, specifically for releasing query artifacts from the AOL search log. We find that the settings required to do so are generally unreasonable, and we conclude that differentially private mechanisms should not be used to obtain the same level of coverage as frequency thresholding using the AOL search log.

5. **Personalized search task identification** – In Chapter 5, we demonstrate the variability in individuals' perceptions of what constitutes a search task, finding a Fleiss' Kappa as low a 0.53 among six annotators' labels across ten user histories. This is the first such analysis in the search task identification literature that we know of. We gather annotations for over 503 user histories extracted from the 2006 AOL search log, labeled by ten annotators—38 times as many user histories as used by the current state of the art research. With this data, we introduce several models for providing personalization, but find they perform similarly to using a random forest classifier trained on global data, all achieving between 94% and 95% macro accuracy across users. Our experiments show the random forest classifier significantly out-performs the current state of the

art model. We further demonstrate that sanitization has an overall mild effect on performance, and can even improve performance under certain conditions, namely when the FT_u mechanism is used with $k = 100$.

6. **Task-aware query recommendation** – We introduce in Chapter 6 a novel task-aware query recommendation model, which combines query recommendations for each query within a task, giving more weight to queries based on either temporal distance or same-task likelihood. It relies on personalized search task identification, as described above. We find that leveraging on-task search context provides a large boost in MRR for many evaluation queries—more than 25% on average. Privacy has a significant impact on recommendation performance, rendering the quality so low as to be impractical in a real system in most cases we consider.

7. **Personalized frustration detection** – In Chapter 7 we explore the effects of personalization on frustration detection and show that personalization can provide substantial performance improvements–as much as 9% in $F_{0.5}$. We also demonstrate that with a simulated user base of 100,000 users, performance of global models is not affected by sanitization.

8. **CrowdLogger** – Using the CrowdLogging framework as a foundation, we implement a browser extension for Firefox and Google Chrome called CrowdLogger (Chapter 8), as a platform with which to perform in-situ studies of user search behavior, evaluate IR applications, and provide research prototypes to interested parties. In addition to implementing the key portions of CrowdLogging, CrowdLogger allows researcher-defined JavaScript/HTML modules to be remotely loaded. CrowdLogger exposes an API for: accessing a user's real-time or past browsing behavior; uploading data to a server privately, anonymously, or in the clear (as approved by the user); interacting with the user via HTML

and JavaScript; accessing a remote server for computational purposes; and saving data on the user's computer. We describe two studies and one prototype that have been implemented with CrowdLogger.

CHAPTER 2
BACKGROUND, DEFINITIONS, AND RELATED WORK

This dissertation touches a number of areas, including privacy, personalization, and several IR applications. We describe these below and how our work fits in. First, we present some terminology that will be used throughout this work.

2.1 Terms and definitions

Moving forward, it will be helpful to understand the terms we make frequent use of. This section tries to put those terms into context; Appendix A provides a summary glossary for convenience.

Search logs, or *query logs* as they are sometimes called, are databases containing information about user search activity. A typical log might contain a set of entries consisting of an identifier, a time stamp, an event descriptor, and pertinent information about the event. Common identifiers are IP addresses, cookie identifiers, or account numbers. Example event descriptions include *query* or *click*. Pertinent information about the event might include the query text or a clicked URL. When we assume that all identical identifiers correspond to a single individual, we refer to a collection of search log entries sharing the same identifier as a *user search log* or *user log* for short. An example user log is shown in Figure 2.1.

Search logs can be used to build models of user behavior, which can then be used to influence search algorithms. For example, learning-to-rank algorithms extract or *mine* ⟨query, search result click⟩ pairs from search logs to learn a ranking function (Joachims, 2002). Many query suggestion techniques, such as the one used

ID	**Time**	**Event Type**	**Event Info.**
1019	11/15/11 09:02:03	query	running shoes
1019	11/15/11 09:02:10	click	www.nike.com
1019	11/15/11 09:03:15	query	asics

Figure 2.1. An example user log where two queries have been entered roughly a minute apart and one URL has been clicked for the first query.

by Boldi et al. (2008), are based on query reformulations pairs mined from search logs, e.g., *running shoes* → *asics* in Figure 2.1. The pieces of data being mined—e.g., query-click pairs and query reformulations—are called *search log artifacts*, or *artifacts* for short. In this work, we refer to a set of artifacts mined from one or more user search logs as a *crowd log* and a model built from a crowd log as a *global model.*

As explained in Chapter 1, when crowd logs are created without concern for privacy, sensitive information may be exposed. When privacy is taken into account and a *privacy policy* is instated to reduce the chance of revealing sensitive information in a crowd log, we consider the resulting crowd log and subsequent global models *sanitized.* An example privacy policy is to include artifacts in a crowd log only if they were mined from at least k different user logs. An artifact present in a crowd log is said to be *sufficiently supported* by the privacy policy.

We can process user logs to build user-specific models, which we call *local models.* We do not sanitize local models under the assumption that users interacting with their own sensitive information do not constitute breaches of privacy. (We do not consider the case where multiple individuals access the same account, though this may be a common situation with, e.g., shared computers in homes, libraries, or Internet cafés.)

All of the IR applications we consider use a notion of *search tasks.* We note that our use of search task in this thesis differs from others in the fields of information retrieval and information seeking, which consider a task to consist of several facets such as its goal, source, and the process by which it is carried out using a variety of digital and physical media. Li (2009) provides an overview of many of these other definitions. For our purposes, a search task is a set of one or more *searches* entered by a user pertaining to the same information need. A search consists of a query submission followed by zero or more non-query interactions, such as clicks and page visits. Tasks are different from *search sessions*, which consist of a sequence of one or more searches within either a specified time period or when a period of inactivity is encountered. Sessions may consist of multiple tasks, and tasks may span multiple sessions. We define *search task identification* to be the problem of grouping a set of user searches into search tasks.

In the field of IR, *personalization* is the act of incorporating information about a user into the processing of an IR application. For example, boosting the rank of search results that are more similar to web sites a user has visited in the past. Another common practice is the use of a user's location, often inferred by IP address or GPS coordinates. We concentrate on local-only personalization that can be performed in isolation on an individual's computer. We consider two different types of personalization: task-aware and supervised learning from user-provided annotations. Note that, in general, personalization can also involve a group of users; we do not consider group-level personalization in this thesis.

One of the IR applications we consider is detecting *searcher frustration.* We define searcher frustration as the self-reported level of frustration a user experiences while engaged in a search.

You can find a glossary of these terms in Appendix A.

2.2 Privacy in Information Retrieval

There is an increasing interest in developing methods for preserving search user privacy while still gleaning information to improve IR applications. At one extreme, no privacy is maintained and search log collectors acquire unfettered access to users' search behavior. This is good for the collector and potentially users as well, who will benefit from more finely tunned IR algorithms. However, users lose all control over how their information will be used and shared. At the other extreme, privacy is preserved, but no useful data is passed on to the collector, reducing the amount of data collectors have to make informed algorithm enhancements.

In this section, we will first take a look at the current state of commercial Web search, since it touches most of our daily lives. Then we discuss related work pertaining to several classes of sanitization mechanisms and corresponding vulnerabilities. For the first part of the related work, we use an informal definition of privacy, namely that a sanitization mechanism preserves privacy if, given the output of the mechanism, an analyst cannot make a reliable guess as to who a query belongs to. A little later on, we will introduce a more formal definition used in much of the most recent research on the topic.

Some of largest US based search engine companies (Google, Yahoo, Microsoft, AOL, and Ask.com) have privacy policies that do little to anonymize search data, keeping IP address and browser cookie identifiers associated with searches for as long as 18 months (Schwartz & Cooper, 2007). For users that do not want their queries tracked, there are some server-side options. For example, some less popular search engines state in their privacy policies that they do not log user interactions, such as Duck Duck Go.[1] Another approach is to use a decentralized search engine, as is done with the peer-to-peer service YaCy.[2] On the client-side, the TrackMeNot project (Howe & Nissenbaum, 2009) is useful for web searchers that want to ensure their data is not logged, or at least not in a meaningful way, by any search engine. The project, offered as a Firefox web browser extension, obfuscates user search queries by sending many other unrelated searches to the target search engine. While the user's query can be logged by the search engine, it will be lost in the noise of the

[1] `http://duckduckgo.com/`

[2] `http://yacy.net`

other searches, thereby protecting the user's intent to some degree. In the current commercial landscape, the consumer's choice is to either allow search engines to collect all pertinent searching behavior over some period of time and reap the benefits of personalization and high quality results, or not allow search engines access to any useful information, sacrificing both personalization and quality of service.

Xiong and Agichtein (2007) enumerated two key privacy concerns for publishing search logs: how queries are related and how sensitive information within a query (e.g., a social security number) is handled. generalizing their categorization slightly, we break the related working into two overarching sanitization classes: 1) techniques that primarily focus on how log entries are related, i.e., how entry identifiers are anonymized; and 2) techniques that also consider the contents of the data contained within each entry. For additional overviews of search log privacy techniques, consult Xiong and Agichtein (2007) and Cooper (2008).

In an unsanitized search log, entries are primarily related by their identifier, where each user has a unique identifier. In the event that identifiers contain identifiable information such as a user name, IP address, or cookie identifier, a simple sanitization approach is to anonymize that information. Anonymizing user identifiers involves mapping each to a unique string that does not contain private information. AOL publicly released a three month search log in 2006 in which users were given anonimized identifiers (some search data was also removed). However, at least one user was identified by a New York Times reporter soon after the release (Barbaro et al., 2006). The reason users could be picked out did not have to do with the content of the identifier, but rather the fact that multiple searches and clicks were linked together using the identifier. It was the text of the queries themselves that provided more and more evidence about who the underlying user was, finally resulting in a positive identification.

Microsoft released a search log to researchers the same year, but only provided anonymized identifiers for user search sessions, preventing queries from the same user but different sessions to be explicitly linked (Microsoft, 2006). By mapping users to multiple pseudo users, it is more difficult to make the same links that the AOL data allowed. First, queries issued on two different days are not directly connected (provided the user was inactive for part of the intervening time period). Second, sessions generally consist of a small number of queries and are therefore less likely to cover a diverse set of topics. When these two points are taken together, it becomes more difficult to gather sufficient evidence to identify the individuals behind the queries. Not impossible, however: Jones et al. (2007) demonstrated that even when breaking users' data into day-long chunks, their gender, age, and zip code can be established with "reasonable accuracy" using classifiers, contributing towards identifying individuals and their search histories.

Another sanitization technique works in the opposite manner: it maps multiple user identifiers to a single pseudo identifier. In 2007, Google released a statement asserting they would, after 18 months, anonymize the IP address associated with each search event by removing the last octet of the address (Fleischer, 2007). By bundling multiple users together, it is more difficult to identify individuals due to the noise of all the other users' data contained in the bundle. However, Jones et al. (2008) published several attacks on identifier-based bundling. On a sample of logs, they were able to identify with 71% precision which queries belong to the same user. By leveraging users' tendencies to search for their own names,[3] the presence of users that own the majority of actions within a bundle, and the ability to infer strings of related searches within a bundle, an attacker can make educated guesses about who a query belongs to.

[3] Roughly 30% of users searched for themselves over a 70 day period.

To overcome the vulnerabilities of identifier-only sanitization, recent works have considered additional log information, such as query terms. One technique relies on hashing individual query terms or whole queries (Cooper, 2008). This allows data miners to analyze query statistics without knowing the contents. However, Kumar et al. (2007) showed that anonimization by means of hashing query terms can be partially breached by mapping the hashes to terms or queries in another, unanonimized query log—for example, the AOL query log.

Adar (2007) presented two methods for sanitizing a search log in the interest of preserving privacy. First, only queries that are issued by at least t users are kept. This is similar to Sweeney (2002)'s k-anonymity work for databases, which assigns a subset of database columns to be a quasi-identifier and the remaining columns to be sensitive attributes. A row in the database is only released if the quasi-identifier is shared by at least $k-1$ other rows. Under Adar's model, a query serves as both the quasi-identifier and the sensitive attribute. Adar's second technique anonymizes the identity of users who enter a series of queries that may reveal who they are when taken together, by clustering syntactically related queries from a user's session and releasing each cluster under its own identifier. This is similar to the identifier-based method of mapping a single user identifier to many pseudo identifiers, though the clustering may reduce the chances of linking data from multiple pseudo identifiers.

Hong et al. (2009) defined k^{δ}-anonymity, where for every user whose data is listed in the sanitized search log, there are $k-1$ other users that are δ-similar to the user in terms of their data. Their method involves grouping similar users and adding or suppressing data to make user groups appear more similar. One downside to this approach is that some data is permanently discarded and artificial data is added. It is unclear how different types of applications are affected by this anonymization process.

A substantial disadvantage with both k^δ-anonymity and t-anonymity discussed above is their lack of theoretical backing. While they make intuitive sense, they do not provide any formal guarantees of privacy. It is difficult to estimate, for a given dataset, what the chances are that a user will be identified. Dwork (2006) introduced a provably private approach called *differential privacy*, targeted to databases in general. Differential privacy applies to randomized algorithms. Such an algorithm is ϵ-differentially private if for all pairs of input data sets that differ by one item, the outputs, e.g., a histogram of queries, only differ within an exponential factor of ϵ. Assuming a user is an item, what this means is that, given the output of an ϵ-differentially private mechanisms, an analyst should not be able to tell with any significant confidence if a given user's data was or was not present in the input data set, since the outputs would only differ by an exponential factor of ϵ.

Kodeswaran and Viegas (2009) introduced a differentially private framework that allows researchers to access attributes of a search log, such as a count of the number of users that enter searches at a certain hour of the day. However, Götz et al. (2011) showed that differential privacy cannot be applied directly to obtain useful collections of search log artifacts, such as queries or query reformulations. Several relaxations can, though, and both Korolova et al. (2009) and Götz et al. (2011) described relaxed differentially private mechanisms for doing so.

Korolova et al. introduced a relaxed differentially private algorithm to release a query click graph from a search log. Their work differs from previous work in this field in two important ways: (1) the output of their sanitization is a query click graph rather than a search log, and (2) they prove theoretical bounds on the effectiveness of their privacy-protecting scheme.

Götz et al. developed an algorithm called ZEALOUS to release histograms of arbitrary artifacts of a search log, e.g., queries, query-URL pairs, or query reformulations. They demonstrated that under one configuration of input parameters, the algorithm is similar to Korolova et al., but allows users to contribute d distinct artifacts where Korolova et al. allow d non-distinct artifacts. Götz et al. showed that under a different configuration of parameters, ZEALOUS provides a more conservative relaxation of differential privacy and thus yields stronger privacy guarantees.

We develop a framework that keeps users' data on their own computers and remotely mines and then uploads data privately to a centralized location. We also explore two new privacy mechanisms for this framework that stem from previous work. One is similar to t-anonymity, but with two key differences: 1) search log artifacts in general are considered, rather than explicitly queries (though those can be artifacts), and 2) t-anonymous artifacts are not linked to each other, preventing explicit linking. We consider two variations of this mechanism, one where t corresponds to the number of users that share an artifact within a search log (called user frequency thresholding) and another where t corresponds to the number of instances of an artifact in the log (called artifact frequency thresholding).

As the thresholding methods do not provide any guarantees about privacy loss, we also consider a variant of the Korolova et al. algorithm, that has potentially higher utility. Because the utility of data sanitized using differentially private methods is affected by the size of the search log, we further investigate how to best process growing search logs. For example, whether search logs should be sanitized each day or in week-long segments.

Finally, we explore the effects of our and existing algorithms on three specific IR applications with and without personalization.

2.3 Selected Information Retrieval Applications

In this section, we discuss related work surrounding three specific IR applications: search task identification, task-aware query recommendation, and searcher frustration

Query No.	Task ID	Query
1	1	florida resorts
2	1	daytona beach hotels
3	2	garden hoses
4	1	flights to florida

Figure 2.2. An example of a search session consisting of four queries that constitute two distinct search tasks.

detection. We cover work that focuses on personalization and privacy for each when applicable.

2.3.1 Search Task Identification

The goal of search task identification is to segment a sequence of searches into a set of tasks consisting of related searches. This problem consists of two steps: classifying pairs of searches as belonging to the same task and then clustering searches into tasks using the classification information. For example, consider the simple search session of four queries shown in Figure 2.2. Queries 1, 2, and 4 are all related to a trip to Florida while Query 3 is its own task. To identify these tasks, an algorithm first classifies each pair of searches, i.e., (“florida resorts”, “daytona hotels”), (“florida resorts”, “daytona beach hotels”), etc., as belonging to the same task or not. Once the predictions are made, the searches are clustered into coherent tasks, such as Task 1 and Task 2 shown in the example.

Jones and Klinkner (2008) introduced hierarchical search task identification. They trained logistic regression models to classify if pairs of queries belonged to the same task using two different levels of tasks: atomic tasks called *goals* and meta tasks called *missions*. Among the most influential features were lexical features, such as Levenshtein distance and the Jaccard Coefficient, as well as search log-based features, such as the probability of one of the queries in the pair being reformulated as the other in the logs. The authors did not consider the clustering step. Building on this work, Boldi et al. (2008) created two classifiers for identifying tasks within a session:

a logistic regression model similar to Jones and Klinkner's (2008) for use with query pairs that only occurred once in a training search log and a rule-based classifier for the other instances. They did not specify any details of the rule-based classifier. They cast the problem of clustering as an Asymmetric Traveling Salesman problem: their algorithm reorders searches in a session by finding the Hamaltonian path that minimizes the weight between consecutive queries in the new ordering (where the weight is defined as $-log(\text{classification score})$). Task breaks are then identified in the reordered sequence of searches where the classification score is below a threshold. Lucchese et al. (2011) built an ad-hoc classifier that integrates both the syntactic and semantic distance between two searches' query text. Among a number of different clustering algorithms they examined, they found the most effective was applying the weighted connected components algorithm over a fully connected graph in which the nodes are queries and the edges are the same-task classification scores. The graph is pruned of any edges below a threshold and each connected component in the pruned graph is considered a task. This method is equivalent to single-link clustering.

We build on the logistic regression classifiers used by Jones and Klinkner (2008) and Boldi et al. (2008) in addition to the semantic features and task clustering technique used by Lucchese et al. (2011). There are three key differences between our work and previous work. First, we do not assume a cross-user definition of a search task; rather, each user may have a slightly different notion of what constitutes a search task. Therefore, we explore augmenting globally learned same-task models with personalized, locally-trained models. Second, we consider the effects of privacy on globally trained same-task models. Third, we consider several datasets, one of which consists of full web search histories from 503 users hand-labeled by 10 annotators. We know of no work has thus far explored personalized search task identification, nor the performance impact of sanitized training data.

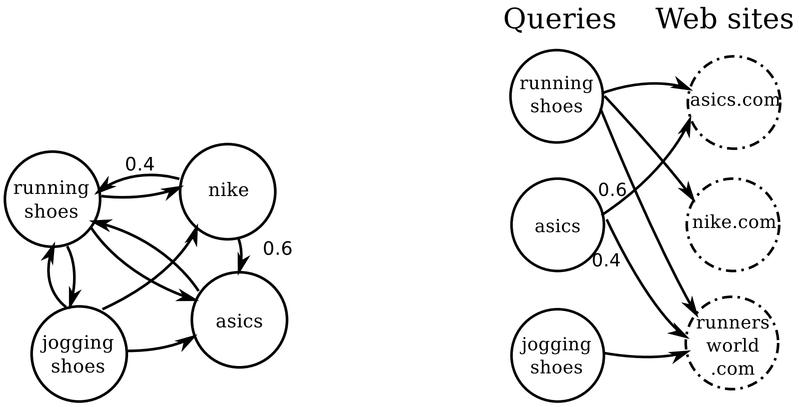


Figure 2.3. An example query flow graph (*left*) and query-click graph (*right*). The query flow graph consists of nodes that represent queries and direct edges that represent that the source query has been reformulated as target query in at least one search session. Weights are typically normalized per node across all outgoing edges (e.g., see the edges originating at the query "nike"). The query-click graph is bipartite; the solid-circle nodes on the left represent queries while the dotted-circle nodes on the right represent clicked web sites. Edges can only go from left to right and indicate that the source query was followed by a click on a link to the target website. As in a query flow graph, the edge weights are typically normalized per node over the outgoing edges (e.g., see the edges originating from the query "asics").

2.3.2 Task-aware Query Recommendation

We also explore task-aware query recommendation, in which previous searches submitted by a user as part of the same task as the current target query are used when generating recommendations. Task-aware recommendation is an instance of context-aware recommendation, which more generally leverages a user's search history when generating query recommendations.

Huang et al. (2003) introduced a search log-based query recommendation algorithm that extracts suggestions from search sessions in a query log that appear similar to the user's current session, thereby incorporating the surrounding search context. They found it outperformed methods that extract suggestions from retrieved documents in many aspects.

Filali et al. (2010) presented a probabilistic model for generating query rewrites based on an arbitrarily long user search history. Their model interpolates the same-task similarity of a rewrite candidate to the reference query with the average similarity of that candidate to all on-task queries from a user's history, weighted by each query's similarity to the reference query. They found that giving some weight to the history, but most weight to the reference query, did best on a set of manually and automatically labeled data.

Mei et al. (2008) presented a personalized query recommendation system that relies on query-click graphs—bipartite graphs where verticies on one side are queries, verticies on the other side are search results, and edges between indicate a search result was clicked for the corresponding query (see the right side of Figure 2.3 for an example). Queries are suggested by performing random walks starting from the initial query. Personalization is performed by updating the graph with the user's query-click history.

Zhang and Nasraoui (2006) constructed a query-to-query graph where directed edges represent reformulations—not necessarily consecutive—that occurred within a search session. A given query reformulation from a search session contributes weight to the corresponding edge in the graph based on the distance of the two queries in the session (in terms of the number of other queries submitted in between) as well as the content similarity. Boldi et al. (2008; 2009; 2009) constructed query flow graphs, where nodes are queries and directed edges are created between pairs of queries that are observed as reformulations in a set of training logs (see the left side of Figure 2.3 for an example). A list of suggestions is generated by performing a random walk on the graph, starting at the query of interest. Search context is integrated by giving non-zero weights to the elements of the random walk initialization vector corresponding to previous queries entered by the user. They compare against a method that uses a query-click graph without contextual information and find that query flow graphs work better. Szpektor et al. (2011) adapted the query flow graph by integrating query templates, which consist of semantic place holders within queries. This approach is helpful in generating recommendations for queries not previously seen by the system, but match a known template. While not directly related to privacy, this technique may be helpful in counteracting the degradation effects of privacy preserving mechanisms. Bonchi et al. (2012) introduced the term-query graph, another adaptation of query flow graphs to improve coverage of unseen queries. Query flow graphs in their original form can only provide recommendations for queries that are contained within the graph since the random walk must start from somewhere. In a term-query graph, an extra layer is added: each distinct term gets a node with an outgoing edge pointed at each query in the query flow graph that contains that term. This layer is not used in the random walk, but rather serves as a kind of inverted index into the query flow graph. The term-query graph can therefore provide recommendations for any query that contains at least one term that exists in the term-query graph. To generate a query recommendation for a given target query, a random walk is performed for each term in the query, where the initialization vector consists of each node in the query flow graph in which the term occurs. The geometric mean of the results for each random walk is then taken to get the final recommendation list. This work did not consider personalization.

Cao et al. (2008) introduced a context-aware recommendation system that converts a series of user queries into concept sequences and builds a suffix tree of these from a large query log. To produce recommendations, a concept sequence is looked up in the suffix tree and the common next queries are given as suggestions. Cao et al. (2009) explored an efficient way to train a very large variable length Hidden Markov Model (vlHMM), which considers sequences of queries and clicks in order to produce query and URL recommendations as well as document re-ranking. The authors trained the vlHMM on a large commercial search log. He et al. (2009) introduced the mixture variable memory Markov model, which models sequences of user queries. They found that leveraging variable length sequences out performed pair-wise recommendation algorithms.

Liao et al. (2012) explored the effect of task-trails on three applications, including query recommendation. They compare two session-based, two task-based, and two single-query recommendation models and found they retrieve complementary sets of suggestions, though the task-based models provided the higher quality suggestions. To identify tasks, they used an SVM model using features similar to Jones and Jones and Klinkner (2008), and the weighted connected components clustering method described by Lucchese et al. (2011).

Götz et al. (2011) explored the effect of privacy mechanisms on query recommendation. They use the technique introduced by Jones et al. (2006), which is non-personalized, and evaluated the effect of sanitization by measuring ranking metrics (e.g., precision, recall, etc.) between suggestion lists generated using the sanitized data and the recommendations using the unsanitized data set as ground truth. This offers a notion of relative utility, but not of absolute utility.

Our work relies on the state-of-the-art query-term graph described by Bonchi et al. (2012) and we introduce a model that allows us to incorporate task context in a more controllable way than previous work, allowing us to more carefully understand the effects of task context.. In addition, we consider the effects of privacy on the term-query graph on absolute performance of query recommendation.

2.3.3 Searcher Frustration Detection

Recall that searcher frustration is defined as the self-reported level of frustration a user experiences while engaged in a search. Knowing that a searcher is frustrated provides useful feedback about an IR application and could potentially be used in adaptive interfaces or algorithms (Feild, Velipasaoglu, et al., 2010).

In a related area, Xie and Cool (2009) explored help-seeking situations that arise in searching digital libraries. They identified fifteen types of help-seeking situations that their 120 novice participants encountered. The authors created a model of the factors that contribute to these help-seeking situations from the user, task, system, and interaction aspects. In a study examining how children search the Internet, Druin et al. (2009) found that all of the twelve participants experienced frustration while searching. The authors pointed out that children make up one of the largest groups of Internet users, making frustration a major concern. In a similar study, Bilal and Kirby (2002) compared the searching behavior of graduate students and children on Yahooligans! They found that over 50% of graduate students and 43% of children were frustrated and confused during their searches. In addition, they found that while graduate students quickly recovered from breakdowns—where users were unable to find results for a keyword search—children did not. Kuhlthau (1991) found that frustration is an emotion commonly experienced during the exploration phase of a search process. She states that encountering inconsistent information from various sources can cause frustration and lead to search abandonment.

While frustration prediction has not been directly studied in the field of IR, *searcher satisfaction* has. Searcher satisfaction in search can have different meanings. We define searcher satisfaction as the fulfillment of a user's information need. While satisfaction and frustration are closely related, they are distinct: searchers can ultimately satisfy their information need but still be quite frustrated in the process (Ceaparu et al., 2004). In previous work, satisfaction has been examined at the task or session level. These satisfaction models only cover searcher satisfaction *after* a task has been completed, not *while* a task is in progress. As such, satisfaction models are useful for retrospective analysis and improvement, but not as a real-time predictor. In contrast, with a frustration model that is defined throughout a search, these real-time solutions are available.

In one web search study, Fox et al. (2005) found there exists an association between query log features and searcher satisfaction, with the most predictive features being click-through, the time spent on the search result page, and the manner in which a user ended a search. They also analyzed browsing patterns and found some more indicative of satisfaction than others, such as entering a query, clicking on one result, and then ending the task. Clicking four or more results was more indicative of dissatisfaction. Huffman and Hochster (2007) found a relatively strong correlation with session satisfaction using a linear model encompassing the relevance of the first three results returned for the first query in a search task, whether the information need was navigational, and the number of events in the session. In a similar study of search task success, Hassan et al. (2010) used a Markov model of search action sequences to predict success at the end of a task. The model outperformed a method using the discounted cumulative gain of the first query's result set, suggesting that a model of the interactions derivable from a query log is better than general relevance in modeling satisfaction.

Frustration and satisfaction modeling are instances of the more general concept of user behavior modeling. The features and approaches used to model different user behaviors are often interchangeable, and there are several different approaches to behavioral modeling in the literature. Huffman and Hochster (2007) predicted session satisfaction using a regression model incorporating the relevance of the top three results returned for the first query, the type of information need, and the number of actions in the session. Hassan et al. (2010) used a Markov model to predict task success and found that sequences of actions, as well as the time between the actions, are good predictors. Downey et al. (2007) created a Bayesian dependency network to predict the next user browsing action given the previous n actions, parameterized by a long list of user, session, query, result click, non-search action, and temporal features. They found that using an action history with more than just the immediately preceding action hurt performance. White and Dumais (2009) explored predicting when users would switch between search engines. Their goal was "not to optimize the model but rather to determine the predictive value of the query/session/user feature classes for the switch prediction challenge" (p. 94). They used a logistic regression model that encompassed query, session, and user level features. They found that using all three feature classes outperformed all other combinations of feature classes and did much better than the baseline for most recall levels.

Our work differs from previous work in three ways. First, we consider the prediction of searcher frustration, something that no previous work explored that we know of. Second, we consider a combination of globally learned and locally learned models, tailoring prediction to individual user's habits. Third, we consider the effects of privacy on both the task identification portion as well as the globally trained frustration detection models.

2.4 Analyses of Privacy and Personalization

Several works have looked at the interaction of privacy and personalization. Many of these works assume that a user's privacy concern is uploading personal information to a server so that it can be used to personalize the returned content. Under this assumption, Krause and Horvitz (2010) explored the privacy-utility trade off for personalized web search and found that little personal information is needed to achieve significant utility. Xu et al. (2007) examined user profiles—a hierarchical set of user interests on the web such as: *sports*→{*soccer*, *football*, *baseball*}—automatically extracted from their the pages they visit. Their framework provides two parameters that users can tune to limit the amount of information that is shared with the server. Zhu et al. (2010) explored anonymizing user profiles for personalized search by bundling users into groups and creating a profile of the group to use for personalization. Similarly, Zhou and Xu (2013) considered providing peer-group personalization—personalization tailored to a group of similar individuals rather than to a single individual—as a means to protect individual's privacy. As in the other cases, personalization was performed on the server-side, and the privacy aspect was about protecting the information shared with the personalization algorithm.

The key differences between these works and ours is that we assume that the privacy concern is with creating global models, not personalization. One reason for this assumption is that we only consider personalization for individuals, not group-level personalization. We further assume that personalization is performed client-side or, when a server is required for computational reasons, the server is trusted to not retain the private information.

2.5 Search logs

We use the 2006 AOL search log for most of the analyses in this thesis. The log contains search activity for over 617,000 AOL users collected between March and May 2006. The dataset was publicly released in August, 2006 and then retracted soon thereafter. However, the data was already copied to several mirror sites by that point. There has been debate over the ethics of using the AOL data for research, given it was retracted (Bar-Ilan, 2007).

The AOL search log is one of three large search logs we have access to. The other two logs are: a one month sample of MSN search sessions from 2006 and several years of data from a medical best-practices search engine. The primary reason we chose to use the AOL search log over other the other two is that it is the richest, most general purpose of the logs. The MSN search log is inappropriate for our analyses because search sessions are not linked. That is, hundreds of search sessions may belong to the same user, but we have no way to establish that information. The privacy mechanisms we study in this thesis rely on having knowledge of all of a user's search activity, and thus the MSN log does not allow us to properly analyze them.

The medical best-practices search engine has a few characteristics that make it an inappropriate data set for our analyses. First, many of the “users” of the search engine are medical institutions, such as hospitals, and the search activity is shared among many individuals. For example, some institutions provide terminals with access to the search engine that anyone can use. Since activity cannot be associated with an individual, our ability to analyze personalization is inhibited. Another reason we elected not to use the medical best-practices search engine logs at this juncture is that it is a niche search engine, and it is unclear whether our findings would be tied specifically to that niche and thus limiting their impact. We believe that analyzing general web search behavior, as is captured by the AOL search logs, allows our findings to be have greater impact.

The AOL search log is the only large-scale data set we have access to that allows us to analyze the effects of both privacy and personalization. We believe we are justified in using this controversial data set since our focus is on better understanding how search behavior can be captured in a privacy preserving manner, and not to identify or otherwise do harm to specific individuals in the data set.

CHAPTER 3

CROWDLOGGING

In this chapter, we describe a novel framework for privately and anonymously aggregating data across a distributed network of users, called CrowdLogging (Feild et al., 2011). We begin by motivating the need for such a framework and describe the framework's internals. We look at several examples of the data aggregated using the framework before introducing five privacy mechanisms that we will use throughout the remainder of this work. We explore several strengths and weaknesses of these privacy mechanisms.

3.1 Motivation

A common approach to collect, store, and mine a search log is to use a *centralized server model*: raw search interaction data from a user base is aggregated into a search log and stored in a central location. This is depicted in Figure 3.1. Search services

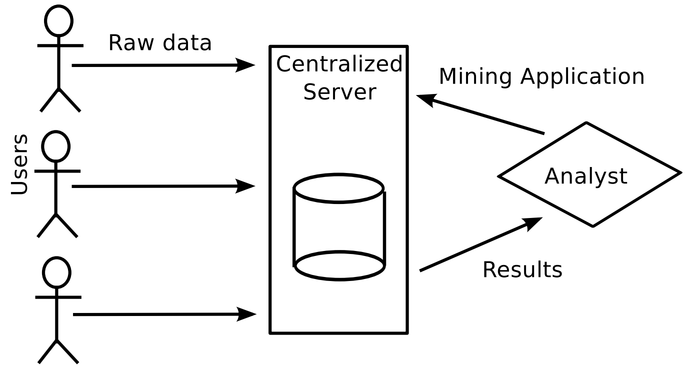


Figure 3.1. The centralized server model, wherein raw user data is uploaded to a centralized location where it can be minded by analysts.

can provide both personalized and non-personalized tools to users, but it requires that users submit their data to the centralized server. The data that users provide may contain sensitive activity and identifying information, raising serious privacy concerns. In addition, users have little control over what data is stored and used on the centralized server. Users must trust the data collectors to maintain their privacy.

What we would prefer is an approach to data collection, storage, and mining that protects user privacy, allows for personalization, and can aggregate data across users. Users should largely have control over their data, both in terms of what is mined and how it is collected. And they should not need to place unwaivering trust in the collectors. In addition, researchers that want to aggregate search behavior across users and distribute this information to the greater research community may be required by their institutional review boards to give due diligence in protecting user privacy. Likewise, large search companies that would like to distribute data about their users would also like to do so without violating their users' privacy.

3.2 Framework

CrowdLogging is a framework for privately aggregating search log data over a distributed group of users. It consists of four components: software installed on users' computers (client software), a bank of anonymizing nodes (anonymizers), a central server, and a privacy policy. Figure 3.2 shows a schematic of the framework.

The client software is responsible for logging a user's search activity to a search log stored on the client machine in its raw form. Though exactly what is logged is left up to the implementor, search logs likely contain web searches, visits to web pages, and clicks on web page links among other search events, along with timestamps. The client software is also responsible for mining data, such as queries, from the log whenever the server requests. We refer to these as mining applications, each of which is a function applied to the user log:

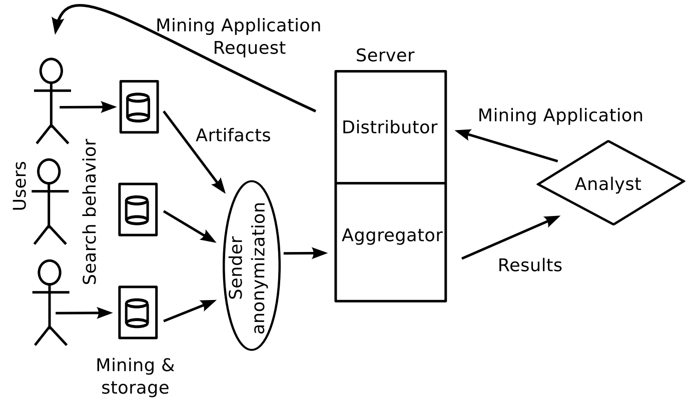


Figure 3.2. The CrowdLogging model. Raw data resides on users' computers. Analysts must request mining applications to be run on the raw data, the results of which are then encrypted, uploaded anonymously, and then aggregated and decrypted on a centralized server.

$$M : \mathit{UserLog} \rightarrow A,$$

where M is the application, *UserLog* is a user's search log, and A is a set of search artifacts. Recall that a search artifact is any piece of information extracted from a search log, such as a query, query reformulation, or query-URL pair. When a mining application is run, the client must encrypt each artifact $a \in A$ using a method that prevents a from being decrypted unless certain criteria are met, subject to the privacy policy in place—we describe this process in greater detail next. Once the artifacts are encrypted, they are uploaded to the server via a bank of anonymizers.

The anonymizers are a set of servers that shuffle data between other anonymizers before finally submitting the data to the centralized server. The purpose of this step is to anonymize the source of an artifact: the anonymizers strip IP information and any other data that reveals where an artifact comes from, rendering the artifacts themselves anonymous. Once data arrives at the centralized server, it can be aggregated and decrypted. The final output is a crowd log.

The encryption, aggregation, and decryption stages must adhere to a privacy policy. Recall that a privacy policy describes how search artifacts must be treated with respect to user privacy in order to form a crowd log. A simple privacy policy is to only include search artifacts in a crowd log that are mined from at least k user logs. To support frequency-based privacy policies (covering all the policies discussed in Chapter 3.4) in CrowdLogging, we rely on a secret sharing scheme for encryption called Shamir's Secret Sharing (Shamir, 1979). Under this scheme, a piece of information is encrypted and a partial key is provided. If enough partial keys are collected, the information can be decrypted. In CrowdLogging, an artifact is encrypted client-side along with a partial key specific to the contributing user. Under the k-users privacy policy, the encryption is performed such that k partial keys are required for decryption. Once the encrypted artifacts arrive at the server, they are grouped along with their partial keys. If at least k partial keys are collected, the artifacts are decrypted. The others are discarded. Note that if the server is compromised, the unencrypted artifacts could be decrypted using a brute force attack.

Under CrowdLogging, we allow encrypted artifacts to consist of two fields: a primary and a secondary field. The primary field is the one on which artifacts are grouped when being decrypted. The optional secondary field is data that can be revealed if the primary field is decrypted, but does not need to be shared among other encrypted artifacts. For example, if a mining application extracts search queries as the artifacts, then the primary field might be the search terms in all lower case (this is a type of normalization) whereas the secondary field may keep the original casing:

$$\langle \text{bars in boston}, \text{bars in Boston} \rangle.$$

Under the privacy policy mentioned above, the normalized query *bars in boston* would have to be extracted from at least k user logs, while each of them may have used a different casing. By allowing a secondary field in encrypted artifacts, we hope to allow analysts to recover small details about artifacts that may be useful, but would not be uncovered if only the primary field were available. How the secondary field is used should be specified in the privacy policy. In this thesis, we do not use the secondary field as it is difficult to quantify the degree to which it compromises privacy.

3.3 Artifact representation

When we extract data via a mining task with a particular IR application in mind, we must decide the form of the artifacts. For example, if we want to extract data for a query recommendation algorithm, it is likely that the artifact will be a query pair. However, the question is raised: how should the query pair be formatted? One method is to extract the original text of adjacent queries from a user's log to use as the artifact. Another approach is to convert the query text in each pair to lower case. Yet another approach is to additionally collapse adjacent whitespace in the query text. For example, the query *ASICS␣␣SHOES* (note the extra space in the middle) could be represented as is, as *ASICS␣SHOES*, or as *asics␣shoes*. Each of these is a different *artifact representation*.

Artifact representation plays a crucial role in CrowdLogging because it determines in part the ubiquity of the artifact, which effects how likely the artifact is to survive sanitization under a given privacy policy. More general representations will be at least as common as less general representations. In this section, we explore several artifact representation schemes and their effects on the simple privacy policy we described earlier, where an artifact must be extracted from at least k user logs to be added to a crowd log.

3.3.1 Exploring artifact representations

We explore artifact representations by considering several simple transformations on queries, query-click pairs, and query pairs from a subset of the AOL search log, spanning the month of March 2006. Without a target application, there are two primary measures of interest: the artifact impression coverage and distinct artifact coverage. Artifact impression coverage is the faction of the total number of artifact impressions that are shared by at least k users: $\frac{|C|}{|L|}$, where C is a crowd log and L is the set of artifacts from which the crowd log was generated. An impression is a single instance of an extracted artifact. Distinct artifact coverage is the fraction of the distinct artifacts shared by at least k users: $\frac{|Distinct(C)|}{|Distinct(L)|}$, where $Distinct(y)$ represents the set of distinct artifacts contained within the given data set y.

The goal of this analysis is to gain an understanding of the effect of artifact representation on both types of coverage. While the ultimate success metric is utility of the application that uses a crowd log, given two representations that are functionally equivalent, the representation that conflates more artifacts together leads to higher coverage, which means more artifacts will be made available in the crowd log. More data means more informed models, and thus increasing coverage is a reasonable goal. However, there are some cases—e.g., representing query pairs as feature vectors or trimming URLs to only the website domain—where over-eager artifact generalization will lead to greater coverage, but the quality of the artifacts will be so degraded with respect to a specific application as to make them useless. We explore the effects of sanitization on utility for specific applications in Chapters 5, 6, and 7.

For representing a query artifact, consider the following classes:

Original: The text as presented in the log.

Lightly cleaned: The strings “www.”, “.com”, “.edu”, “.net”, and “.org” along with punctuation are removed from the query text.

Heavily cleaned: First, the text is lightly cleaned. Then each white-space delimited term is removed if it belongs to a short list of blacklisted terms or stemmed[1] otherwise, after which all terms are reordered alphabetically.

[1] Stemming is the process of reducing a word to its root (or *stem*).

The queries in the AOL search log have already been converted to contain all lowercase and no redundant whitespace, so we do not consider those representations. The effect of the representations above for queries (both individually and in query-click pairs) on the number of distinct users that share those representations are shown in the plots in Figure 3.3.[2] The plots are cropped to show the effects up to $k = 100$ users. Both light and heavy cleaning make queries and query-click pairs more common, though the significance is less clear. For some applications, such as query intent classification,[3] it may be important to maintain substrings like "www" and ".com" as they suggest a user has a navigational intent. However, other applications may not require this information, in which case the improvement in impression coverage is worth the loss of data. The differences are also more pronounced for larger values of k, which suggests that these relatively simple representations are most useful in situations where k is large, as dictated by the privacy policy imposed on the system. The distinct coverage (the lower two plots) is less affected by the alternative representations.

For query reformulation pairs, which we will refer to as query pairs, we have added additional features:

Undirected: Query pairs are directed by nature: a pair (a, b) signifies that the query b was the first query to follow query a. This representation reorders the queries alphabetically. E.g., the pairs (a, b) and (b, a) resolve to the same pair (a, b).

Feature vector: Features are extracted from a pair of queries and then binned. We consider three features: the time between the queries (three bins), the Jaccard coefficient between the query text (three bins), and the number of character

[2] The 'distinct' plots report the fraction of the distinct artifacts of each respective *representation*, not the fraction of distinct original-form artifacts.

[3] For example, predicting if the user is looking for a homepage, like www.facebook.com, rather than for information.

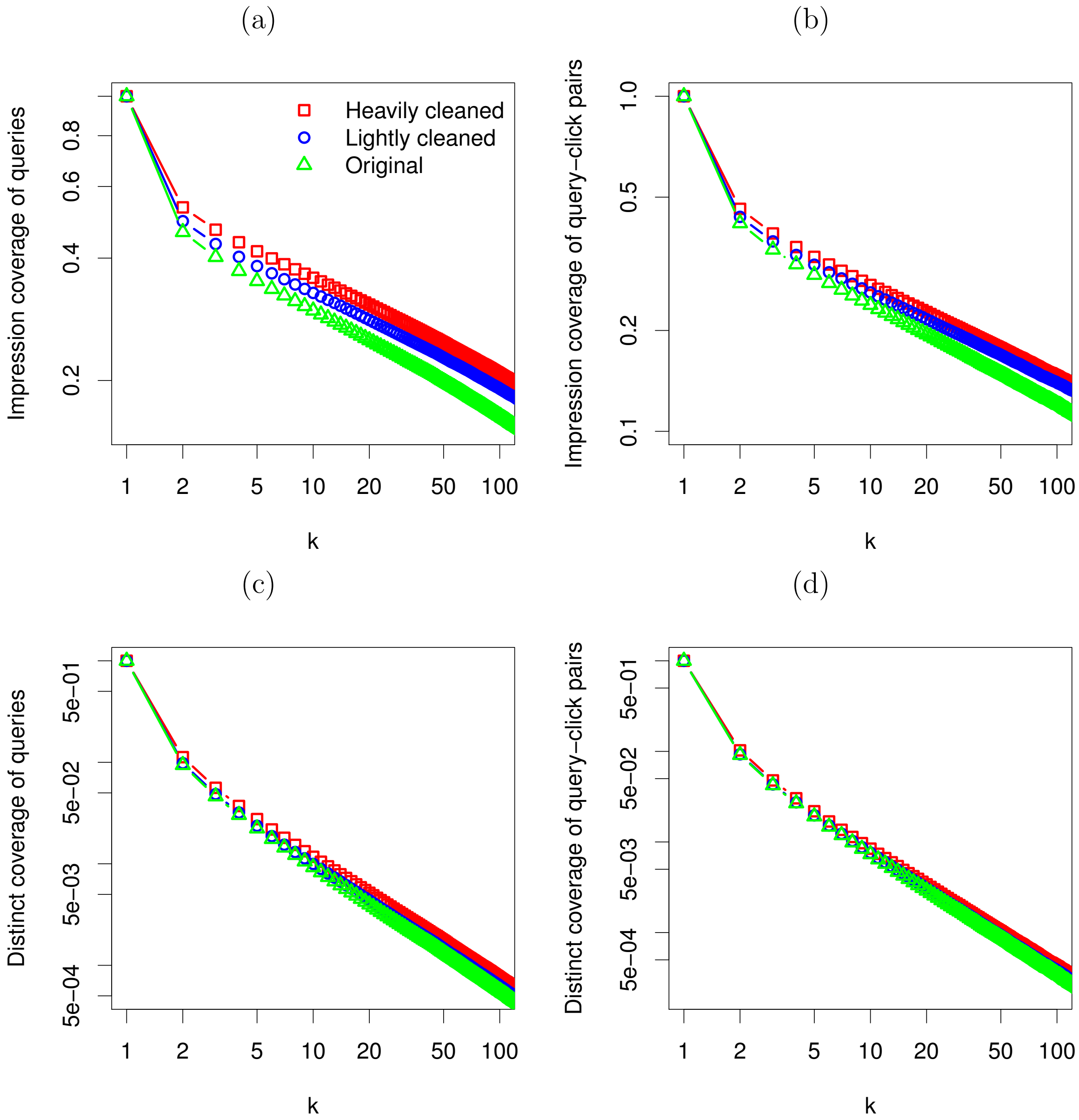


Figure 3.3. The impression coverage (top) and distinct coverage (bottom) of various representations for queries (left) and query-click pairs (right) as a function of the number of users (k) that share each artifact. Note that the axes are in log space.

trigrams that overlap between the query text (four bins). We describe the binning methods in detail in Chapter 5. The feature vector representation is useful when there is no need for text. For example, in search task identification as described in Chapter 5.

Figure 3.4 shows the portion of query pairs that are shared by different numbers of users. The graphs on the left are cropped to show the effects for $k = 1$ to 100 users. The full graphs are shown on the right and give a clearer picture of the feature vector representation. The representation to notice right away is the feature vector; its representation space is sufficiently low—consisting of $3 \times 3 \times 4 = 36$ possibly values—that $k = 1,222$ when the first artifact becomes unsupported. However, no text is recovered by using a feature vector representation. Of the other representations, we see that re-ordering the queries to be alphabetical increases impression coverage, but not as much as lightly cleaning the text. A combination of heavy cleaning and alphabetizing the pair results in a substantive improvement, especially for larger values of k. Similar to the query and query-click pair artifacts, the differences are less pronounced for distinct coverage (lower plots) than for impression coverage (upper plots).

3.4 Privacy

As we saw in Chapter 1, it is important to consider privacy when aggregating user data for use in IR applications. The CrowdLogging framework allows virtually any privacy policy to be integrated into the process. For example, a policy might mandate that only artifacts that occur in the logs of 100 different users can be published in the final crowd log. In this section, we first describe several mechanisms that are used in the information retrieval community to preserve user privacy and can be used as privacy policies in CrowdLogging. We demonstrate the potential utility of these mechanism by analyzing the coverage—the proportion of distinct artifacts or

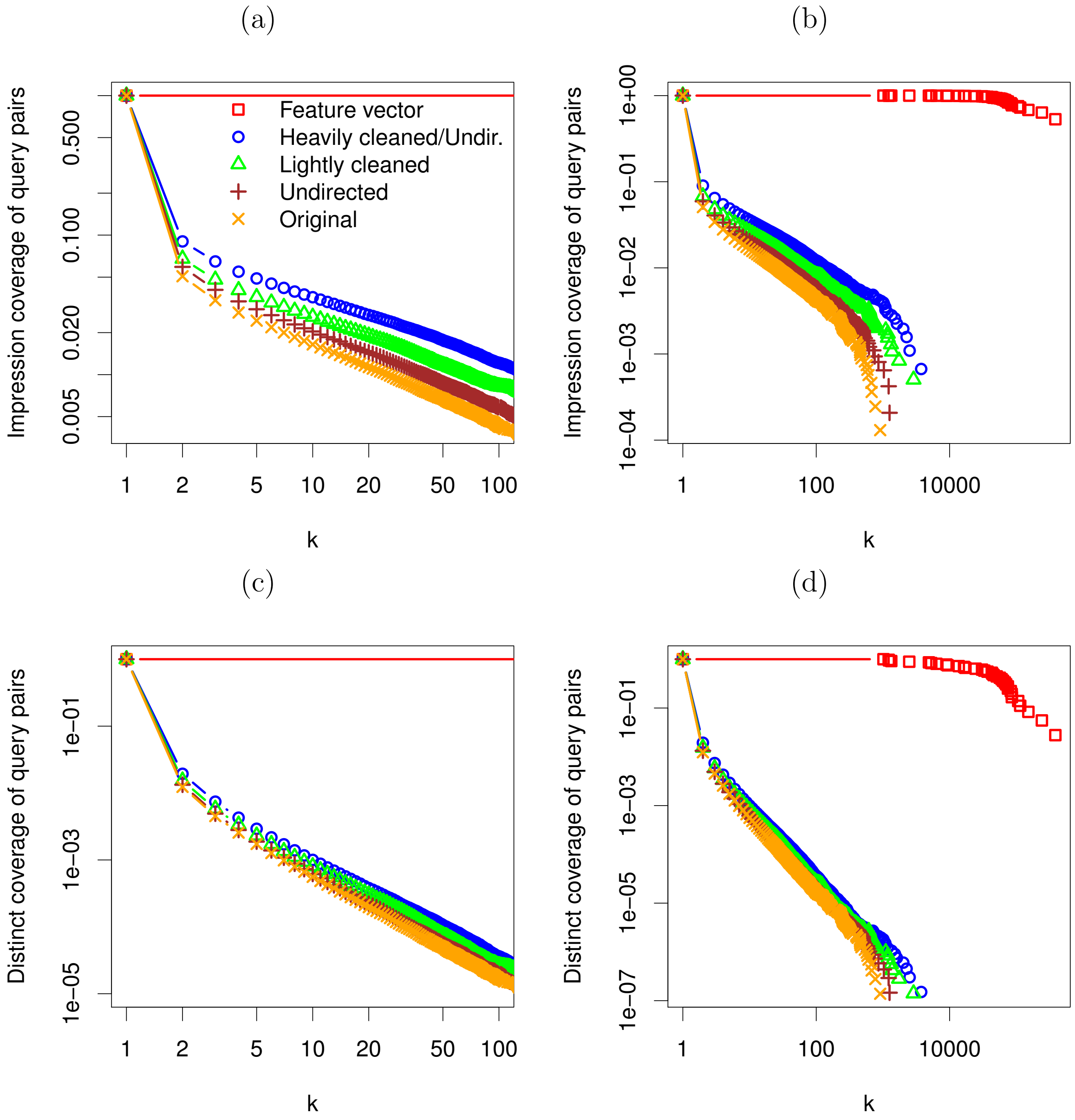


Figure 3.4. The impression coverage (top) and distinct coverage (bottom) of various representations for query reformulation pairs as a function of the number of users (k) that share each artifact.

artifact impressions released in the crowd log—for several artifact types. We also explore several issues surrounding these mechanisms. The contributions we introduce in this section are: 1) a novel privacy mechanism referred to as DP_u, 2) an analysis of attacks on a class of privacy policies called frequency thresholding (FT_u and FT_a), 3) an examination of the parameters settings required under differential privacy in order to obtain approximately the same coverage under frequency thresholding for releasing query artifacts from the AOL data set, and 4) an analysis of the privacy trade-offs involved with processing a long-running search log.

3.4.1 Privacy Mechanisms

In this work, we consider two classes of privacy policies: artifact frequency thresholding and differential privacy. For consistency with the literature, we refer to the individual functions as *mechanisms*, though any privacy mechanisms can be thought of as a CrowdLogging privacy policy. We have two goals with the privacy mechanisms we examine: 1) prevent sensitive information from being released and 2) prevent an individual from being identified. For the first goal, all mechanisms provide some kind of thresholding, assuming that artifacts that occur frequently or across multiple users will suppress sensitive information. For the second goal, all mechanisms release artifacts independently, meaning that there is no explicit link between two artifact impressions indicating whether they were extracted from the same user log. Our assumption is that if it is impossible to link two or more artifact impressions together, and only sufficiently supported artifacts are released, then an attacker will not be able to combine enough information to identify who contributed an artifact. For example, this would most likely have prevented the method used by the New York Times reporter that identified a user in the 2006 AOL data set (Barbaro et al., 2006). We describe both classes of privacy mechanisms below followed by an analysis of the potential utility both classes allow on the AOL query log.

3.4.1.1 Frequency thresholding

Frequency thresholding mechanisms are simple, straightforward, and arguably, quite natural. We define two such mechanisms: *artifact frequency thresholding* (FT_a) and *user frequency thresholding* (FT_u).

With the FT_a mechanism, an artifact is sufficiently supported if it occurs k or more times in a given search log. We call each occurrence an *artifact impression.* For example, if a user enters the same query many times, then each counts as one query artifact impression. A major weakness of this mechanism is that it will release sensitive artifacts if a user has entered them a sufficient number of times.

A mechanism that does not violate privacy as easily as FT_a is FT_u, in which an artifact is sufficiently supported if it occurs in k or more distinct user logs. This is very similar to the t-anonymity method described by Adar (2007), except that it applies to an arbitrary artifact. Its primary advantage over FT_a is that whether a sensitive artifact occurs only once or many times in a particular user log, it only increments the user count by one. Thus, a single user in isolation cannot cause an artifact to be released. However, we will see in the next section that this mechanism has several issues of its own.

There is no analytical notion of privacy quantification under frequency thresholding. As such, we allow an unlimited number of crowd logs to be generated from a given search log. For example, one crowd log for queries, another for query-reformulation pairs, and another query-click pairs. As we explore a little later, allowing multiple artifact types to be extracted from the same search log, combined with the lack of random perturbation of counts, makes it possible for attackers to link some artifacts to the same user and even infer unsupported artifacts.

3.4.1.2 Differential privacy

Unlike the frequency thresholding techniques described above, differentially private mechanisms provide theoretical bounds on the amount of privacy leaked, and thus can be considered provably private, provided a number of assumptions hold. In this section, we introduce differential privacy and briefly describe several relaxations introduced in the information retrieval literature to publish data from search logs privately. We describe the algorithms of Korolova et al. (2009) and Götz et al. (2011) in addition to our variation of the Korolova et al. algorithm (Feild et al., 2011). Each offers a different view of the data and incurs different privacy and utility costs.

Differential privacy, introduced by Dwork (2006, 2008), is a term applied to a randomized privacy mechanism $\mathcal{A}$, meaning $\mathcal{A}$ will produce nearly identical output given two very similar inputs. We can quantify the difference in outputs by a parameter ϵ and when doing so, we refer to the mechanism as being ϵ-differentially private. For example, if we consider the input to be a data set of query instances extracted from users and $\mathcal{A}$ to be an algorithm that picks which artifacts to publish, the output of $\mathcal{A}$ given a set of queries should be nearly identical to the output given the same set of queries, but with any single individual's query instances removed. This means that an analyst should not be able to tell from the output of $\mathcal{A}$ whether or not any particular user's queries were included *or excluded* from the input search log. The formal definition is as follows:

Definition 3.4.1 (ϵ-Differential Privacy (Dwork, 2008))**.** A randomized mechanism $\mathcal{A}$ is ϵ-differentially private if for all data sets D and D' that differ in one individual's data, and all $S \subseteq \text{Range}(\mathcal{A})$:

$$Pr[\mathcal{A}(D) \in S] \leq \exp(\epsilon) \cdot Pr[\mathcal{A}(D') \in S]$$

Note that when the privacy parameter ϵ is decreased, the difference in the outputs is exponentially decreased: the smaller ϵ, the less privacy is leaked, but the less accurate the output. When $\epsilon = 0$, no privacy is leaked, but the output becomes useless.

One consequence of this definition is that if an attacker were to know the contents of all but one user log, they should not be able to infer with any certainty what the held out user log's contents are. However, in order to make this claim, we need to make a few assumptions about search logs as pointed out by Kifer and Machanavajjhala (2011). First, we assume that each individual user's search behavior is independent of all other users. Second, we assume that we are the only holders of the input data set and therefore no one else has released information—sanitized or not—from that data set. Finally, we assume that any information released from earlier search logs that contain search behavior from one or more of the users contained within our data set is not correlated. This last assumption is necessary because information derived from correlated data sets can provide an attacker with enough information to infer information from the sanitized crowd log of the current data set.

Differential privacy was created with databases in mind and Götz et al. (2011) demonstrated the impracticality of achieving reasonable accuracy from the output of an ϵ-differentially private mechanism on a search log. They analyzed inaccuracy—the probability of not publishing very frequent items as well as the probability of publishing very infrequent items from the input search log—and showed that publishing nothing is more accurate than publishing data under ϵ-differential privacy.

Two relaxations of differential privacy have been proposed for sanitizing search logs: *indistinguishability* (DP_a (Korolova et al., 2009), DP_u (Feild et al., 2011)) and the less relaxed *probabilistic differential privacy* (ZEALOUS (Götz et al., 2011)). We provide a high level sketch of these here and describe the technical details in Appendix B. Throughout this work, we will refer to this family of mechanisms as "differentially private mechanisms" for ease, even though they are relaxations.

The DP_a mechanism (Korolova et al., 2009) is a form of (ϵ, δ)-indistinguishability. This introduces a slack variable δ to allow a limited amount of additional privacy to be leaked. DP_a produces a noisy histogram of artifacts. The algorithm allows each user to contribute d artifacts to a collection. Noise sampled from a Laplacian distribution with variance b is added to the impression count of every collected artifact and checked against a threshold k; if the noisy count surpasses the threshold, the artifact is added to the crowd log with a new count: its original count plus newly sampled noise. By setting the ϵ, δ, and d parameters, the noise parameter b and k can be maximized. DP_a is the differentially private parallel of FT_a.

Our mechanism, DP_u (Feild et al., 2011), is based on DP_a, but varies in one substantial way: rather than requiring that artifacts in the crowd log occur at least k times, we require that they have been contributed by at least k users. This distinction has ramifications for how k is computed, allowing k to be higher for the same d relative to DP_a. DP_u is the differentially private parallel of FT_u.

The final mechanism we explore is ZEALOUS (Götz et al., 2011), which also produces a noisy histogram, but using a process that is (ϵ, δ)-probabalistically differentially private, a more conservative relaxation of differential privacy than indistinquishability. ZEALOUS begins by collecting d distinct artifacts per user. Of the collected artifacts, those that occur less than k' times are discarded. Noise is added to those that remain and if the noisy count exceeds a second threshold k, the corresponding artifact is added to the published histogram. The required parameters are: ϵ, δ, d, and U, where U is the number of users in the input data set.

There are many constraints imposed by differential privacy and its relaxations that can cause issues for search log analysts. First, in order to bound privacy guarantees, only a limited number of artifacts may be extracted from each user. This potentially ignores a large percentage of the original search log.

Second, the privacy that is leaked each time an algorithm is run on the same search log is additive—if we extract a set of queries Q and query pairs QP from the same log, then produce $CL_Q = DP_u(Q)$ and $CL_{QP} = DP_u(QP)$ with the parameters ϵ, δ, and d, then $CL_Q \cup CL_{QP}$ are $(2\epsilon, 2\delta)$-indistinguishable. That is, we have doubled the amount of privacy that is leaked than if we had only extracted one set of artifacts.

Third, the number of artifacts that are released under differential privacy is substantially less than for a non-provably private mechanism, such as frequency thresholding. This property of privacy preserving mechanisms is perhaps the most difficult barrier in providing reasonable utility for applications, as we will see next.

Despite these issues, differential privacy based mechanisms provide a means of quantifying the amount of privacy leaked during the creation of a crowd log.

3.4.1.3 Coverage of privacy mechanisms

One way to assess the utility of a privacy mechanism is to consider the resulting crowd log's coverage of distinct artifacts and artifact impressions from the search log. While we will evaluate performance for three individual IR applications later in this thesis, here we consider the coverage for several artifact types.

In Figure 3.5, we show the coverage for queries, query reformulations, and query-click pairs across different levels of k. The first thing to notices is that the distinct coverage is extremely low even at very low values of k. This is because the majority of query, query pair, and query-click pair artifacts are unique, or at least rare. Depending on the application, low distinct artifact coverage may or may not matter. In cases where artifacts corresponding to trends are desired, rare artifacts are not necessary to achieve high performance. We consider the effects of artifact coverage on performance for three applications in Chapters 5–7.

The two frequency thresholding mechanisms always produce higher coverage on this dataset for $k < 200$. We can also see that while FT_a and FT_u are very similar,

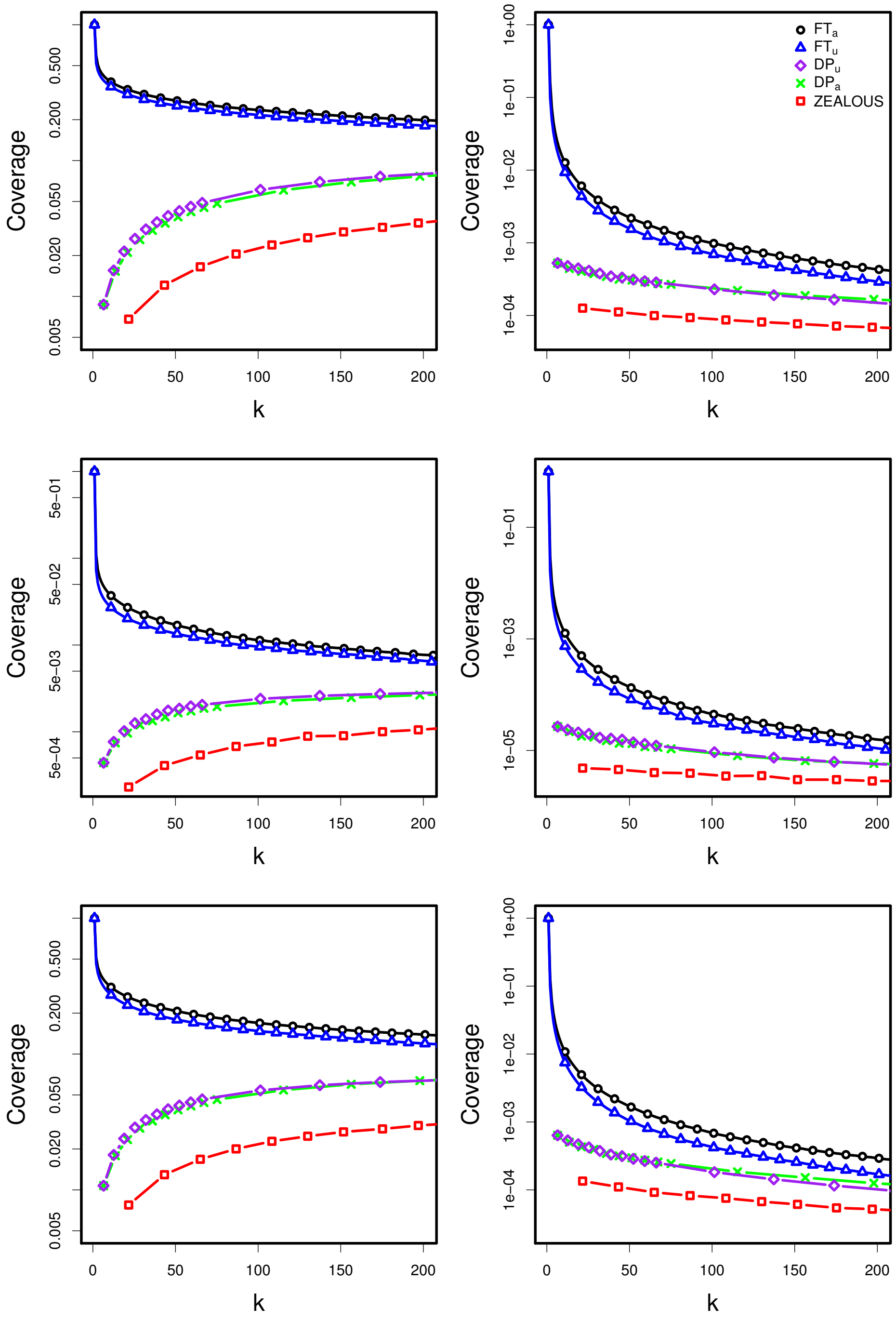


Figure 3.5. The impression coverage (left) and coverage of distinct artifacts (right) across five privacy mechanisms for query artifacts (top), query reformulation pairs (middle), and query-click pairs (bottom), extracted from three months of AOL logs. The y-axis is shown in log scale. For DP_a, DP_u, and ZEALOUS, $\epsilon = \ln(10)$ and $\delta = 1/657427$.

FT_a generally performs better (releases more data). DP_a and DP_u also perform quite similarly to each other. ZEALOUS consistently provides the lowest coverage. Both frequency thresholding techniques behave similarly on impression versus distinct artifact coverage: as k increases, coverage falls, quickly at first before tapering off. The differentially private mechanisms are a bit different; coverage actually increases with k when considering artifact coverage. This is due to the varying d associated with any level of k. By holding constant ϵ and δ, we must increase d in order to achieve higher values of k, allowing more head artifacts with their high impression counts to be collected. The differentially private mechanisms show the opposite trend with respect to distinct artifact coverage. While d increases with k, the increase in k prevents less common artifacts from being released, hampering the distinct coverage.

3.4.2 Exploring privacy mechanisms

Having introduced several privacy mechanisms, we now discuss some of the questions we have regarding them. We explore three aspects. First, we consider some of the attacks to which frequency thresholding mechanisms are vulnerable. Second, we approximate the privacy leaked when producing crowd logs with frequency thresholding. We do this by tuning the ϵ and d parameters of the differential privacy mechanisms until they produce very similar results to those produced by FT_a and FT_u. Finally, we consider the trade-offs involved with segmenting a search log, extracting a crowd log from each segment, and combining the results. We are motivated by the potential of using a greater number of each user's artifacts, but as we will see, smaller segments require stronger privacy restrictions, which results in fewer artifacts being supported.

3.4.2.1 Compromised privacy under frequency thresholding

Frequency thresholding allows more artifacts to be released than differentially private mechanisms because it neither requires noise to be added nor limits the number

	Artifact	**Instances**
$\triangleright$	bob	5,000
$\triangleright$	netflix	6,000
	humane society	10
$\triangleright$	westminster, md	5,000

Table 3.1. An extreme example of where a power user can be distinguished from other users, allowing that user's artifacts (marked by $\triangleright$) to be linked.

of artifacts any single user can contribute. However, this same advantage results in the inability to theoretically quantify the privacy compromised by frequency thresholding and leads to several instances of blatant privacy violations. In this section, we look at several examples of how privacy might be compromised under frequency thresholding. For these examples, let $Count(x, y)$ be the number of instances of the artifact x in the data set y.

Example 1 Suppose we have U users contributing search data. Of these users, we have one user, Bob, who is a power user—he has performed vastly more searches than any other user. Now suppose we extract all users' queries Q and we publish a histogram CL_Q of artifacts and their counts: $CL_Q = FT_u(Q)$, i.e., each artifact $a \in CL_Q$ is supported by at least k distinct users. Of the artifacts in CL_Q, some subset B intersects with the artifacts mined from Bob's search log. Because Bob has so many more searches than any other user, we find that the artifacts in B have significantly higher impression frequencies than the published artifacts not in B. Although we cannot say for sure, a careful analyst could reasonably suspect that there exists a user from which those B artifacts were mined, as depicted in Table 3.1. This allows artifacts to be linked, which could then reveal the user's identity. FT_a, too, falls victim to this attack. The provably private mechanisms discussed before are not susceptible to this attack, mainly because each user can only contribute a fixed number of artifacts.

Example 2 A second kind of attack leverages the ability to extract an unlimited number of crowd logs from a single search log. Let a, b, and c be three distinct queries. We extract queries (Q), adjacent query pairs (QP), and adjacent query triples (QT) and publish their respective sanitized logs with $k = 5$: $CL_Q = FT_u(Q)$, $CL_{QP} = FT_u(QP)$, and $CL_{QT} = FT_u(QT)$, which is empty. Table 3.2 shows that, because the queries a, b, and c and the query pairs (a, b) and (b, c) are sufficiently supported and we know exactly what their frequencies are, we can infer that at least one user has the *insufficiently supported* query triple (a, b, c). This has two repercussions: first, it means that crowd logs generated by FT_u can be used to generate data that is inconsistent with what FT_u would itself allow to be released, and second, if the queries a and c are identifying when taken together, this breach compromises the identity of the affected users. FT_a can be similarly attacked.

To demonstrate an instance of this kind of breach on the AOL query logs, we use the following algorithm. Any time we see a query q_i that occurs in CL_Q with $k = 5$, we will examine the query pairs in CL_{QP} and verify two conditions: (1) if there exists a query q_{i-1} that immediately precedes q_i such that $\mathit{Count}((q_{i-1}, q_i), CL_{QP}) > \mathit{Count}(q_i, CL_Q)/2$ and (2) if there exists a query q_{i+1} that immediately follows q_i such that $\mathit{Count}((q_i, q_{i+1}), CL_{QP}) > \mathit{Count}(q_i, Q)/2$. If both conditions hold and we assume that the statistics reported in CL_Q and CL_{QP} are accurate, then we can infer with 100% accuracy that the query triple (q_{i-1}, q_i, q_{i+1}) exists in at least one user log. A breach occurs when at least one of these triples lacks sufficient support for $k = 5$, which is what was used to generate CL_Q and CL_{QP}.

On the AOL query log, we can infer 40 query triples this way. Of those 40 triples, 33 are sufficiently supported at $k = 5$, that is, they occur in CL_{QT} with $k = 5$. That means that 7 query triples were entered by fewer than 5 distinct users, but inferable. This is a relatively small fraction of all unsupported triples—there are a

	Artifact	Instances
	Queries	
	a	5
	b	7
	c	5
	Query pairs	
	a, b	5
	b, c	5
	Query triples	
$\triangleright$	a, b, c	3

Table 3.2. An example where running multiple experiments on the same data can reveal information, e.g., the query triple demarcated by $\triangleright$ is not sufficiently supported at $k = 5$. We can infer that the triple (a, b, c) was contributed by at least one individual. The user count of each artifact is not shown, but assume all query and query pair artifacts are sufficiently supported at $k = 5$.

total of 19,402,216 unsupported triples at $k = 5$—but nonetheless, it demonstrates the vulnerability of frequency thresholding.

Example 3 A third example of a breach of frequency thresholding is as follows. Assume Bob is analyzing a set of logs sanitized with FT_u and knows (1) that his friend Alice's user log is part of a given search log and (2) that Alice entered a query q_i at some point in the recent past.[4] Assume that q_i appears in CL_Q with $Count(q_i, CL_Q) = n$. Now suppose that all possible pairs involving q_i appear in the accompanying sanitized query pair log, CL_{QP}, i.e., $Count((\cdot, q_i), CL_{QP}) + Count((q_i, \cdot), CL_{QP}) = n$. Furthermore, assume that the query q_i is only ever preceded by one query, q_{i-1}. Then every user that submitted the query q_i also submitted the preceding query q_{i-1}. Therefore, Bob now knows that Alice submitted q_{i-1}.

These examples demonstrate that frequency thresholding is vulnerable to revealing information that is inconsistent with its policy as well as information that can be

[4] This is not an unfair assumption. For instance, office workers can easily see coworkers' computer screens in many office settings. In addition, people sometimes post search queries on social networking sites and forums.

combined with auxiliary information to establish a connection between individuals and artifacts. These are edge cases and might be rare, but we stress that when privacy matters, so do the edge cases. It would seem that adding noise to the released counts and limiting the amount of data a single user can contribute would go a long way towards addressing these vulnerabilities. However, the concern then becomes how much noise to add and how much data can each user contribute. This is exactly the niche that differential privacy and its relaxations attempt fill.

3.4.2.2 Obtaining similar coverage across mechanisms

As we saw earlier in this chapter, frequency thresholding mechanisms substantially out-perform their differentially private counterparts in terms of coverage on the AOL data set. Ideally, we would be able to achieve the relatively high coverage rates that we observed for frequency thresholding and at the same time have the theoretical understanding that braces differentially private mechanisms such as DP_a and DP_u. With this ideal situation in mind, two questions of practical interest are: 1) what DP_a and DP_u parameter settings would be necessary to achieve a similar coverage to FT_a and FT_u, respectively?[5] and 2) are those settings reasonable? In this section, we address these questions specifically for query artifacts released using the AOL search log. Although the specific conclusions we draw do not necessarily generalize to other data sets or artifact classes, the process we use and the analysis we conduct can be applied to other data sets and artifact classes.

DP_a and DP_u have several parameters: the number of artifacts to sample per user d, the threshold k, the noise parameter b, the privacy parameter ϵ, and the privacy slack value δ. As we describe in Appendix B, the typical way to use DP_a and DP_u is to set k, ϵ, and δ, calculate the optimal d, and then compute b (which is based on

[5] We pair these mechanisms for the sole reason that they share similar thresholding strategies, namely DP_a and FT_a threshold on artifact frequency and DP_u and FT_u threshold on user frequency.

ϵ and d). In approximating the coverage produced using frequency thresholding, we assume that a major factor is the number of artifacts sampled per user. If we only sample a small portion of artifacts per user under differential privacy, then we likely will not be able to achieve nearly the same coverage as under frequency thresholding. Given this assumption, we will hold d constant and instead optimize ϵ for a given value of k in this section.[6]

The first detail we consider is how to choose d. While frequency thresholding considers all artifacts per user, we need to bound d when using the DP_a and DP_u mechanisms. If we set $d = 100$ then 92% of users contribute 100% of their query artifacts in the AOL data set. If we wanted 95% or more of users to contribute 100% of their query artifacts, we would need to increase d substantially; instead, we use $d = 100$ and claim that it covers a sufficient majority of users. We should note that in a realistic setting, we would not know what value of d would result in the majority of users contributing all of their artifacts without incurring a privacy cost (even then, it would be a noisy count).

For these experiments, we set $d = 100$, and hold δ constant at $1/U = 1/657427$ (i.e., one over the number of users in the data set). We can then vary k and compute the optimal ϵ, and then calculate $b = d/\epsilon$. To see how ϵ fluctuates relative to k, we have plotted the two in Figure 3.6 for the DP_a, DP_u, and ZEALOUS mechanisms. ZEALOUS is shown as a comparison point, though we will not use it for our experiments in this section. We can see that for lower values of k (meaning more data is released), ϵ is quite high—high enough to be practically meaningless. However, as k increases, ϵ decreases quickly.

In Table 3.3, we show the optimal value of ϵ under DP_a and DP_u for a given threshold, k. Our hypothesis is that using this table, we should be able to establish

[6] In actuality, ϵ cannot be analytically solved for under DP_a; rather we must sweep a range of ϵ values to find the one that produces the desired value of k, or one that is close.

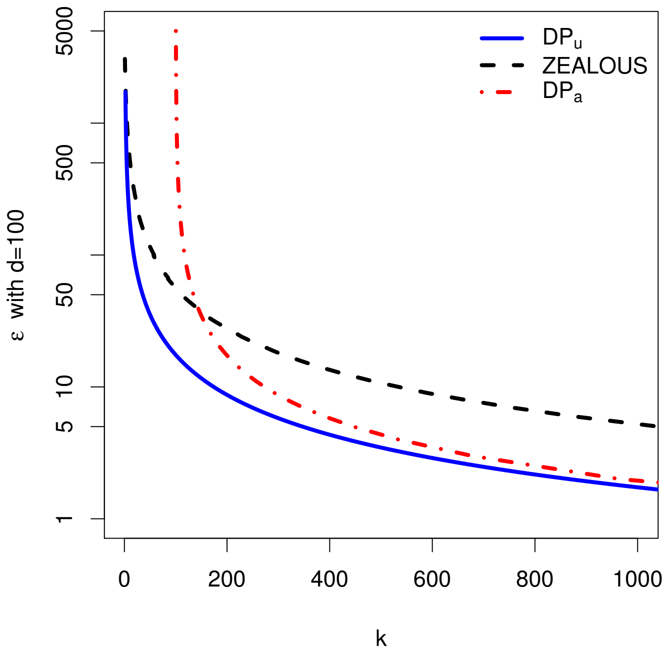


Figure 3.6. The ϵ required in order for obtain a given level of k for each of the three differential-privacy mechanisms. Mechanisms use $d = 100$, $U = 657427$ (the number of users in the AOL data set), and $\delta = 1/U$.

DP_u		DP_a	
k	ϵ	k	ϵ
1	–	–	–
2	1730.81	–	–
4	576.94	–	–
8	247.26	–	–
16	115.39	–	–
32	55.83	–	–
64	27.47	101	1730.82
128	13.63	128	61.82
256	6.79	256	11.10
512	3.39	512	4.21
1024	1.69	1026	1.87

Table 3.3. The optimal ϵ for a given value of k. We use $d = 100$, $U = 657427$ (the number of users in the AOL data set), and $\delta = 1/U$. Note that under DP_a, ϵ cannot be determined analytically given a specific k. Rather, we sweep across values of ϵ with a step size of 0.01 and use the ϵ that yields the closest value of k to the target (i.e., the value of k listed for DP_u).

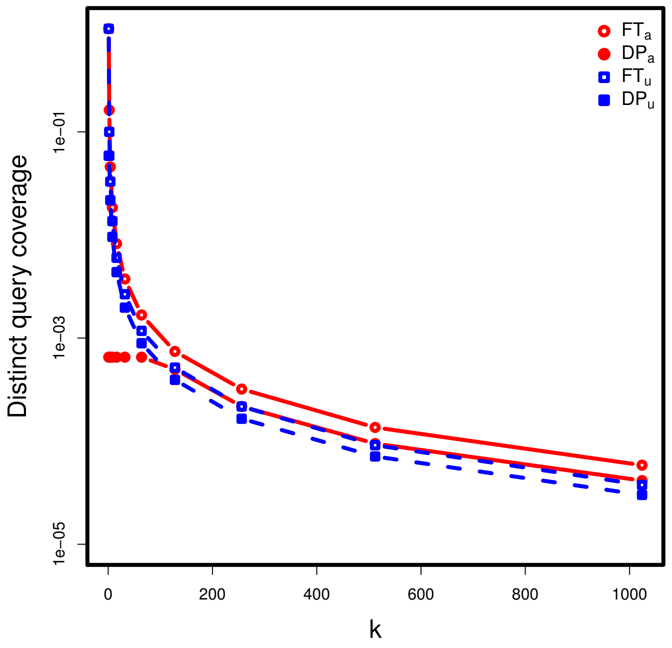


Figure 3.7. The distinct query coverage over the AOL search log using FT_a and FT_u with all data and DP_a and DP_u with $d = 100$, $\delta = 1/657427$, and the value of ϵ shown in Table 3.3. DP_a and DP_u coverage is averaged over ten samples with noise added.

the ϵ necessary to achieve a query artifact coverage similar to FT_a and FT_u at a given value of k on the AOL data set. To put this hypothesis to the test, we compare the distinct query artifact coverage obtained under FT_a and FT_u to DP_a and DP_u, respectively. The results are shown in Figure 3.7. We can see that the coverage between the FT and DP mechanisms is close for query artifacts. This signifies that our method of setting parameters for DP_a and DP_u to approximate coverage under frequency thresholding is effective, at least for extracting query artifacts from the AOL data set. However, as we noted earlier, the privacy parameter ϵ optimized for lower values of k under both DP_a and DP_u is impractically high. Achieving coverage similar to that produced under frequency thresholding using differential privacy mechanisms requires unreasonable settings.

Note that the values of ϵ shown in Figure 3.6 and Table 3.3 are dependent on two parameters: first, the number of artifacts contributed per user, d—we picked a value that we knew covered a large portion of users. For different artifacts, this number will need to change if an implementor wants to encapsulate the majority of users. While we assumed an oracle value, in a live system, the implementor will either have to extract the number of artifacts each user has, which will incur a privacy cost, or an estimate will have to be made, in which case the resulting coverage may not be as close to frequency thresholding as our experiments demonstrated. Second, we set U to the number of users in the data set; for a new data set, U should be adjusted based on that data set's size.

By considering the parameter settings required to make the coverage of DP_a and DP_u approximate that of frequency threshold, we are not drawing any connection between the underlying mechanisms. It is important to note that the differential privacy parameters do not apply to the frequency thresholding mechanisms themselves—they cannot, since frequency thresholding mechanisms do not satisfy the differential privacy definitions. It would be incorrect to use, e.g., FT_a, and then state that its corresponding privacy loss at k is a particular value ϵ. Rather, the differential privacy parameters can only be used to quantify privacy loss when used with the corresponding differentially private mechanism.

As a final note, we emphasize that our results in this section pertain only to query artifacts extracted from the AOL data set and do not necessarily generalize to other artifacts or other data sets.

3.4.2.3 Sanitizing search logs more effectively

A live logging system must cope with continually growing user data and how to use it effectively as soon as possible. For frequency thresholding, there is no issue—it can be run over data an unlimited number of times, meaning the old and new data can be concatenated and processed under FT_u or FT_a. This is not the case for the differentially private mechanisms without leaking additional privacy—if we generate

n crowd logs, each with parameters (ϵ, δ), the total privacy leaked is captured by $(n \cdot \epsilon, n \cdot \delta)$. In this section, we begin to analyze how differentially private mechanisms can be more effectively applied to growing search logs.

Our goal is to maintain the privacy parameters ϵ and δ. The variables we can set are the number of artifacts to sample from each user d, and the set of artifacts from which the d artifacts are sampled. We can vary the latter without affecting the privacy parameters because differentially private algorithms are only dependent on the number of artifacts per user and the total number of users. Consequently, an input data set D with 300 artifacts is treated exactly the same as a data set D' with 3 million artifacts. In both cases, we sample d artifacts per user. Provided d remains the same, $|D| - d < |D'| - d$; that is, fewer artifacts are ignored from D than from D'. At the same time, however, samples taken from D' may provide better coverage due to the possibility of increased artifact diversity.

One way to control the size of the logs is to constrain the time frame—for example, sanitizing a day's, week's, or month's worth of user logs. Korolova et al. (2009) explored this by directly comparing the coverage of query artifacts for different time spans, finding that distinct artifact coverage is not substantially affected by time span, while artifact impression coverage is dependent on both the timespan and d, with higher levels of d and longer time spans providing better coverage. However, a more useful analysis is to compare the utility of one sanitized log (e.g., a month long log) versus combining n sanitized subsets of that log (e.g., 31 day-long logs).

A complicating factor is that each time span is, under conservative assumptions, correlated with every other time span. This assumes that a user's search activity one day is dependent on their search behavior another day. Because this data is correlated, the privacy costs must be summed across the time spans. Suppose we want to maintain $\epsilon = \ln(10)$ and $\delta = 1/657427$ as the cost for our crowd log and a month is the longest time span we consider. The crowd log generated from the entire month would stay at $\epsilon = \ln(10), \delta = 1/657427$; each of two half-month time spans would use $\epsilon = \ln(10)/2, \delta = 1/(2 \times 657427)$; and each of 31 day-long logs would use $\epsilon = \ln(10)/31, \delta = 1/(31 \times 657427)$. Since ϵ, δ, and d interact to establish k, it may be that we can only sample so few artifacts per shorter time frame, or k becomes too large, that nothing is gained.

The trade off, then, is sampling a larger portion of each user's artifacts versus higher values of k, limiting the output, especially of tail artifacts. To understand these trade offs empirically, we conducted an experiment over the March 2006 segment of the AOL search log. Figures 3.8 and 3.9 show the coverage across query, query pair, and query-click pair artifacts extracted with the DP_a, DP_u, and ZEALOUS privacy mechanisms and various d settings. the trends make it clear that using longer spanning logs provide greater coverage. However, day-long logs do provide an advantage, if very slight, when extracting query artifacts when $d = 1$. For less common artifacts like query-pairs, longer spanning logs are essential to reveal anything, especially when using ZEALOUS—the month-long log is the only log that provided non-zero coverage.

The take away from this analysis is that longer time spans provide greater coverage than aggregating crowd logs produced from smaller segments of the log. If the assumption that log segments with overlapping users are correlated can be challenged, then the privacy costs would not need to be summed. Then, perhaps, aggregating multiple crowd logs would be a better solution.

3.5 Summary

In this chapter, we introduced CrowdLogging, a framework for logging, mining, and aggregating data in a privacy-aware manner. We demonstrated the impact that artifact representation has on coverage. We also introduced five privacy mechanisms that can crowd logs within the CrowdLogging framework, including a formalization

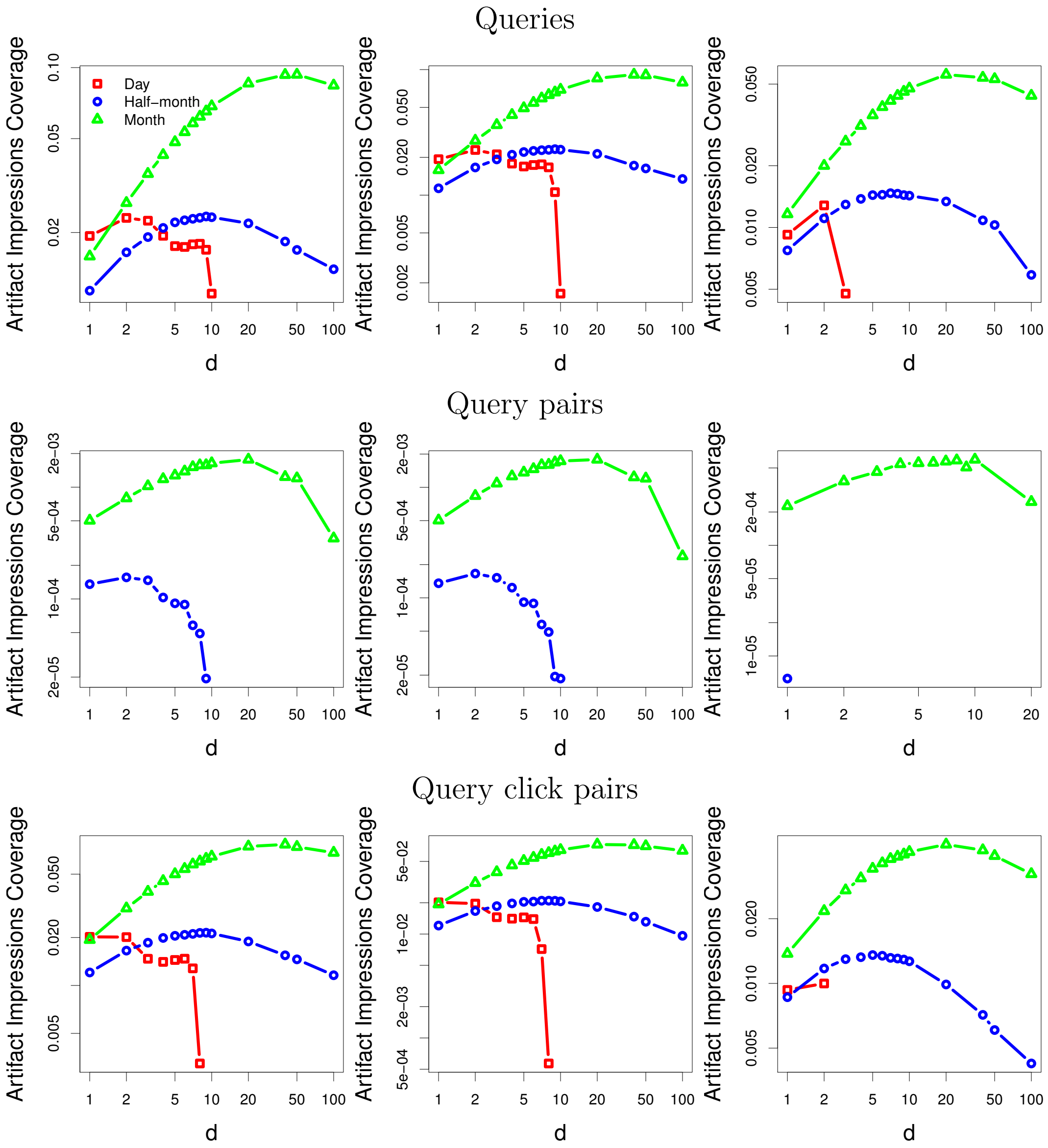


Figure 3.8. The impressions coverage of queries (top), query pairs (middle) and query-click pairs (bottom) achieved by combining 31 1-day sanitized logs (Days), two half-month sanitized logs (Half-month), and a single month sanitized log (Month) for March, 2006 in the AOL search log. For each privacy mechanism DP_a (left), DP_u (middle), and ZEALOUS (right) we used $\epsilon = \ln(10)$ and $\delta = 1/657427$. This is plotted on a log-log scale.

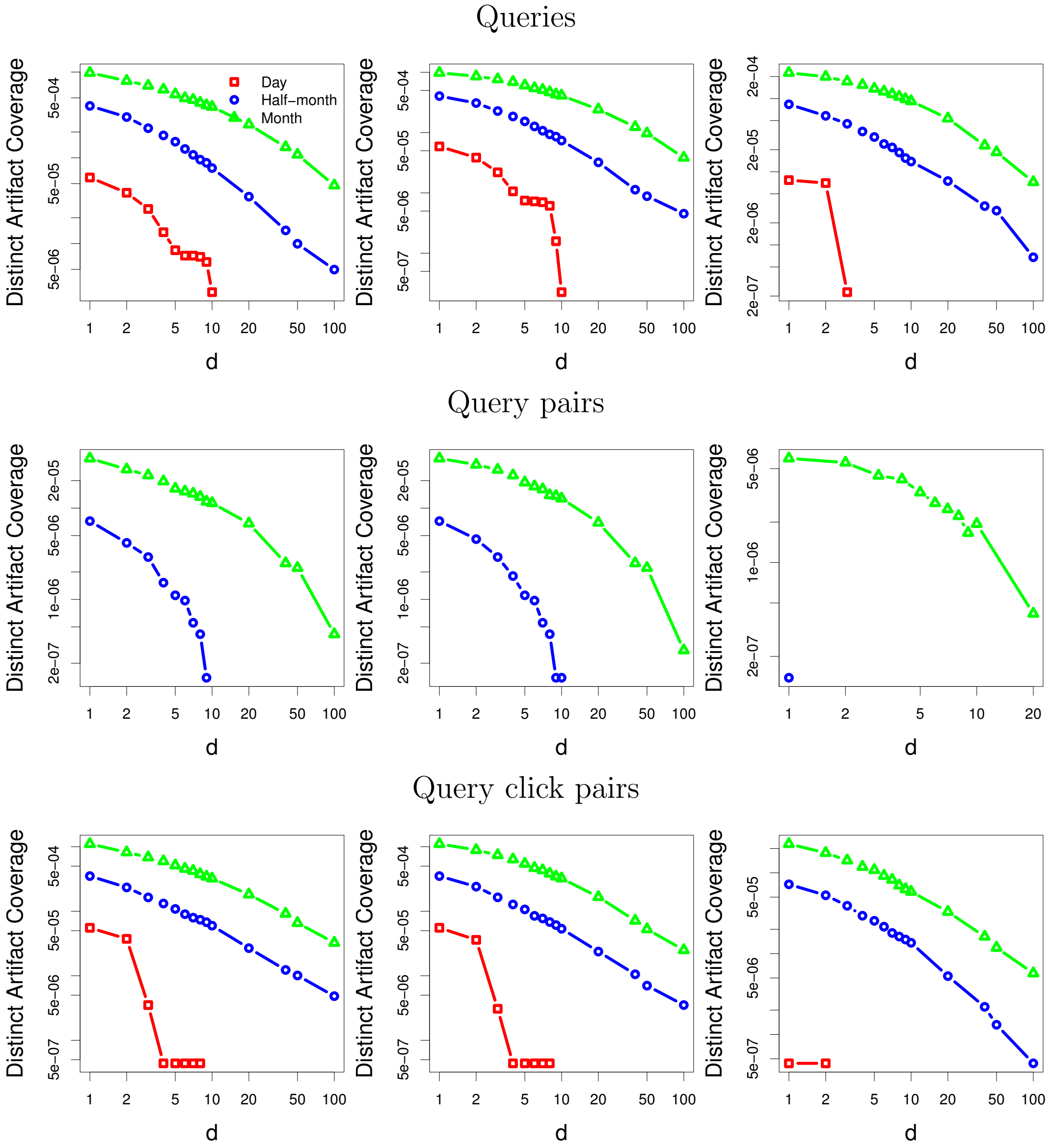


Figure 3.9. The distinct artifact coverage of queries (top), query pairs (middle) and query-click pairs (bottom) achieved by combining 31 1-day sanitized logs (Days), two half-month sanitized logs (Half-month), and a single month sanitized log (Month) for March, 2006 in the AOL search log. For each privacy mechanism DP_a (left), DP_u (middle), and ZEALOUS (right) we used $\epsilon = \ln(10)$ and $\delta = 1/657427$. This is plotted on a log-log scale.

of two naïve frequency thresholding mechanisms, FT_u and FT_a as well as a novel (ϵ, δ)-indistinguishable mechanism, DP_u. We evaluated their potential effectiveness using distinct artifact coverage and artifact impression coverage for three artifact types: queries, query pairs, and query-click pairs. We described several ways in which crowd logs released using frequency thresholding can be attacked to link artifacts, identify user participation, and infer insufficiently supported artifact. We empirically measured the frequency with which some insufficiently supported artifacts can be inferred from AOL crowd logs. We explored parameter settings for DP_a and DP_u to achieve approximately the same query artifact coverage produced using frequency thresholding on the AOL search log. We found that ϵ must be set impractically high, especially for lower values of k, causing us to conclude that it is not possible to produce coverage similar to frequency thresholding using reasonable parameter settings. Finally, we explored segmenting search logs into smaller sets in order to obtain higher coverage. However, we found that processing larger logs produces better coverage at the same privacy loss.

The impact of these analyses is of a practical nature. A query log analyst interested in privacy must make many decisions when deciding what mechanism to use, how to process data, and what the consequences of those choices are. Our analyses provide insights on these fronts with respect to a large search log.

CHAPTER 4
INTEGRATING LOCAL AND GLOBAL INFORMATION

In this and the following chapters, we explore integrating local user data with globally aggregated data for use in IR applications, introducing a general template for implementing IR applications that leverage both kinds of data. We look specifically at three applications: search task identification, context aware query recommendation, and context aware website suggestion.

We first motivate why IR applications should make use of rich local user search data in addition to sanitized global data aggregated across many users. We then present a general local-global data integration template, to which all IR applications described in the following chapters will adhere. Finally, we end with a brief outline of the evaluation strategy we use for the IR applications presented in the coming chapters. In Chapter 8 we will describe CrowdLogger, an open source system we have constructed to support these types of studies.

4.1 Local and global data

There are several reasons to use local user data in combination with globally aggregated data in a search application. First, let us define global data as the data included in a crowd log and local data as the data contained within a user log.

We have two motivations for using local data. First, it is extremely rich in detail and information compared with sanitized global data. If an application requires information that is not present in the global data due to sanitization, then local data may provide that information. Second, local data pertains specifically to the individual associated with the data. Thus, a user profile can be built from that data, providing models of a user's interests and habits. That is, it provides a way of personalizing an IR application. This information is not conveyed in globally aggregated data.

Global data also has its benefits. First, it provides a notion of popularity and general trends—a background model—that cannot be captured by a single individual's data. For example, when recommending queries to a user given they have just entered query q, it may be desirable to offer not only follow up queries the user has entered in the past for q, but also the popular reformulations of q across all users. Second, global data becomes most important when local data is sparse. For example, if a user has no search history, then the recommendations from the global model become the only recommendations.

4.2 Integrating local and global information

In this section, we present a common set of steps that implementers can specify when integrating local and global data into IR applications. This template allows us to describe how an applications collects, aggregates, and uses local data. The template, shown in Figure 4.1, is appropriate when using either sanitized or un-sanitized global data.

In following this template for an IR application, an implementer must describe how to perform these steps. For our purposes, much of the work in Step 1 is taken care of by the CrowdLogging framework described in Chapter 3, for example, extracting and aggregating artifacts, as well as sanitizing the data as a preprocessing step. Assuming we use CrowdLogging, we need only to specify what artifacts will be extracted in Step 1-a and how to preprocess the sanitized data in Step 1-b, if applicable. Examples of preprocessing include building a global model, such as a query flow graph, a trained machine learning model, or a maximum likelihood lookup table.

Local-global integration template

1. Collect and aggregate global data
    (a) Extract artifacts from user logs
    (b) Perform any necessary preprocessing of aggregated data
2. Perform any preprocessing of the local data
3. Process IR application input using local and global data
    (a) Gather required global data
        i. Client sends a request to the server
        ii. Server processes the client request
        iii. Server sends the response to the client
    (b) Integrate local and global data given application input
    (c) Return output

Figure 4.1. The list of procedures that must be defined for an IR application.

Similar to the global preprocessing, the local data preprocessing in Step 2 is only performed if applicable. Examples of preprocessing on local data include building language models of previously searched topics, identifying search tasks, or trained classifiers, to name a few.

The final step is more complex—it details how the IR application integrates local and global data to produce output. Given an input, IR applications must consider several elements, namely: how local data is accessed and used, how global data is accessed and used, and where computation occurs.

First consider these elements in the context of a non-privacy preserving, *centralized local-global integration model*, shown in Figure 4.2a. Under this model, a user submits a request consisting of the user's identification (x_i in the figure) and the application's input to the server. The IR application performs its computation on the server, accessing both the user's private information as well as the global information. The output of the application is then sent back to the client where it is displayed to the user. Note that his model does not preserve privacy—the server has access to all

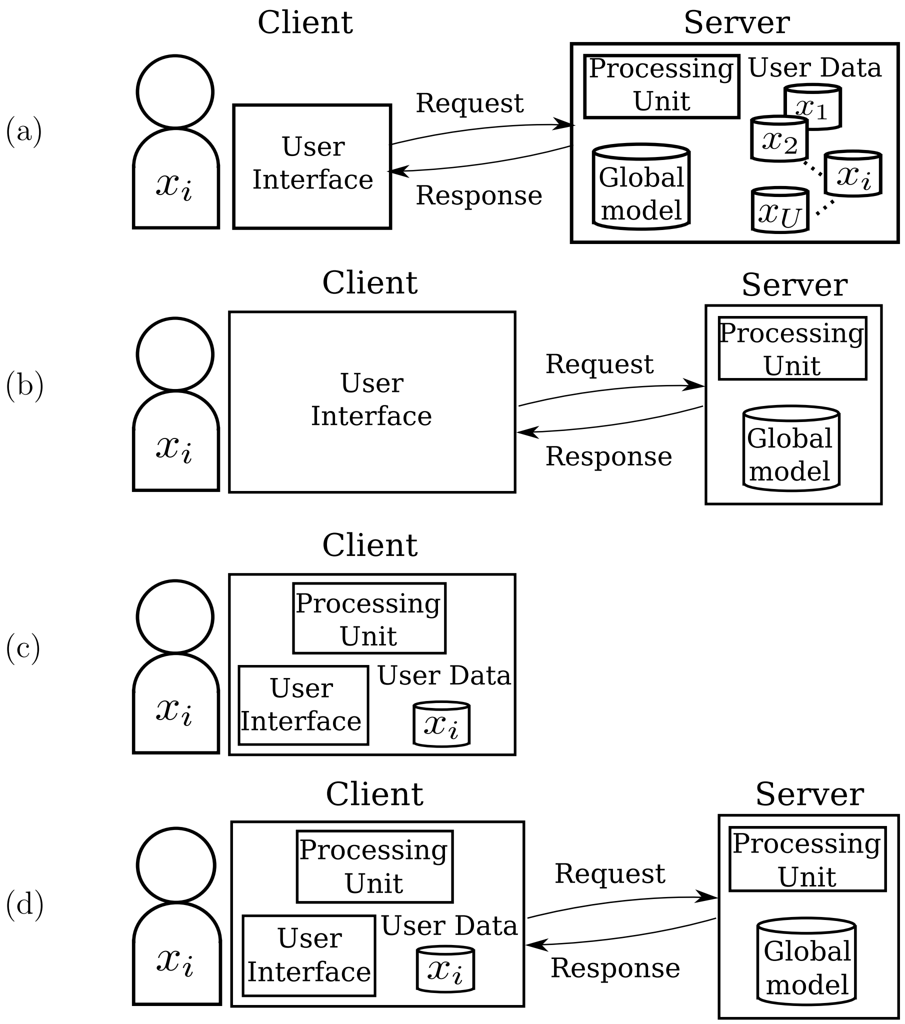


Figure 4.2. Figure (a) shows a *centralized integration model*, where all data is stored and processed on the centralized server. The client machine sends requests and displays the output from the server, e.g., through a web interface. Figure (b) shows a *global only model* where sanitized global data is stored on a server, but no local data is stored. Figure (c) shows a *local only model*, where local data is stored on a user's computer, but no global data is stored. Figure (d) shows the *partially distributed integration model*, where global data is sanitized and stored on a server and local data resides on the user's machine. Both the client and the server can perform computations for a given IR application.

users' data. However, it does provide an upper bound with which to compare privacy preserving models.

One model that is privacy preserving stores only sanitized global data on the server, as shown in Figure 4.2b. However, this model is not personalized because it does not make use of any local data. A different model that is both privacy preserving and personalized stores local data on the client side is shown in Figure 4.2c. This is private because only the user has access to the local data. While personalization is provided, no global data is used in this model, preventing the implementation of IR applications that require global data such as common query reformulations.

The model we propose is one where processing is spread across both the client and the server. We call this the *partially distributed local-global integration model*, shown in Figure 4.2d. It is considered *partially* distributed rather than *fully* distributed because some computation still occurs on the centralized server. The client machine hosts the user's personal data and the server hosts the sanitized global data. This protects user privacy and gives users the ability to tightly control their data.

Due to the potential size of the global data in a web setting, we assume it cannot be copied in full to a user's computer. Any computation that occurs on the user's machine has all of the local data at it disposal, however, this must be weighed against the limited computational resources available. Any computations that occur on the server have full access to the global data, but not the user's local information. Under this model, we assert that the three steps given an input, shown in Step 3 of Figure 4.1, must be specified for an application. Step 3-a-i. allows the implementer to specify what data is included in the request to the server. At the implementers discretion, and the consent of the user, local information can be sent with this request so that the server can process it.

To understand each step better, consider a few examples. In Step 3-a, we may have an IR application that requires a classifier trained on global data. Provided the classifier is trained on the server as a preprocessing step (i.e., it is not dependent on the client request), then the server's only responsibility is to pass the classifier to the client. In another example, consider the server building a graph of the global data—such as a query flow graph, where nodes are queries and directed edges represent that one query has been rewritten as another—in a preprocessing step. This graph is too large to download as a whole to the client machine, so instead the client can request a portion of the graph, e.g., all nodes and edges within three hops of the query q. Alternatively, the server could perform part of the application's processing rather than only fetch data. Using the graph example, the client could send the server a query and its context and have the server generate a list of query recommendations using the graph of global data.

This last example introduces an interesting tension: how much local information should be given to the server. If too much local information is provided, the partially distributed integration model reduces to the centralized integration model. Providing the associated search context along with a query may be too much information and violates privacy to some degree; there is no answer to this as it is a policy decision. In this work, we will consider applications that send single queries without additional local information.

To conclude this section, we present a list of questions in Figure 4.3 to which we provide answers for each IR application we discuss in the coming chapters. These questions map directly to the local-global data integration template shown in Figure 4.1.

4.3 Evaluating IR applications

In the following chapters, we will describe several IR applications that rely on global or local data. Our analyses will consider the affects of sanitizing the global data and incorporating personalization using local data.

IR application implementation questionnaire

▷ **Preprocessing**

1. What artifacts will be extracted?
2. What preprocessing will take place on the global data?
3. What preprocessing will take place on the local data?

▷ **On input**

4. What is the input?
5. What request will the client send to the server?
6. What processing will the server perform for the client request?
7. What response will the server send to the client?
8. How will the server data be integrated with the local data?
9. What is the output of the application?

Figure 4.3. The list of question that must be answered for each IR application.

The configurations will include global data produced under each of the privacy preservation mechanisms: FT_a, FT_u, DP_u, DP_a, and ZEALOUS. We will explore performance for various parameter settings. In addition, we consider the special cases where only global *or* local data is considered.

We will use several datasets: the 2006 AOL search log, the TREC 2010-11 Session Track data, and data we collected during a user study in 2009.

CHAPTER 5

SEARCH TASK IDENTIFICATION

In this chapter, we describe the problem of search task identification (STI), our approach to adding personalization, and an analysis of the effect of personalization and sanitization on STI performance.

Recall from Chapter 2, STI is the process of grouping a set of user searches into search tasks. With varying assumptions, this general process has also been referred to as *chain finding* (Boldi et al., 2008; Radlinski & Joachims, 2005), *query segmentation* (Boldi et al., 2008), and *task segmentation* (Jones & Klinkner, 2008). Identifying tasks is useful for many applications, such as establishing a focused context for use with context-aware tools, profiling long term interests, and aiding users in exploring their search histories.

One approach to STI that has been used in previous work breaks the problem into two parts: (1) classifying pairs of searches as belonging to the same task or not (Jones & Klinkner, 2008) and (2) clustering searches into tasks using the classification scores (Boldi et al., 2008; Lucchese et al., 2011). What defines two searches as sharing a common task is subjective and typically left up to the human assessors providing the same-task labels (Boldi et al., 2008; Jones & Klinkner, 2008; Lucchese et al., 2011; Radlinski & Joachims, 2005). Jones and Klinkner (2008) specify two levels of tasks: *search goals* and *search missions*. A search goal consists of one or more queries issued towards a common, atomic information need. A search mission is a set of related goals. Beyond these levels, however, no objective definitions are given.

Because of the subjectivity involved, we do not provide a strict definition of what it means for two searches to be a part of the same task.[1] Instead, we allow each user of the system to make that decision. For example, one user may decide that a series of searchers to find final scores from a set of sporting events held the previous night should be grouped by the same task, while another may further divide them into specific sports. Neither of these is wrong, just different, and reflect each individual's perspective.

With this to-each-his-own definition of what a search task is, it is natural to ask two questions: how different are user's definitions and how can personalization be incorporated into STI. In addition, we would like to know how sanitization affects the performance of STI models, including those that integrate global and local information. Formally, we have the following questions:

RQ1. How different are users' perceptions of what a search task is?

RQ2. How does personalization affect the quality of STI?

RQ3. What impact does sanitizing global data have on the quality of STI?

To address these questions, we consider a large set of labeled AOL data. We consider the inter-annotator agreement between same-task labels for a subset of the labeled data—an analysis that no other work has reported to our knowledge. We consider a number of existing models for STI and consider the ways they can be used in a personalized setting. We then explore sanitization by using the FT_a, FT_u, DP_a, DP_u, and ZEALOUS privacy mechanisms to produce global models.

Our key contributions are: (1) results using a collection of labeled data consisting of the entire three-month histories of 503 AOL users; (2) an analysis of agreement among the seven annotators over labels provided for ten AOL users, demonstrating that annotators (and therefore users) have different internal views of what constitutes

[1] This is very similar to the definition of *relevance* in information retrieval.

a search task; (3) random forest classification models consisting of novel features that significantly out-perform the current state of the art; (4) several local and local-global integrated models that statistically significantly out-perform global-only models; and (5) an analysis of the effects of sanitization, in which we observe that the best performing model trained using a single same-task annotation from each of 352 users can obtain similar performance to using all annotations.

Figure 5.1 outlines the implementation we will follow for personalized STI throughout this chapter. In Section 5.1 we describe how global data will be mined, collected and processed. We then describe how local data is used and combined with global data in Sections 5.2 and 5.3. We present our experimental setup and the data we use in Section 5.4. We present our findings in Section 5.5 before wrapping up with a discussion of limitations in Section 5.6 and a summary in Section 5.7.

5.1 Global data

As mentioned, we consider STI to consist of two subproblems: same-task classification and task clustering. In this section, we describe how global data is mined and aggregated to form global same-task classification models, including the effect of multiple artifact representations, and how clustering is performed when no data is provided by the user.

5.1.1 Same-task classification

Three recent approaches to classifying two searches as belonging to the same task are the following. (1) Jones and Klinkner (2008) trained a logistic regression model between all pairs of searches in a session. (2) Boldi et al. (2008) created two classifiers: a logistic regression model for use with query pairs that only occurred once in the training data (this covers roughly 50% of their training data) and a rule-based classifier for the other instances. In the case of the logistic classifiers used by these

Personalized search task identification

▷ **Preprocessing**

1. *What artifacts will be extracted?*

   ◇ Feature vectors between pairs of queries labeled as belonging to the same-task or not (see Section 5.1 for details).

2. *What preprocessing will take place on the global data?*

   ◇ A random forest model will be trained on the sanitized feature vectors.

3. *What preprocessing will take place on the local data?*

   ◇ A user-specific random forest model will be trained for each user with labeled data.

▷ **On input**

4. *What is the input?*

   ◇ A set of searches.

5. *What request will the client send to the server?*

   ◇ N/A (the global model will be downloaded from the server as a preprocessing step)

6. *What processing will the server perform for the client request?*

   ◇ N/A

7. *What response will the server send to the client?*

   ◇ N/A

8. *How will the server data be integrated with the local data?*

   ◇ **Same-task classification.** Same-task labels will be predicted for all unlabeled query pairs using both the local (if available) and global classifiers.

   ◇ **Task clustering.** Queries will be grouped by means of single-link clustering using thresholded same-task classification scores as edge weights; if local data is available, predicted edges that violate the set of must-link and cannot-link constraints are ignored, prevent clusters from being merged that contain queries we know should be kept separate (see Section 5.2 for details).

9. *What is the output of the tool?*

   ◇ The input searches, grouped by search task.

Figure 5.1. Implementation specifications for personalized search task identification.

Feature	Description	Lucchese	Random Forest
time-diff	Time difference in seconds.		✓
jaccard	The Jaccard Coefficient.	✓	✓
levenshtein	One minus the Levenshtein string edit distance.	✓	✓
overlap	Jaccard coefficient of character trigrams.		✓
semantic-wikipedia	Cosine similarity of Wikipeida pages, weighted by tf-idf for each query.	✓	✓
semantic-category	Like semantic-wikipedia, but each page is mapped to its category (scores for duplicate categories are summed).		✓
semantic-wiktionary	Like semantic-wikipedia, but over Wiktionary.	✓	✓

Table 5.1. Features between two queries used in the models.

two groups, the important features are generally the time between two queries and various syntactic features between the query text, such as the Jaccard coefficient. Boldi et al. did not specify any details of the rule-based classifier, so we have no way to replicate their two-classifier approach. (3) The state of the art classification model was introduced by Lucchese et al. (2011). They built an unsupervised classifier[2] that integrates both the syntactic and semantic distance between two searches' query text. They directly compared their method to Boldi et al., finding their model performed better.

We consider two models: the one introduced by Lucchese et al. (we will refer to this as the Lucchese model) and a random forest model using the features used by the Lucchese model and two additional ones: time-diff and semantic-category. All the features used are listed in Table 5.1, as well as which models use them. We added time-diff since previous work has shown time to be useful (Boldi et al., 2008; Jones &

[2] It does, however, contain a parameter that can be tuned.

Klinkner, 2008), though both Jones and Klinkner and Lucchese et al. demonstrated that temporal features alone are not sufficient. We introduce the semantic-category to be a generalization of the semantic-wikipedia feature used by Lucchese et al.. The intuition is that some semantically similar terms may not occur in the same document together, but rather in separate documents within the same general category. Taken together, these features touch on the temporal (time-diff), lexical (jaccard, levenshtein, and overlap), and semantic (semantic-wikipedia, semantic-category, and semantic-wiktionary) similarity spaces between searches.

We chose a fully supervised machine learning classifier as our second model (as opposed to the ad-hoc Lucchese model) since it empirically learns the relationships between features. The choice of a random forest classifier was based on preliminary experiments with several off-the-shelf classifiers, including: decision trees, support vector machines, and logistic regression classifiers. Random forest models performed consistently well and they have a interesting property that made them stick out: rather than learning one set of weights for features, they instead create several sub-classifiers (decision trees), each considering a random sample of features in isolation and learning a set of weights for them, as well as a meta-classifier that combines the predictions from each of the sub-classifiers. Ho (1995) demonstrated that a set of decision trees trained using random subsets of features are able to generalize in complimentary ways, and that their combined performance is greater then any one by itself.

In order to train global versions of the Lucchese and random forest models, we rely on feature vector artifacts aggregated into a crowd log. But first, we must obtain annotations from the users contributing the artifacts. We ask users to group some subset of their searches into search tasks. Once grouped, we extract the features for each pair of queries in the annotated set, applying the label *same-task* or *different-task* based on whether the user grouped the two into the same search task. We know from

Bin1:
Time difference (seconds: $[0, \infty]$**)**

$$\text{bin1}(x) = \begin{cases} 0.00 & x < 30\text{seconds}, \\ 0.25 & x < 1\text{minute}, \\ 0.50 & x < 1\text{hour}, \\ 1.00 & x \geq 1\text{hour}. \end{cases}$$

Lexical and semantic features ($[0, 1]$)

$$\text{bin1}(x) = \begin{cases} 0.00 & x < 0.025, \\ 0.25 & 0.025 \leq x \leq 0.250, \\ 0.50 & 0.250 < x < 0.750, \\ 0.75 & 0.750 \leq x \leq 0.975, \\ 1.00 & x > 0.975. \end{cases}$$

Figure 5.2. The first of two binning methods we explored (bin1).

Section 3.3 that artifact representation has a significant impact on supportability. With up to seven unconstrained features, every feature vector has the potential to be unique.

Rather than use continuous feature values, we discretize them into a small number of bins. We considered two methods for binning. The first (bin1) maps each of the features with ranges [0,1] (all features but time-diff) to one of the values {0.0,0.25,0.5,0.75,1.0}, and time-diff into {0.0,0.25,0.5,1.0}. The mapping is shown in Figure 5.2. The second (bin2) binarizes the values: values for all features are either 1 or 0. The mapping is shown in Figure 5.3. Using a large sample of labeled data (see Section 5.4.2 for details about the full dataset), we found, not surprisingly, that bin2 releases more distinct artifacts and artifact impression than the other methods for a given value of k. Figure 5.4 shows the coverage of using the original feature vectors (truncated to three decimal places) and those produced using the two binning methods over the training data used in the experiments we describe in Section 5.4. It

Bin2:
Time difference (seconds: $[0, \infty]$**)**

$$\text{bin2}(x) = \begin{cases} 0 & x < 1\text{minute}, \\ 1 & x \geq 1\text{minute}. \end{cases}$$

Lexical and semantic features ($[0, 1]$)

$$\text{bin2}(x) = \begin{cases} 0 & x < 0.5, \\ 1 & x \geq 0.5. \end{cases}$$

Figure 5.3. The second of two binning methods we explored (bin2). This is the method we used for our final experiments.

is clear that the original, unconstrained feature vectors are not suitable for privacy-aware data aggregation. Based on the increased coverage possible when using bin2, and the performance gains we found when applying privacy mechanisms to bin2 feature vectors on a development data set, we only consider bin2 for the remainder of this chapter.

5.1.2 Task clustering

Any clustering technique can be used to group searches into tasks based on the same-task classification scores. We use the weighted connected components (WCC) method described by Lucchese et al., who found this method outperforms several other techniques, including the query chaining approach used by Boldi et al. The WCC algorithm creates a fully connected graph where each search in a given session is represented as a node and the edges are the same-task classification scores between searches. The graph is then pruned of any edges with a weight below a threshold η and each connected component in the pruned graph is considered a task.

An example of this process is shown for a sample search history in Figure 5.6 (the ground truth is displayed in Figure 5.5).

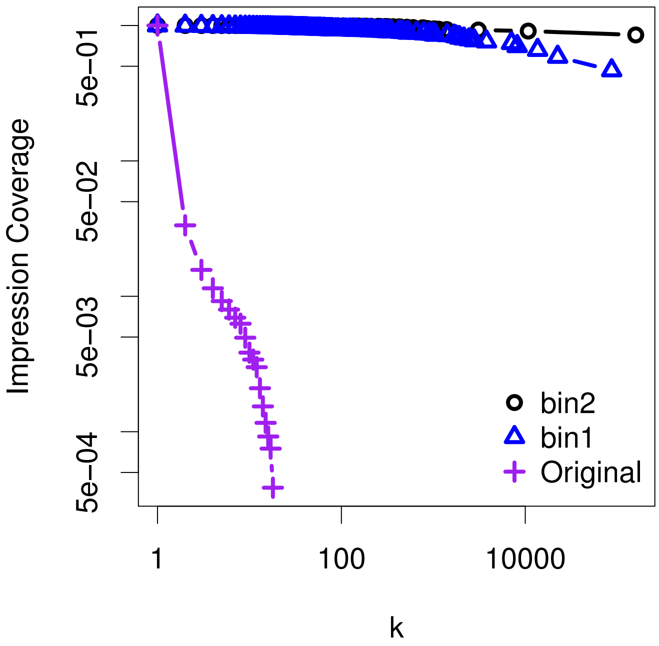


Figure 5.4. The impression coverage of the original feature vectors (up to three decimal places) and feature vectors binned with the bin1 and bin2 methods as the impression threshold increases (FT_a). Both axes are shown in log scale.

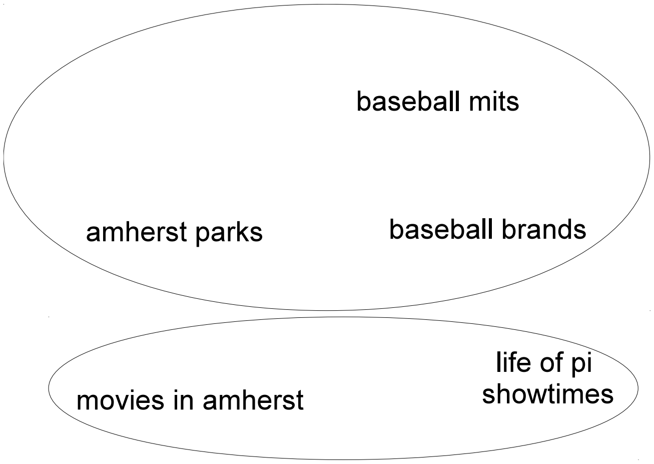


Figure 5.5. An example search history with two search tasks.

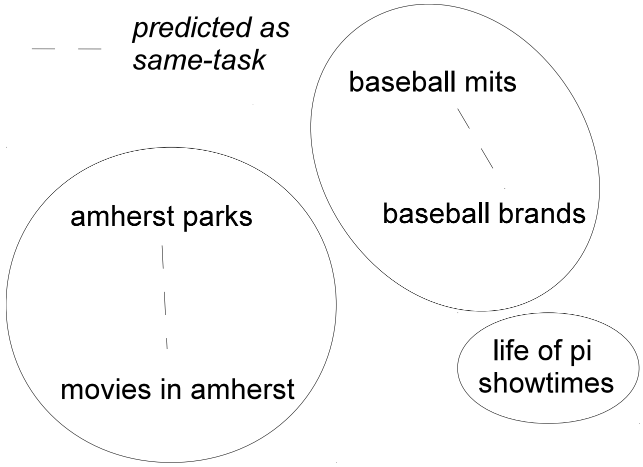


Figure 5.6. An example search history with automatically generated clusters using weighted connected components clustering and predictions from a globally trained model.

5.2 Local data

Provided a user has manually grouped a subset of their searches into search tasks, we can use that data to train local models. We consider locally trained equivalents of the two global models: Lucchese and a random forest classifier. Unlike with global models, where we need to consider sanitization, local models do not need to be binned, though doing so is perfectly acceptable. We found that the unbinned features work better than binned for local data, so we will only consider that case.

When local data is available, we use a more advanced form of clustering. User annotations provide a set of constraints: the label for every pair of annotated queries is either must-link or cannot-link. We being clustering by first inserting the labeled edges. We then classify all missing edges, sort them in non-ascending order by weight, and remove all edges with an edge weight less then a threshold η. For the remaining edges, we insert them one at a time; any time an edge e causes two clusters to merge, we first check to ensure that no cannot-link edges span the two clusters. If such an edge exists, then e is removed, the two clusters are not merged, and we move on to the next edge.

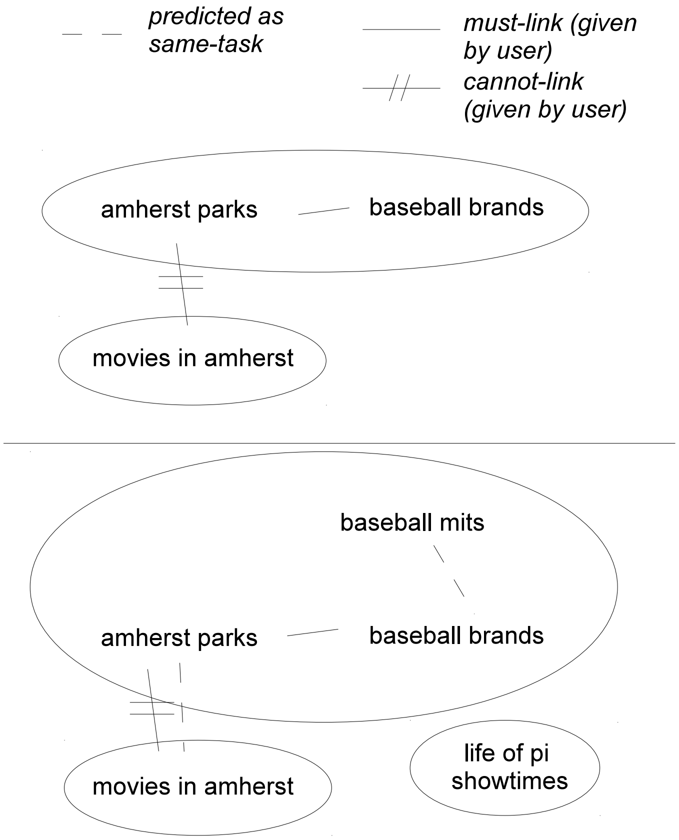


Figure 5.7. An example search history with automatically generated clusters using a set of must-link and cannot-link edges provided by the user's annotations (derived from the user clustering at the top) and predictions for the remaining edges (on the bottom).

An example of the constrained clustering used when local data is available is shown in Figure 5.7. Note how the cannot-link edge between *amherst parks* and *movies in amherst* overrides the prediction that the two should be clustered together.

5.3 Integrating local and global data

We consider several variations of local and global data integration. The simplest uses the global classification scores for same-task prediction, but use local data in the clustering phase, as described in Section 5.2. We refer to this method of integration as global score, local clustering (GScoreLCluster).

We also consider several ways of combining the predictions from global and local classifiers: minimum, maximum, product, and mean. In experiments on a development set, we found the mean integration to perform consistently well, usually out-performing the global- or local-only models, while the other score integration techniques typically did not. For the remainder of this chapter, we will only consider the mean, which we refer to as local-global score mean (LGMean).

5.4 Experimental setup

In this section, we describe several experiments to answer the three research questions we posed at the beginning of this chapter. Afterwards, we discuss the data set used for the experiments.

5.4.1 Experiments

Experiment 1. The first of our research questions is aimed at quantifying the differences and users' perceptions of search tasks. To test this, we turn to inter-annotator agreement—a measurement of how closely a set of annotators' responses align with each other. Specifically, we ask a set of A annotators to group searches into search tasks for U separate search histories. We decompose each grouping into labeled query pairs: *same-task* or *different-task*. Agreement is measured on the labeled query pairs. If two annotators form all the same groups, with the exception of one query, this may result in many differences among their labeled pairs, and lower their agreement. If agreement is high, then we can say that users share a similar perception of search tasks. If we find some degree of disagreement, then we can assume that users' perceptions differ with respect to search tasks. We measure agreement using Fleiss' Kappa (Fleiss, 1971), which is a statistical measure of agreement between more than two annotators supplying categorical values.

Experiment 2. Our second research questions asks how personalization affects the quality of STI. To evaluate this, we use the following setup. First, we reserve a large set of data for training the global Lucchese and random forest classifiers. We then use several sets of user histories for local training and testing. No data from users in this second set is contained in the first set. For each user history in the second set, we split the data into three sections: local training, testing, and other. The testing partition is fixed as the last 30% of the user's history. We will vary the local training partition, using labels between n randomly selected queries from the non-test partitions. Whatever is not used for local training or testing is considered part of the other partition (this segment must still be labeled in order for clustering to work). For each value of n, we conduct R randomizations. We consider three local models using Lucchese and random forests: local only, GScoreLCluster, and LGMean. As baselines, we use the global only classifiers and a dumb classifier that always predicts that two queries are from different search tasks. We evaluate success using macro query pair label accuracy over users. This is a reasonable measure because it gives equal weight to each user and does not favor either the *same-task* or *different-task* labels—clustering two cannot-link searches separately is just as important as clustering two must-link searches together. If personalization is helpful, then we should find that the local-only or integrated models out-perform their global counterparts.

Experiment 3. Our third and final experiment asks how sanitizing global data affects the performances of global models trained on that data. To answer this, we use the same global set used in Experiment 2 and apply the FT_a, FT_u, DP_a, DP_u, and ZEALOUS privacy mechanisms on that data. For the frequency thresholding mechanisms, we use $k = \{2, 5, 10, 20, 50, 100\}$ and for the differentially private mechanisms, we use $d = \{1, 2, 4, 8, 16, 32\}$. We re-run the global and integrated models used in Experiment 2 and evaluate in the same manner. The key areas of interest are how the global and integrated models perform relative to 1) using no sanitization, 2) using other sanitization methods, and 3) the local-only models, which are invariant to global sanitization.

5.4.2 Data

Previous research in STI has relied on search task annotations of subsets of user search histories extracted from search logs. The annotations are provided by third parties, e.g., other researchers. Both Jones and Klinkner and Boldi et al. used proprietary data available only to Yahoo! researchers, while Lucchese et al. labeled data for the first week of thirteen "power users" in the 2006 AOL search log (comprising 307 sessions, 1,424 queries, and 554 tasks). These previous works considered relatively small slices of user histories—testing on a few days up to a week of data. While Lucchese et al. made their data available, we decided that a larger sample of users spanning a longer time frame would provide more sound results.

To that end, we formed two sets of data using the 2006 AOL search log: one for measuring inter-annotator agreement and another for training and testing STI approaches. First, let us describe the AOL data set. It spans queries and the domains of clicked results for 657,426 users over March 1–May 31, 2006. A total of 36,389,567 entries are contained in the data set, of which 22,125,550 are queries, 21,011,340 are new queries (adjacent duplicates ignored), 10,154,742 are distinct queries, 7,887,022 are

	All	**March**	**April**	**May**
Queries	22,125,550	8,575,191	7,377,635	6,172,724
Avg. queries/user	33.65	16.49	15.15	15.24
Med. queries/user	12.00	7.00	6.00	6.00
Max queries/user	212200.00	1793.00	212200.00	8218.00
1st Quartile	4.00	3.00	2.00	2.00
3rd Quartile	34.00	18.00	15.00	16.00
IQR	30.00	15.00	13.00	14.00
Upper fence	79.00	40.50	34.50	37.00
Sessions	10,998,729	4,291,816	3,005,875	3,701,038
Avg. sessions/user	16.73	8.26	7.38	7.64
Med. sessions/user	7.00	4.00	4.00	4.00
Max sessions/user	888.00	320.00	384.00	335.00
1st Quartile	3.00	2.00	1.00	2.00
3rd Quartile	19.00	10.00	8.00	9.00

Table 5.2. Queries-per-user and sessions-per-user statistics across the three months of the 2006 AOL search log. This considers all query instances; i.e., duplicate queries from the same user are not ignored.

next-page actions, and 19,442,629 are result clicks (so called click-throughs). Query and session statistics are shown in Table 5.2. Note that we use 26 minutes as the session timeout, per the findings of Lucchese et al. (2011).

For evaluating STI algorithms, we used the following sampling technique. We first decided to consider only users with between 12–79 queries. This corresponds to the median and upper fence (Q3 + 1.5·IQR, where Q3 is the third quartile and IQR is the inter-quartile range. This segment of users provides us with sample that submitted enough queries to be interesting, but who are not outliers, as are the users labeled by Lucchese et al. We then sampled 1,000 users and asked ten annotator (some overlapping with the group that provided annotations for the inter-annotator agreement set) to form search tasks for each of the users. We collected annotations for a total of 503 users, each labeled by one annotator. Statistics of the data set are described in Table 5.3. We further broke the set into folds by annotator (see Table 5.4), motivated by the possibility that each annotator has his or her own unique

Annotators	10	
Users	503	
Sessions	8,740	
Queries	15,779	
Tasks	6,448	
Interleaved tasks	3,134	
Avg. queries/task	2.4	
Av. duration/task (minutes)	8	
Labeled pairs	310,287	
same-task	40,940	(13%)
different-task	269,347	(84%)

Table 5.3. Statistics of the data set used to to evaluate STI algorithms.

Annotator	**Annotated users**	**Train/development or Test**
1	50	train/dev
2	50	train/dev
3	50	train/dev
4	3	train/dev
5	100	train/dev
6	50	train/dev
7	50	train/dev
8	33	test
9	97	test
10	21	test

Table 5.4. Statistics the annotator folds and training/testing sets.

perception of a search task. We randomly selected three annotators and restricted their data to the final evaluation set. Data from the other seven annotators was designated for training and development. Thus, we cannot learn latent relationships specific to the annotators in the test set using the training set. Statistics about the training/development and test sets are listed in Table 5.5.

For measuring inter-annotator agreement, we used a different sampling technique. We considered only users roughly in the middle 50% relative to session counts. That is, users with between 4–19 sessions. We chose a lower bound of four rather than three (the first quartile), because we felt that three was too few. Our motivation

	Train/dev	**Test**
Annotators	7	3
AOL users	352	151
Queries	21,697	8,858
Tasks	9,122	3,774
Labeled pairs	224,057	86,506
same-task	28,747 (13%)	12,252 (14%)
different-task	195,309 (87%)	74,254 (86%)

Table 5.5. Statistics of the train and test sets.

Annotators	6
AOL users	10
Sessions	92
Queries	131
Avg. Tasks/annotator	80
Labeled pairs	939

Table 5.6. Statistics of the data set used to measure inter-annotator agreement.

for using sessions counts rather than query counts was based in the belief that more sessions would correspond to a greater number of tasks, thus providing an interesting data set for measuring inter-annotator agreement. From this dataset, we randomly sampled ten users' data. We asked a group of six graduate students and professors to annotate the data from the ten AOL users by grouping searches into tasks with shared topics or search goals. Summary statistics are shown in Table 5.6.

5.5 Results and analysis

Experiment 1. Considering the set of pair-wise annotations provided for queries from ten AOL users, we found a Fliess inter-annotator agreement of $\kappa = 0.78$. We consider this high agreement. However, most of this agreement comes from the majority of queries being placed in different search tasks, which is rather uninteresting—one would expect the majority of queries to be placed in different search tasks regardless of an individual's perception of a search task.

When we ignore instances where everyone agrees that two queries are from different tasks, the agreement drops to $\kappa = 0.53$. Said differently, for all the annotations where at least one annotator provided a same-task label (there were 196 such labels out of a total of 939), agreement is moderate. At least one disagreement was found for 118 of the labels. One pair of searches for which annotators disagreed was "zip codes" and "driving directions", issued sequentially by the corresponding AOL user. Two of the six annotators placed this pair of searches in the same task.

Experiment 2. For personalized models—those using only local data or an integration of global and local data—we found that personalization is helpful. A plot of performance is shown at the top of Figure 5.8 and a table with the numbers is shown at the top of Table 5.7 when all local training data is used (70% of a user's data). Every model was run with 50 randomly sampled local data subsets; the mean is plotted with 95% confidence intervals shown in dotted lines (these are tight bounds in most cases, and are therefore covered by the line representing the mean). The global and integrated random forest models (LGAvgRF, GScoreLClustRF and GlobalRF) performed best. The two integrated models performed significantly ($p < 0.001$; two-sided t-test) though not substantially better than the GlobalRF model. The differences are in the thousandths place: 0.945 vs. 0.942 vs. 0.940. The Lucchese models were not far behind. The local-only random forest model (LocalRF) was the poorest performing model outside of the DiffTask baseline. As a general trend, smaller local training set sizes (with fewer than 10 or 15 searches) do not seem to generate enough data for the local-only or LGAvg models to outperform their global or GScoreLClust counterparts. Our conclusions are that personalization helps, but it is not required to achieve high performance. If a user has labeled no or a small number of queries, then GScoreLClustRF is the best choice; if more local data is available for training, then LGAvgRF is appropriate.

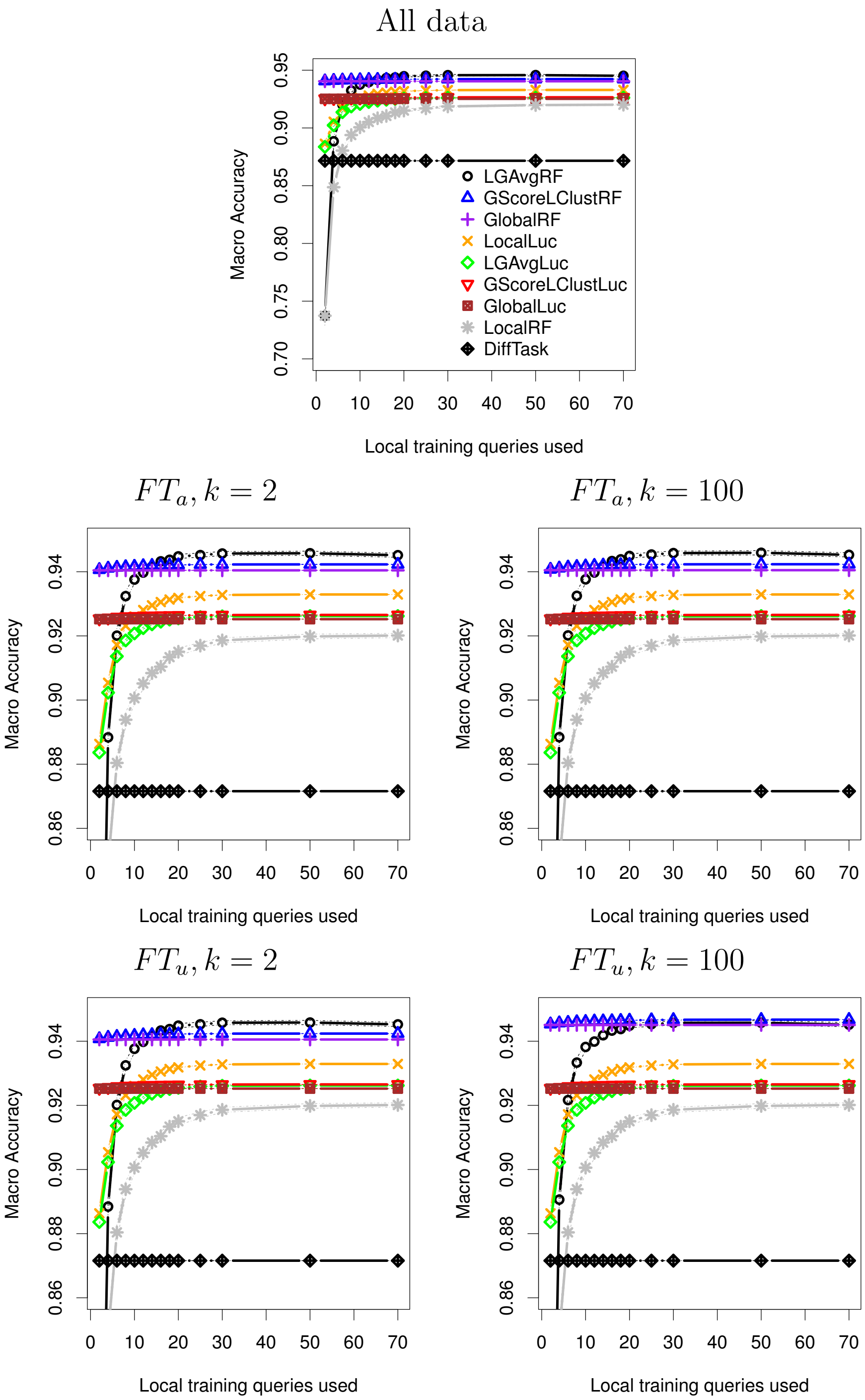


Figure 5.8. The performance when global data is not sanitized (top) and for the extreme settings of k for the two frequency thresholding privacy mechanisms we considered. The random forest models are denoted by RF, while the Lucchese models are denoted by Luc.

Experiment 3. In the third experiment, we considered sanitizing global data. The effects of sanitization on the performance of global and integrated models is shown in part in Figures 5.8 (for the frequency thresholding mechanisms) and 5.9 (for the differential privacy mechanisms). We only show plots of the extreme parameters we explored, i.e., $k = \{2, 100\}$ for frequency thresholding and $d = \{1, 32\}$ for the differential privacy mechanisms. We report the macro accuracy when all local training data (70% of a user's data) is available for the integrated model in Table 5.7. Using FT_a for sanitization does not affect performance. FT_u with $k = 2$ resembles FT_a, but when k is increased to above 20, we find that the performance of the GlobalRF and GScoreLClustRF goes up to the highest of any models under any conditions. A possible explanation for this is that FT_u at $k = 100$ is revealing the most general (across users) and frequent feature vectors, preventing the classifiers from over fitting the training data.

For the differentially private mechanisms, we used $U = 500$, $\delta = 1/500$, and $\epsilon = \lg(10)$. In addition, since they all require sampling artifacts from users, we resampled global data in parallel with the random sampling for local training data. We generally saw drops in the performance of global and integrated models. The various mechanisms behave very differently, however. For example, DP_a is relatively stable between the extremes of $d = 1$ and $d = 32$. DP_u shows similar performance to DP_a at $d = 1$, though with greater variance (the confidence intervals are wider than under other conditions). When d is increased to 32, however, performance plummets: models trained with global data are not much better than the DiffTask baseline. The reason for this difference in performance is evident from the feature vector impression and distinct coverage, shown in Figure 5.10. While impression coverage is high for FT_u at $d = 32$ distinct coverage is almost zero, preventing useful learning.

The ZEALOUS mechanism exhibits the opposite behavior—it shows very poor performance for $d = 1$, but very reasonable performance at $d = 32$. While the distinct coverage under ZEALOUS does decrease with higher levels of d, it seems that at $d = 32$, it manages to release the distinct vectors that help classifiers discriminate between the same- and different-task classes.

Based on these results, we recommend the FT_u mechanism with $k = 100$ for implementers and users comfortable with frequency thresholding. For differential privacy, DP_a and DP_u should both be used with lower values of d, whereas ZEALOUS should use higher levels of d (though higher levels may be detrimental as distinct coverage will continue to fall).

5.6 Limitations

One drawback of this study is in its reliance on third-party annotators. While this is common in the STI literature, it would be useful to conduct a user study into how users group their own searches into tasks.

Another limitation is that we only consider the case when users in the training set annotate all of their tasks, not just a handful. An interesting analysis would be the effect of users only contributing annotations for a handful of queries.

Finally, we examined personalized STI techniques that used annotations for a random sample of local user queries. A fuller personalized experience may achievable through the use of, e.g., active learning, where users are asked to annotate query pairs for which an algorithm has low confidence in its prediction.

5.7 Summary

In this chapter, we explored search task identification (STI). We measured the degree to which a set of annotators agree on a search task clustering, finding that users perceive tasks differently—we found an agreement of $\kappa = 0.53$ when at least one of six annotators labeled a pair of searches as belonging to the same task. This is the first such experiment that we know of in the STI literature.

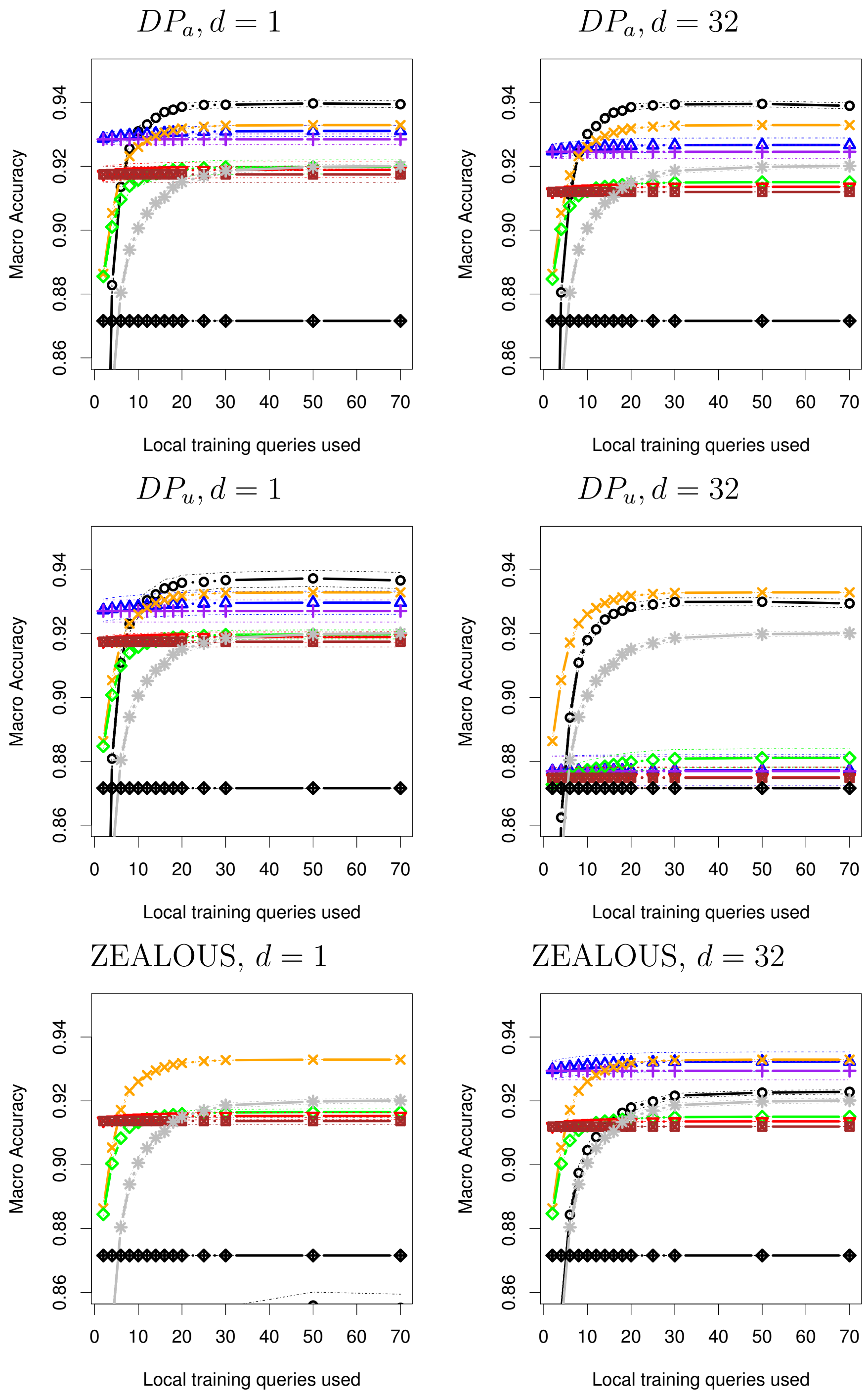


Figure 5.9. The performance when for the extreme settings of d for the three differential privacy mechanisms we considered. The random forest models are denoted by RF, while the Lucchese models are denoted by Luc.

	All Data
LGAvgRF	0.945
GScoreLClustRF	0.942
GlobalRF	0.940
LocalLuc	0.933
LGAvgLuc	0.926
GScoreLClustLuc	0.927
GlobalLuc	0.925
LocalRF	0.920
DiffTask	0.872

	$FT_{\mathbf{a}}$					
	k2	**k5**	**k10**	**k20**	**k50**	**k100**
LGAvgRF	0.945	0.945	0.945	0.945	0.945	0.945
GScoreLClustRF	0.942	0.942	0.942	0.942	0.942	0.942
GlobalRF	0.940	0.941	0.941	0.940	0.940	0.940
LGAvgLuc	0.926	0.926	0.926	0.926	0.926	0.926
GScoreLClustLuc	0.927	0.927	0.927	0.927	0.927	0.927
GlobalLuc	0.925	0.925	0.925	0.925	0.925	0.925

	$FT_{\mathbf{u}}$					
	k2	**k5**	**k10**	**k20**	**k50**	**k100**
LGAvgRF	0.945	0.945	0.945	0.945	0.945	0.945
GScoreLClustRF	0.942	0.942	0.942	0.947	0.947	0.947
GlobalRF	0.941	0.941	0.941	0.945	0.945	0.945
LGAvgLuc	0.926	0.926	0.926	0.926	0.926	0.926
GScoreLClustLuc	0.927	0.927	0.927	0.927	0.927	0.927
GlobalLuc	0.925	0.925	0.925	0.925	0.925	0.925

	$DP_{\mathbf{a}}$					
	d1	**d2**	**d4**	**d8**	**d16**	**d32**
LGAvgRF	0.939	0.938	0.938	0.939	0.938	0.939
GScoreLClustRF	0.931	0.929	0.931	0.927	0.925	0.927
GlobalRF	0.928	0.926	0.928	0.925	0.923	0.925
LGAvgLuc	0.920	0.919	0.917	0.915	0.915	0.915
GScoreLClustLuc	0.919	0.918	0.916	0.914	0.914	0.914
GlobalLuc	0.917	0.917	0.915	0.912	0.912	0.912

	$DP_{\mathbf{u}}$					
	d1	**d2**	**d4**	**d8**	**d16**	**d32**
LGAvgRF	0.937	0.938	0.939	0.939	0.939	0.929
GScoreLClustRF	0.930	0.931	0.932	0.927	0.924	0.877
GlobalRF	0.927	0.928	0.929	0.925	0.922	0.877
LGAvgLuc	0.920	0.919	0.918	0.916	0.915	0.881
GScoreLClustLuc	0.919	0.918	0.917	0.914	0.914	0.875
GlobalLuc	0.917	0.917	0.915	0.913	0.912	0.875

	ZEALOUS					
	d1	**d2**	**d4**	**d8**	**d16**	**d32**
LGAvgRF	0.855	0.881	0.908	0.914	0.923	0.923
GScoreLClustRF	0.439	0.460	0.464	0.464	0.940	0.932
GlobalRF	0.129	0.129	0.129	0.129	0.936	0.929
LGAvgLuc	0.917	0.926	0.926	0.926	0.926	0.915
GScoreLClustLuc	0.915	0.927	0.927	0.927	0.927	0.914
GlobalLuc	0.914	0.925	0.925	0.925	0.925	0.912

Table 5.7. Macro accuracy over users. The random forest models are denoted by RF, while the Lucchese models are denoted by Luc.

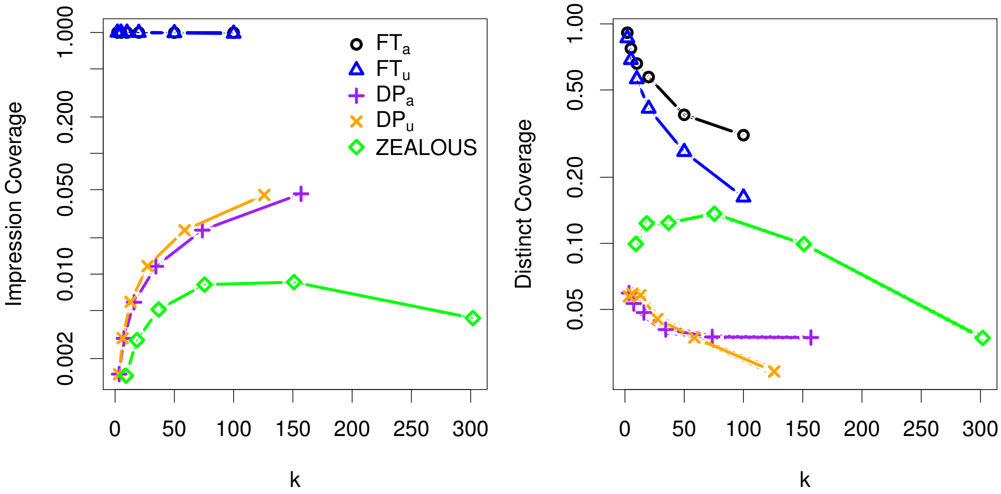


Figure 5.10. Impression (left) and distinct (right) coverage of feature vectors from the training/development set consisting of all seven features described in Table 5.1, discretized with bin2, along with the same-task label. The coverage using only the subset of features required by the Lucchese classifier is very similar.

We presented six personalized models, three based on the current state-of-the-art STI system by Lucchese et al. (2011), and three using a random forest classifier. We found the random forest models to perform best with respect to macro accuracy, and that personalization using both local and global information produces statistically significant gains when users annotate ten or more queries. However, a global random forest model does very well on its own. We also measured the effects of sanitization on STI performance, finding that the effects are minimal, though both ZEALOUS and DP_u exhibited poor performance with certain values of d. In fact, with FT_u at $k = 100$, we actually saw improvements in performance.

CHAPTER 6

TASK-AWARE QUERY RECOMMENDATION

Query recommendation is a common tool used by search engines to assist users in reformulating queries. When a search task requires multiple searches, the sequence of queries form a context around which new queries can be recommended. This context constitutes local data and utilizing it for recommendation provides a personalized experience. Figure 6.1 illustrates a series of queries issued by a user consisting of two search tasks: 1) finding information about the history of black powder firearms and 2) preparing for the GMAT standardized test. Given this sequence, our goal is to generate a list of query suggestions with respect to the most recently submitted query, or *reference query*, which is "black powder inventor" in this example. Notice, however, that the user has interleaved the two tasks such that no two adjacent queries are part of the same task. If we use the entire context to generate recommendations, two of the queries will be *off-task* with respect to the reference query and three (including the reference query) will be *on-task*. This chapter explores the effects that on- and off-task contexts have on query recommendation. We also consider the effects that sanitization of query reformulations has on recommendation quality. While previous work has considered task-aware query recommendation over logged user data, we are not aware of any work that has systematically explored the effects of on-task, off-task, and mixed contexts or sanitization on recommendation performance.

Though the example in Figure 6.1 may seem an extreme case, consider that Lucchese et al. (2011) found 74% of web queries were part of multi-tasking search sessions in a three-month sample of AOL search logs; Jones and Klinkner (2008) ob-

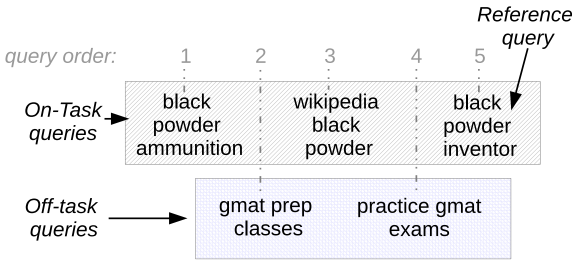


Figure 6.1. A search context with interleaved tasks.

served that 17% of tasks were interleaved in a 3-day sample of Yahoo! web searches; and Liao et al. (2012) found 30% of sessions contained multiple tasks and 5% of sessions contained interleaved tasks in a sample of half a billion sessions extracted from Bing search logs. In addition, in a labeled sample of 503 AOL user search histories spanning three months, we found 57% of search tasks consisted of two or more queries (see Figure 6.2), but there was only a 45% chance that any two adjacent queries were part of the same task (see Figure 6.3). Figure 6.3 shows the likelihood of seeing n tasks in any sequence of x queries, e.g., 10-query sequences typically consist of 3–7 search tasks. This means that a context consisting of the most recent n queries is very likely to consist of sub-contexts for several disjoint tasks, none of which may be a part of the same task as the reference query.

The goal of this chapter is to better understand the effects of on-task, off-task, and mixed contexts as well as privacy on query recommendation quality. We also present and analyze several methods for handling mixed contexts. We address five questions concerning query recommendation:[1]

RQ1. How does on-task context affect query recommendation performance?

RQ2. How does off-task context affect query recommendation performance?

[1] Experiments and analysis for the first four questions were presented in a SIGIR 2013 paper (Feild & Allan, 2013).

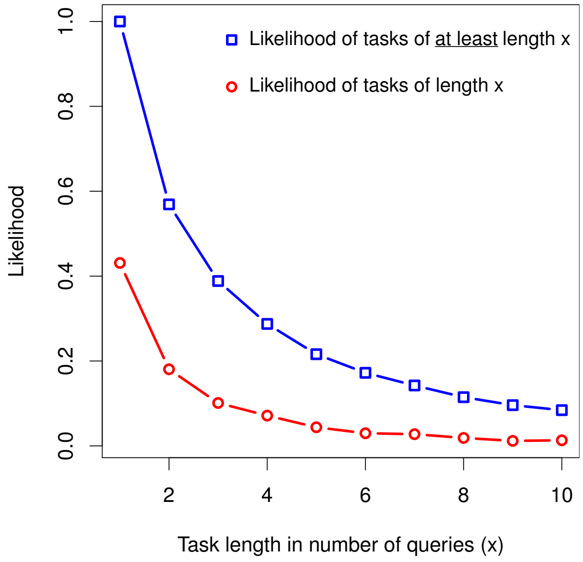


Figure 6.2. Distribution of tasks lengths observed in a labeled sample of the AOL query log.

RQ3. How does mixed context (on- *and* off-task queries) affect query recommendation performance?

RQ4. How do the following three methods affect query recommendation performance in a mixed context? (a.) using only the reference query, (b.) using the most recent m queries, or (c.) using the most recent m queries with same-task classification scores to weight the influence of each query.

RQ5. How does the sanitization method used to produce the crowd log of query reformulations effect recommendation performance?

To answer these questions, we perform a number of experiments using simulated search sequences derived from the TREC Session Track (Kanoulas et al., 2010, 2011). For recommendation, we rely on random walks over a query flow graph formed from a subset of the 2006 AOL query log. We measure query recommendation performance by the quality of the results returned for a recommendation, focusing primarily on

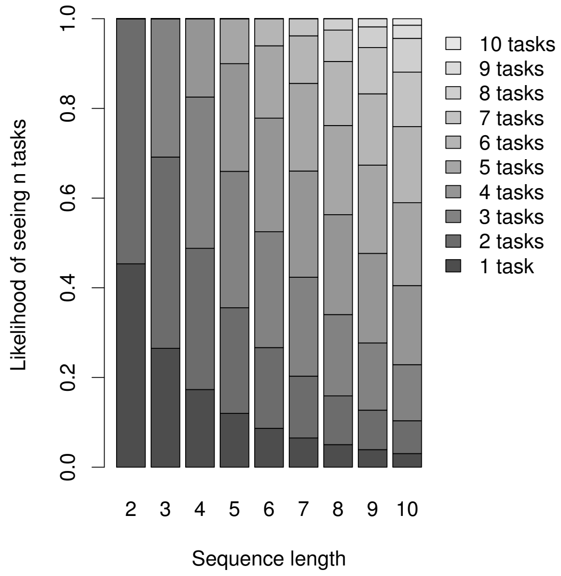


Figure 6.3. The distribution of seeing n tasks in a sequence of x queries as observed in a labeled sample of the AOL query log.

mean reciprocal rank (MRR). Our results show that on-task context is usually helpful, while off-task and mixed contexts are extremely harmful. However, automatic search task identification (Chapter 5) is a reliable way of detecting and discarding off-task queries.

There are four primary contributions that stem from this chapter: (1) an analysis of task-aware query recommendation demonstrating the usefulness of on-task query context, (2) an analysis of the impact of automatic search task identification on task-aware recommendation, in which we show the state of the art works very well, (3) an analysis of the effect of sanitized global data on recommendation performance, and (4) a generalized model of combining recommendations across a search context, regardless of the recommendation algorithm.

Figure 6.4 outlines the implementation that we will follow throughout this chapter. We describe how global data is aggregated, preprocessed, and used to provide query recommendations in Section 6.1. In Section 6.2, we outline how local data will be

Task-aware query recommendation

▷ **Preprocessing**

1. *What artifacts will be extracted?*
   ◇ Ordered pairs of queries that co-occur in the same session (see Section 6.1 for details).
2. *What preprocessing will take place on the global data?*
   ◇ A Term-Query Graph will be created (see Section 6.1 for details).
3. *What preprocessing will take place on the local data?*
   ◇ A search context will be generated consisting of the reference query and other queries from the user's search history that are classified as belonging to the same task as the reference query (see Section 6.2 for details).

▷ **On input**

4. *What is the input?*
   ◇ A search context consisting of a reference query and zero or more on-task queries.
5. *What request will the client send to the server?*
   ◇ A request will be made for each query in the context individually.
6. *What processing will the server perform for the client request?*
   ◇ The server will generate a list of query recommendations with scores for a given query.
7. *What response will the server send to the client?*
   ◇ The list of scored query recommendations.
8. *How will the server data be integrated with the local data?*
   ◇ The recommendations generated for each query in the search context will be merged together (see Section 6.3 for details).
9. *What is the output of the application?*
   ◇ A list of query recommendations for the reference query.

Figure 6.4. Implementation specifications for task-aware query recommendation.

utilized to form search contexts relative to a reference query. We describe several client-side integration models in Section 6.3. Finally, in Sections 6.4 and 6.5, we outline our experiments and analyze the results.

6.1 Global data

The query recommendation algorithm we use is based largely on the query flow graph (QFG) work of Boldi et al. (2008) and the term-query graph (TQGraph) work of Bonchi et al. (2012). We use a query flow graph G in which the vertices V are queries from a query log L and the edges E represent reformulation probabilities. For any vertex $v \in V$, the weights of all outgoing edges must sum to 1. A *reformulation* is defined as an ordered pair of queries (q_i, q_j) such that the pair occurs in a user search session in that order, though not necessarily adjacent. A *session* is defined to be the maximal sequence of queries and result clicks such that no more than t seconds separate any two adjacent events. The outgoing edges of v are normalized across all sessions and users in L.

While we do not require reformulations to be adjacent, Boldi et al. did. By considering all reformulations—adjacent and otherwise—within a session, we expand the coverage of G beyond using only adjacent query reformulations. We assume that for reformulations in which q_i and q_j are from different tasks, q_j will be an infrequent follower of q_i, and therefore statistically insignificant among q_i's outgoing edges. Boldi et al. used a thresholded and normalized chaining probability for the edge weights, but we do not due to the sparseness of our data (the AOL query logs).

To generate recommendations, we rely on a slight adaptation of the query flow graph, called the term-query graph (Bonchi et al., 2012). This adds a layer to the QFG that consists of all terms that occur in queries in the QFG, each of which points to the queries in which it occurs. Given a query q, we find recommendations by first producing random walk scores over all queries in Q for each term $t \in q$.

To compute the random walk with restart for a given term t, we must first create a vector $\mathbf{v}$ of length $|V|$ (i.e., with one element per node in Q). Each element corresponding to a query that contains t is set to 1 and all others are set to 0. This is our *initialization vector*. Next, we must select the probability, c, of restarting our random walk to one of the queries in our initialization vector. Given the adjacency matrix of G, A, and a vector $\mathbf{u}$ that is initially set to $\mathbf{v}$, we then compute the following until convergence:

$$\mathbf{u}_{i+1} = (1-c)A\mathbf{u}_i + c\mathbf{v}. \tag{6.1}$$

After convergence, the values in $\mathbf{u}$ are the random walk scores of each corresponding query q' for t. We denote this as the term-level recommendation score $\widehat{\mathbf{r}}_{\text{term}}(q'|t)$.

One issue with using the random walk score for a query is that it favors frequent queries. To address this, Boldi et al. use the geometric mean $\mathbf{r}_{\text{term}}$ of the random walk score $\widehat{\mathbf{r}}_{\text{term}}$ and its normalized score $\overline{\mathbf{r}}_{\text{term}}$. Given an initial query q, the scores for an arbitrary query q' can be computed by:

$$\begin{aligned}\overline{\mathbf{r}}_{\text{term}}(q'|q) &= \frac{\widehat{\mathbf{r}}_{\text{term}}(q'|q)}{\widehat{\mathbf{r}}_{\text{uniform}}(q')}, &(6.2)\\ \mathbf{r}_{\text{term}}(q'|q) &= \sqrt{\widehat{\mathbf{r}}_{\text{term}}(q'|q)\cdot\overline{\mathbf{r}}_{\text{term}}(q'|q)}\\ &= \frac{\widehat{\mathbf{r}}_{\text{term}}(q'|q)}{\sqrt{\widehat{\mathbf{r}}_{\text{uniform}}(q')}}, &(6.3)\end{aligned}$$

where $\widehat{\mathbf{r}}_{\text{uniform}}(q')$ is the random walk score produced for q' when the initialization vector $\mathbf{v}$ is uniform.

The final query recommendation vector is computed using a component-wise product of the random walk vectors for each term in the query. Specifically, for each query $q' \in V$, we compute the query-level recommendation score $\mathbf{r}_{\text{query}}(q'|q)$ as follows:

$$\mathbf{r}_{\text{query}}(q'|q) = \prod_{t\in q}\mathbf{r}_{\text{term}}(q'|t). \tag{6.4}$$

In the implementation we outlined in Figure 6.4, the TQGraph sits on the server. The client anonymously requests recommendations for individual queries. The server then generates the recommendations for a query and sends the scored recommendations back to the client.

6.2 Local data

As stated in the implementation we outlined in Figure 6.4, local processing involves generating a search context C of queries prior to a given reference query, where $|C| = m$ and $C[m]$ is the reference query. While there are many ways to construct a search context, for the purpose of task-aware query recommendation, we will consider task contexts, where all queries in C are part of the same task as the reference query $C[m]$.

To decide what queries to include in a task context, we first define a function $\mathbf{sametask}(i, m, C)$ that outputs a prediction in the range [0,1] as to how likely the queries $C[i]$ and $C[m]$ are to be part of the same search task given the context C. For all queries $C[i]$, $\mathbf{sametask}(i, m, C) > \tau$, where τ is a threshold. We use the Lucchese search task identification heuristic described in Chapter 5 (Lucchese et al., 2011).[2] In deciding if two queries q_i and q_j are part of the same task, we calculate two similarity measures: a lexical and a semantic score, defined as follows.

The lexical scoring function $\mathbf{s}_{\text{lexical}}$ is the average of the Jaccard coefficient between term trigrams extracted from the two queries and one minus the Levenshtein edit distance of the two queries. The score is in the range [0,1]. Two queries that are lexically very similar—ones where a single term has been added, removed, or reordered, or queries that have been spell corrected—should have an $\mathbf{s}_{\text{lexical}}$ score close to one.

[2] Our experiments were performed prior to our analysis of search task identification described in Chapter 5, which is why we use the Lucchese classifier and not a random forest model.

The semantic scoring function $\mathbf{s}_{\text{semantic}}$ is made of up two components. The first, $\mathbf{s}_{\text{wikipedia}}(q_i, q_j)$, creates the vectors v_i and v_j consisting of the tf·idf scores of every Wikipedia document relative to q_i and q_j, respectively. The function then returns the cosine similarity between these two vectors. The second component, $\mathbf{s}_{\text{wiktionary}}(q_i, q_j)$ is similarly computed, but over Wiktionary entries. We then set $\mathbf{s}_{\text{semantic}}(q_i, q_j) = \max\big(\mathbf{s}_{\text{wikipedia}}(q_i, q_j), \mathbf{s}_{\text{wiktionary}}(q_i, q_j)\big)$. As with the lexical score, the range of the semantic score is [0,1].

The combined similarity score, $\mathbf{s}$, is defined as follows:

$$\mathbf{s}(q_i, q_j) = \alpha \cdot \mathbf{s}_{\text{lexical}}(q_i, q_j) + (1 - \alpha) \cdot \mathbf{s}_{\text{semantic}}(q_i, q_j).$$

We can define a same-task scoring function to use s directly, as follows:

$$\mathbf{sametask}_1(i, j, C) = \mathbf{s}(C[i], C[j]). \tag{6.5}$$

Alternatively, we can run one extra step: single-link clustering over the context C using s as the similarity measure. Clustering allows us to boost the similarity score between two queries that are only indirectly related. Similar to Liao et al. (2012), our choice of clustering follows the results of Lucchese et al. (2011), who describe a weighted connected components algorithm that is equivalent to single-link clustering with a cutoff of η. After clustering, we use the notation $K_C[q]$ to represent the cluster or task associated with query q in context C; if two queries $q, q' \in C$ are part of the same task, then $K_C[q] = K_C[q']$, otherwise $K_C[q] \neq K_C[q']$. A scoring function that uses task clustering is the following:

$$\mathbf{sametask}_2(i, j, C) = \max_{\substack{k \,\in\, [1, |C|]\,: \\ k\, \neq i \wedge \\ K_C[C[k]] \,= K_C[C[j]]}} \mathbf{s}(C[i], C[k]). \tag{6.6}$$

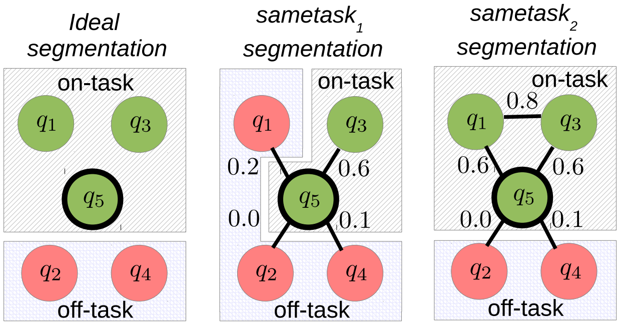


Figure 6.5. Examples of on-task/off-task segmentations using **sametask**$_1$ and **sametask**$_2$ scoring. The reference query, q_5, sits in the bolded center node. Note that the edge (q_1, q_5) goes from 0.2 using **sametask**$_1$ to 0.6 under **sametask**$_2$ due to q_1's strong similarity to q_3, which has a similarity score of 0.6 with q_5.

Note that **sametask**$_2$ will return the highest similarity between $C[i]$ and any member of $C[j]$'s tasks, excluding $C[i]$. Figure 6.5 illustrates a case in which **sametask**$_2$ improves over **sametask**$_1$; note, however, that **sametask**$_2$ can also be harmful when an off-task query is found to be similar to an on-task query.

6.3 Integrating local and global data

In this section, we introduce formal definitions of general and task-based contexts. We explore several method of integrating local context information with recommendations produced from a global model.

6.3.1 Generalized context model

The recommendation models described earlier, and recommendation algorithms in general, that generate suggestions with respect to a single query can be easily extended to handle additional contextual information. The basic idea is simple: when generating recommendations for a query, consider the search context consisting of the m most recently submitted queries from the user, weighting the influence of each according to some measure of importance. Many functions can be used to measure the importance of a context query. The two we consider in this paper are how far back in a user's search history a query occurs and whether the query appears to be related to a user's current task. Others may include whether a context query was quickly abandoned or spell corrected, how many results the user visited, the time of day they were visited, and other behavioral aspects. In this section, we introduce a generalized model that makes it easier for us to talk about the various importance functions we are interested in and how they can be used with additional functions in future work.

Assume that we have a search context C that contains all the information about a user's search behavior related to a sequence of m queries, with the m^{th} query, $C[m]$, being the most recently submitted query. Also assume that we have a set of n importance functions, $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_n$ and corresponding weights $\lambda_1, \lambda_2, \ldots, \lambda_n$ that tell us how much weight to give to each of the importance functions. We will represent corresponding functions and weights as tuples in a set $F = \{\langle \lambda_1, \boldsymbol{\theta}_1 \rangle \, \langle \lambda_2, \boldsymbol{\theta}_2 \rangle \, \ldots, \langle \lambda_n, \boldsymbol{\theta}_n \rangle\}$. We compute the context-aware recommendation score for a query suggestion q' as follows:

$$\mathbf{r}_{\text{context}}(q'|C, F) = \sum_{i=1}^{m} \left(\mathbf{r}_{\text{query}}(q'|C[i]) \sum_{\langle \lambda, \boldsymbol{\theta} \rangle \in F} \lambda \cdot \boldsymbol{\theta}(i, m, C) \right). \qquad (6.7)$$

Each importance function $\boldsymbol{\theta}$, takes three parameters: i, m, C, where i is the position of the query within context C for which importance is being measured and m is the position of the reference query. In this work, the reference query is always the last query of C, but the model does not require that assumption. The $\mathbf{r}_{\text{context}}$ recommendation scoring function scores q' with respect to each query in the context ($C[i]$) and adds that score to the overall score with some weight that is computed as the linear combination of the importance function values for that query.

6.3.2 Decaying context model

One of the importance functions we consider in this paper is a decaying function, where queries earlier in a user's context are considered less important than more recent queries. As such, queries submitted more recently have a greater influence on recommendation scores. This has the intuitive interpretation that users are less interested in older queries, otherwise they would not have moved on to new queries.

Boldi et al. (2008) discussed a decay weighting method for entries in the random walk initialization vector ($\mathbf{v}$ in Eq. 6.1). They proposed that each query in a search context receive a weight proportional to β^d, where d is the distance in query count from the current query. For example, the second most recent query would get a weight of β^1, because it's one query away from the most recent query.

While the Boldi et al. method is specific to recommendations using random walks, we can transfer their exponential decay function to our model as follows:

$$\mathbf{decay}(i, j, C) = \beta^{j-i} \tag{6.8}$$

$$\mathbf{r}_{\text{decay}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle 1.0, \mathbf{decay}\rangle\}). \tag{6.9}$$

6.3.3 Task context model

While decaying the influence of queries earlier in a search context is a natural importance function, we are also interested in functions that incorporate the degree to which a query is on the same task as the reference query. It is reasonable to assume (an assumption we test) that queries from a search context that are part of the same task should be more helpful in the recommendation process than queries that are not.

As we have stated earlier, Lucchese et al. observed that 74% of web queries are part of multi-task search sessions (Lucchese et al., 2011) while Jones and Klinkner found that 17% of tasks are interleaved in web search (Jones & Klinkner, 2008).Using a labeled sample of the AOL query log, we observed an exponential decrease in the likelihood that the previous m queries are part of the same task as m increases (see Figure 6.3). This suggests that using the m most recent queries as the the search context for generating recommendations will likely introduce off-topic information, causing recommendations that seem out of place. Therefore, it may be beneficial to identify which queries from that context share the same task as the reference query.

Formally, given a search context C with m queries, we define a task context T to be the maximal subset of queries in C that share a task common to $C[m]$:

$$T \subseteq C \mid \forall\, i \in [1, m], C[i] \in T \iff \mathbf{sametask}(i, m, C) > \tau, \tag{6.10}$$

where $\mathbf{sametask}(i, m, C)$ follows one of the definitions given in Section 6.2.

Once we have T, the natural question to pose is how do we use it? One method would be to treat T just as C and use it with the $\mathbf{r}_{\text{decay}}$ function, i.e., $\mathbf{r}_{\text{decay}}(q'|T)$. However, it may be that the off-task context is still useful, just not as useful as T. To support both of these points of view, we can use the following *hard task* recommendation scoring functions:

$$\mathbf{taskdecay}(i, j, C) = \beta^{\mathbf{taskdist}(i,j,C)}, \tag{6.11}$$

$$\mathbf{hardtask}(i, j, C) = \begin{cases} \mathbf{taskdecay} & \text{if } \mathbf{sametask} > \tau, \\ 0 & \text{otherwise}, \end{cases} \tag{6.12}$$

$$\mathbf{r}_{\text{hardtask}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle\lambda, \mathbf{hardtask}\rangle, \langle 1-\lambda, \mathbf{decay}\rangle\}), \tag{6.13}$$

where λ can be used to give more or less weight to the task context and **taskdist** is the number of on-task queries between $C[i]$ and $C[j]$. If we set $\lambda = 1$, we only use the task context, whereas with $\lambda = 0$, we ignore the task context altogether. If we use $\lambda = 0.5$, we use some of the task information, but still allow the greater context to have a presence. Note that we have left off the parameters to **decay** and **sametask** in Eq. 6.13 for readability.

This approach may work well if one is comfortable with setting a hard threshold τ on the output of the **sametask** function. If, however, we want to provide a mechanism by which we use the output of **sametask** as a confidence, we can use the following *soft task* recommendation scoring functions:

$$\mathbf{softtask}(i, j, C) = \mathbf{sametask} \cdot \mathbf{decay}, \tag{6.14}$$

$$\mathbf{r}_{\text{softtask}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle \lambda, \mathbf{softtask} \rangle, \langle 1 - \lambda, \mathbf{decay} \rangle\}). \tag{6.15}$$

Here, λ smooths between using and not using the same-task scores to dampen the decay weights. We have left off the parameters to **sametask** and **decay** in Eqs. 6.14 and 6.15.

Two additional models we consider are both variations of what we call *firm task* recommendation, as they combine aspects of both the hard and soft task models. The first, called **firmtask1**, behaves similarly to the soft task model, except that we give a the weight of zero to any queries with a same task score at or below the threshold τ. The second, called **firmtask2**, is identical to the hard task model, except that the task classification score is used in addition to the **taskdecay** weight. Mathematically, the firm task recommendation models are described by:

$$\mathbf{firmtask1}(i, j, C) = \begin{cases} \mathbf{sametask} \times \ \mathbf{decay} & \text{if } \mathbf{sametask} > \tau, \\ 0 & \text{otherwise,} \end{cases} \tag{6.16}$$

$$\mathbf{r}_{\text{firmtask1}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle \lambda, \mathbf{firmtask1} \rangle, \langle 1 - \lambda, \mathbf{decay} \rangle\}), \tag{6.17}$$

$$\mathbf{firmtask2}(i, j, C) = \begin{cases} \mathbf{sametask} \ \times \ \mathbf{taskdecay} & \text{if } \mathbf{sametask} > \tau, \\ 0 & \text{otherwise,} \end{cases} \tag{6.18}$$

$$\mathbf{r}_{\text{firmtask2}}(q'|C) = \mathbf{r}_{\text{context}}(q'|C, \{\langle \lambda, \mathbf{firmtask2} \rangle, \langle 1 - \lambda, \mathbf{decay} \rangle\}). \tag{6.19}$$

	sametask	$=$	$[.8,\ .2,\ .1,\ .9,\ 1.0]$
a.	$\mathbf{r}_{\text{decay}}$	$\approx$	$[.4,\ .5,\ .6,\ .8,\ 1.0]$
b.	$\mathbf{r}_{\text{softtask}}$	$\approx$	$[.3,\ .1,\ .1,\ .7,\ 1.0]$
c.	$\mathbf{r}_{\text{firmtask1}}$	$=$	$[.3,\ .0,\ .0,\ .7,\ 1.0]$
d.	$\mathbf{r}_{\text{firmtask2}}$	$=$	$[.5,\ .0,\ .0,\ .7,\ 1.0]$
e.	$\mathbf{r}_{\text{hardtask}}$	$\approx$	$[.6,\ .0,\ .0,\ .8,\ 1.0]$

Figure 6.6. An example of the degree to which each query in a context contributes (right column) given its predicted same-task score (top row) for: (a.) decay only, (b.) soft task, (c.) firm task-1, (d.) firm task-2, and (e.) hard task recommendation. We set $\beta = 0.8$ for all, and $\lambda = 1, \tau = 0.2$ for b.–e.

Note that unlike the hard task model, the decay component of the **firmtask1** model is affected by every query, not just those above the threshold.

For example, suppose we have a context C with five queries, $q_1, \ldots, q_5$. Relative to the reference query, q_5, suppose applying **sametask** to each query produces the same-task scores [0.4, 0.2, 0.1, 0.95, 1.0]. If we set $\tau = 0.2$, then $T = [q_1, q_4, q_5]$. Using $\beta = 0.8$, notice in Figure 6.6 how the importance weight of each query in the context changes between using only the decay function (a.) and setting $\lambda = 1$ for the task-aware recommendations (b.–e.). Note that when $\lambda = 0$, the hard, firm, and soft task recommendation scores are equivalent (they all reduce to using the decay-only scoring function).

There are two primary differences between using hard- and soft task recommendation. First, hard task recommendation does not penalize on-task queries that occur prior to a sequence of off-task queries, e.g. in Figure 6.6, we see that q_1 is on-task and **hardtask** treats it as the first query in a sequence of three: $\beta^{n-1} = 0.8^2$. Conversely, the soft task recommendation model treats q_1 as the first in a sequence of five: $\beta^{m-1} = 0.8^4$.

Second, soft task recommendation can only down-weight a query's importance weight, unlike hard task recommendation, which can significantly increase the weight of an on-task query further back in the context (e.g., the far left value for $\mathbf{r}_{\text{hardtask}}$

in Figure 6.6). At the same time, however, soft task recommendation only allows a query to be given a zero weight if its same-task score is zero. The two firm task models balance these aspects in different ways.

6.4 Experimental setup

In this section, we describe the data, settings, and methodology used for the experiments.

6.4.1 Constructing a query flow graph

We extracted query reformulations from the 2006 AOL query log, which includes more than 10 million unique queries making up 21 million query instances submitted by 657,426 users between March–April 2006. Considering all ordered pairs from a 30-query sliding window across sessions with a maximum timeout of 26 minutes, we extracted 33,218,915 distinct query reformulations to construct a query flow graph (compared to 18,271,486 if we used only adjacent pairs), ignoring all dash ("-") queries, which correspond to queries that AOL scrubbed or randomly replaced. The inlink and outlink counts of the nodes in the graph both have a median of 2 and a mean of about 5. If we were to use only adjacent reformulations from the logs, the median would be 1 and the mean just under 2.

6.4.2 Task data

We used the 2010 and 2011 TREC Session Track (Kanoulas et al., 2010, 2011) data to generate task contexts. The 2010 track data contains 136 judged sessions, each with two queries (totaling 272 queries), covering three reformulation types: drift, specialization, and generalization relative to the first search. We ignore the reformulation type. The 2011 track data consists of 76 variable length sessions, 280 queries (average of 3.7 queries per session), and 62 judged topics. Several topics have multiple corresponding sessions. In total, we use all 212 judged sessions from both years. The relevance judgments in both cases are over the ClueWeb09 collection. Our goal is to provide recommendations to retrieve documents relevant to the last query in each session, thus we mark the last query as the reference query.

Each session constitutes a single task, and henceforth we refer to the sessions as tasks. Since the TREC data consists of single tasks, we need some way of simulating the case that multiple tasks are interleaved. We describe our approach for this next.

6.4.3 Experiments

To answer our four research questions, we use the following set of experiments. Throughout all of these experiments, the baseline is to use only the reference query for generating query recommendations.

Experiment 1. For RQ1, which seeks to understand the effect of including relevant context on recommendation performance, we use each task T from the TREC Session Track and recommend suggestions using the most recent m queries for $m = [1, |T|]$. If incorporating context is helpful, then we should see an improvement as m increases. Note that $m = 1$ is the case in which only the reference query is used.

Experiment 2. To address RQ2, which asks how off-task context affects recommendation performance, we modify the experiment described above to consider a context of $m = [1, |T|]$ queries such that queries 2 through $|T|$ are off-task. To capture the randomness of off-task queries, we evaluate over R random samples of off-task contexts (each query is independently sampled from other tasks, excluding those with the same TREC Session Track topic) for each task T and each value of $m > 1$. If off-task context is harmful, we should see a worsening trend in performance as m increases.

Experiment 3. To address RQ3, which asks how query recommendation performance is affected by a context that is a mix of on- and off-task queries, we rely on a simulation of mixed contexts. As we saw in Figure 6.3, the probability that a sequence of m queries share the same task decreases exponentially as m increases, and so the mixed context assumed in RQ3 is realistic if not typical. We simulate mixed contexts by taking each task T of length n and considering a context window of length $m = [1, n + R]$, where R is the number of off-task queries to add into the context. The last query in the context q_m always corresponds to the last query q_n in T. Queries $q_1, \ldots, q_{m-1}$ consist of a mix of the queries from T and other tasks from the TREC Session Track. The queries from T will always appear in the same order, but not necessarily adjacent.

To incorporate noise, we initially set $C = []$. We select R off-task queries as follows: first, we randomly select an off-topic task, O, from the TREC Session Track and take the first R queries from that task. If $|O| < R$, we randomly selected an addition off-topic task and concatenate its first $R - |O|$ queries to O. We continue the process until $|O| = |R|$. We now randomly interleave T and O, the only rule being that T_n—the reference query—must be the last query in C. For a given value of R, we can perform many randomizations and graph the effect of using the most recent $n + R$ queries to perform query recommendation.

Experiment 4. The fourth research question, RQ4, asks how mixed contexts should be used in the query recommendation process. We have limited ourselves to consider three possibilities: (a.) using only the reference query (i.e., our baseline throughout these experiments), (b.) using the most recent $n + R$ queries (i.e., the results from Experiment 3), or (c.) incorporating same-task classification scores. Experiment 4 concentrates on (c.) and analyzes the effect of incorporating same-task classification scores during the search context integration process. This is where we will compare the task-aware recommendation models described in the previous section.

Experiment 5. The final research question, RQ5, questions the impact of privacy on recommendation performance. Given each of the sanitized crowd logs, we re-run

Experiments 1–4, allowing us to understand the effect on query recommendation performance.

6.4.4 Technical details

For the query recommendation using the TQGraph, we used a restart probability of $c = 0.1$, as was found to be optimal by Bonchi et al. (2012).[3] To increase the speed of our recommendation, we only stored the 100,000 top scoring random walk results for each term. Bonchi et al. found this to have no or very limited effects on performance when used with $c = 0.1$.

For task classification, we used the parameters found optimal by Lucchese et al. (2011): $\eta = 0.2$ (used during task clustering) and $\alpha = 0.5$ (used to weight the semantic and lexical features). We also set $\tau = \eta$ since τ is used in much the same way in the task-aware recommendation models.

To evaluate recommendations, we retrieved documents from ClueWeb09 using the default query likelihood model implemented in Indri 5.3 (Metzler & Croft, 2004).[4] We removed spam by using the Fusion spam score dataset (Cormack et al., 2011) at a 75th percentile, meaning we only kept the least spammy 25% of documents.[5]

To address Experiment 5, we consider several values of d, $d \in \{1, 2, 4, 8, 16, 32\}$, for the differential privacy mechanisms DP_a, DP_u, and ZEALOUS. We set $\epsilon = 10$, which is very high, but necessary in order to generate non-empty crowd logs (and even then, we encountered a few empty crowd logs). We set $\delta = 1/657427$. We took 10 samples for each value of d and average performance across those samples. We considered several values of k for the frequency thresholding mechanisms FT_a and

[3] Note that they refer to the restart value α, where $c = 1 - \alpha$.

[4] http://www.lemurproject.org/indri/

[5] http://plg.uwaterloo.ca/~gvcormac/clueweb09spam/

FT_u: $k \in \{2, 5, 10, 50\}$. All crowd logs were generated over the full three months of the AOL query log.

6.5 Results and analysis

In this section, we cover the results of each of the experiments described in Section 6.4.3. We then discuss the meaning of our findings as well as their broader implications.

6.5.1 Experimental results

In all experiments, we measured recommendation performance using the mean reciprocal rank (MRR) of ClueWeb09 document retrieved for the top scored recommendation averaged over the 212 TREC Session Track tasks. There are several ways one can calculate relevance over the document sets retrieved for recommendations, such as count any document retrieved in the top ten for any of the context queries as non-relevant (rather harsh), indifferently (resulting in duplicate documents), or by removing all such documents from the result lists of recommendations. We elected to go with the last as it is a reasonable behavior to expect from a context-aware system. We removed documents retrieved for any query in the context, not just those that are on-task. This is a very conservative evaluation and is reflected in the performance metrics.

Experiment 1. With this experiment, our aim was to quantify the effect of on-task query context on recommendation quality. Focusing on the top line with circles in Figure 6.7, the MRR of the top scored recommendation averaged over the 212 tasks performs better than using only the reference query (middle line with squares). To generate these scores, we used the $\mathbf{r}_{\text{decay}}$ model with $\beta = 0.8$, as set by Boldi et al. (2008) in their decay function. For each value of m, if a particular task T has fewer than m queries, the value at $|T|$ is used. The MRR scores are low because for a

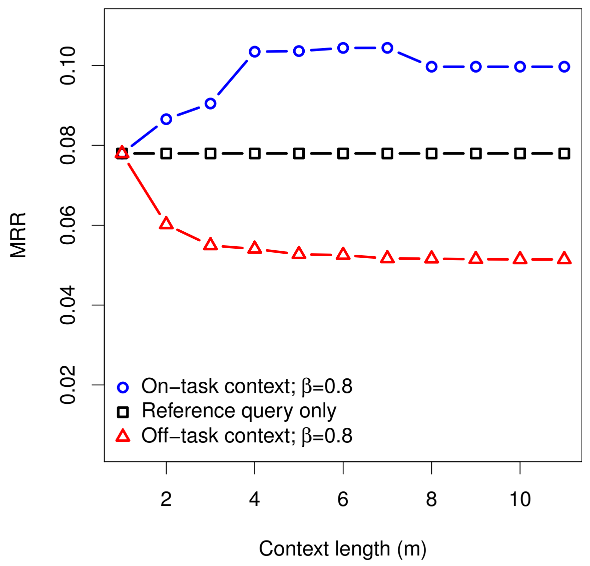


Figure 6.7. The effect of adding on-task (blue circles) and off-task (red triangles) queries versus only the reference query on recommendation MRR (black squares). MRR is calculated on the top scoring recommendation.

large number of tasks, none of the methods provide any useful recommendations. We performed evaluations where such tasks were ignored and found that the MRR does indeed increase and the relationship between the methods plotted stays the same. To ensure comparability with future work, we report on all tasks.

While performance is better on average in Figure 6.7, the top bar chart in Figure 6.8 breaks the performance down by the TREC search tasks and we can see that there are many tasks for which on-task context is very helpful, as well as several where it hurts. Note that some of the tasks are not displayed for readability.

Experiment 2. The goal of the second experiment was to ascertain the effect of off-task context on query recommendation. We generated 50 random off-task contexts for each task and report the micro-average across all trials. The bottom line with triangles in Figure 6.7 shows that adding off-task queries under the $\mathbf{r}_{\text{decay}}$ model with $\beta = 0.8$ rapidly decreases recommendation performance for low values of m before more or less leveling off around $m = 5$ (it still decreases, but much slower).

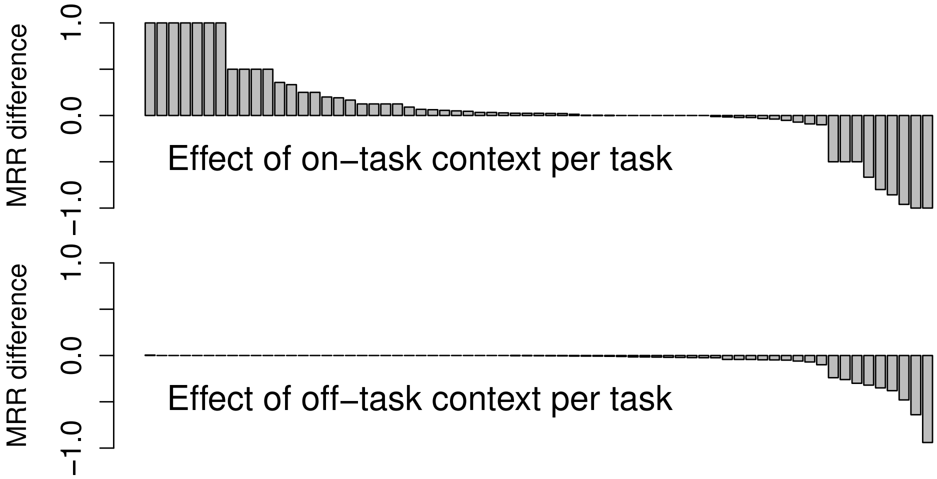


Figure 6.8. The per-session effect of on- and off-task context on the change in MRR of the top scoring recommendation. The y-axis shows the difference between the MRR of using context and using only the reference query. A higher value means context improved MRR. Note that 145 tasks were removed as neither on- nor off-task context had an effect. The tasks are not aligned between the two plots and cannot be compared at the task level.

Its performance is well below that of the baseline of using only the reference query, making it clear that off task context is extremely detrimental.

Turning to the bottom plot in Figure 6.8, we see that off-task context has an almost entirely negative effect (there is an ever so slight increase in performance for the task represented by the far left bar). Interestingly, for the severely compromised tasks on the far right, the effect is not as negative as when on-task context hurts. We have not conducted a full analysis to understand this phenomena, but one possible cause is the averaging over 50 trials that takes place for the off-task contexts. We leave investigations into this issue for future work.

Experiment 3. With Experiment 3, we wanted to understand the effect of mixed contexts—consisting of both on- and off-task queries—on query recommendation performance. As explained earlier, the experiment explores the performance of tasks when R noisy queries are added to the entire set of on-task queries. The bottom line with triangles in Figure 6.9 shows just this, using $\mathbf{r}_{\text{decay}}$ with $\beta = 0.8$. The far left point, where $R = 0$, lines up with the far right point of the on-task line in Figure 6.7. We randomly generated 50 noisy contexts per task for each value of R. The solid line shows the micro-averaged MRR over all tasks' samples. The dotted lines on either side show the minimum and maximum values for the micro-average MRR on a set of 1,000 sub-samples (with replacement) of the original 50. As you can see, the bounds indicate relatively low variance of the micro-average across the 212 tasks. There are still certain tasks for which performance is very high or very low (that is, the bounds on the micro-average do not inform us of the variance among tasks).

An important observation from this experiment is that performance dips below that of the baseline when even a *single* off-task query is mixed in. This is quite startling when you consider that the chances of three queries (at $R = 1$, all contexts are of at least length three) in a row belonging to a single task are below 30% (see Figure 6.3) and that roughly 40% of tasks in the wild are of length three or more (see Figure 6.2). These results clearly show that blindly incorporating mixed context is a poor method of incorporating context.

Experiment 4. In the fourth experiment, we hoped to determine the effects of using recommendation models that consider the reference query only, the entire context, or the entire context, but in a task-aware manner. The first two were addressed in the previous experiments, where we learned that using the reference query is more effective than blindly using the entire context. Figure 6.9 shows the results of using the models we introduced in Section 6.3.3. We used the same randomizations as in Experiment 3 and likewise generated the minimum and maximum bounds around each model's performance line. For these experiments, $\textbf{sametask}_1$ scores were used to produce same-task scores. We also performed the experiment using $\textbf{sametask}_2$ and found it was comparable. We used $\lambda = 1$ for all task-aware models; setting it to anything less resulted in an extreme degradation of performance.

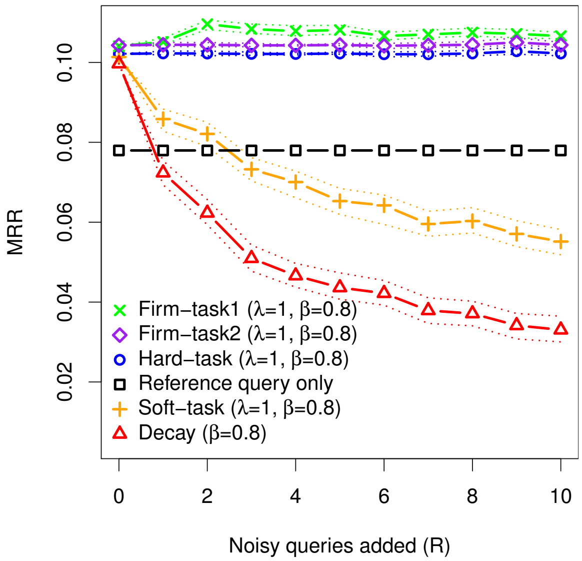


Figure 6.9. The effect of adding off-task queries to a task context on MRR when same task classification is used and is not used versus only using the reference query (black squares). The $\mathbf{sametask}_1$ scoring method is used for all task-aware recommendation models. MRR is calculated on the top scoring recommendation.

There are several interesting observations. First, the firm-task models performed best, though it is likely that the performance of the $\mathbf{r}_{\text{firmtask1}}$ model (the top line with x's) would decrease with larger amounts of noise because the decay function depends on the length of the context, not the number of queries predicted to be on-task. Thus, for on-task queries occurring early in very large contexts, the decay weight will effectively be 0. You may notice that this model also increases for a bit starting at $R = 2$. This is likely due to the decay function used: since every query in the context, and not just the on-task queries, counts toward the distances between an on-task query and the reference query under $\mathbf{r}_{\text{firmtask1}}$, on-task queries are actually down-weighted to a greater degree than in the $\mathbf{r}_{\text{firmtask2}}$ and $\mathbf{r}_{\text{hardtask}}$ models. The graph indicates that this change in weighting is helpful. This also suggests that setting β differently may improve the performance of the other models.

The $\mathbf{r}_{\text{firmtask2}}$ model (diamonds) comes in at a close second and narrowly outperforms $\mathbf{r}_{\text{hardtask}}$ (circles). All three models outperform the baselines—using only the reference query (squares) and $\mathbf{r}_{\text{decay}}$ over the entire context (triangles).

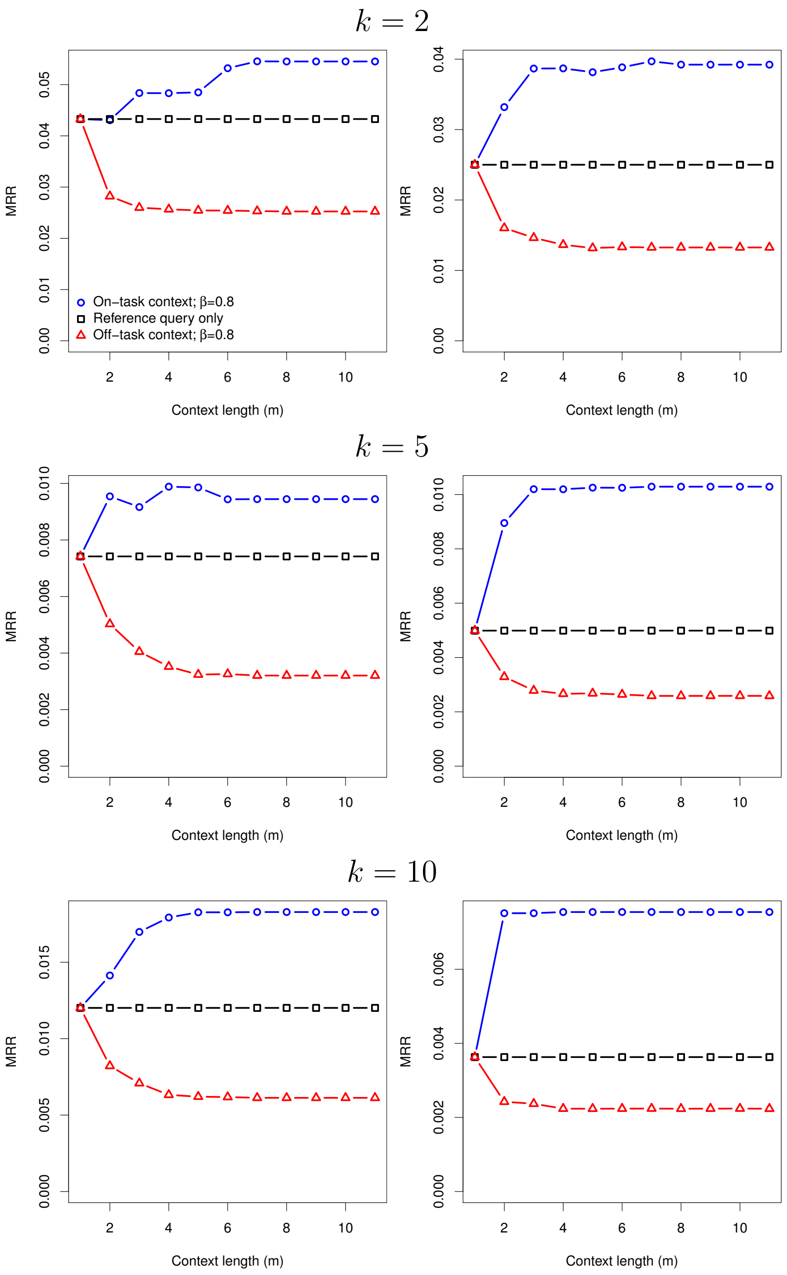
The $\mathbf{r}_{\text{softtask}}$ model, however, performs rather poorly. While it can offer some improvement over using just the reference query for a small amount of noise, once the noise level reaches four off-task queries, it is not longer viable. It does, however, outperform the decay model applied in a task-unaware manner.

Another interesting point is that the performance of the task-aware models is actually better at $R = 0$ than if the known tasks are used. The likely explanation is that the same-task scores prevent on-task queries that are quite different from the reference query from affecting the final recommendation. These kinds of queries may introduce more noise since their only recommendation overlap with the reference query may be generic queries, such as "google". This is not always the case, however. For example, one task consists of the queries ["alan greenspan", "longest serving Federal Reserve Chairman"]. The first query is detected to be *off-task*, however, it is needed to generate decent recommendations since the reference query generates generic Federal Reserve-related queries and not ones focused on Alan Greenspan.

Overall, though, the same-task classification with a threshold $\tau = 0.2$ worked well. The same-task classification precision relative to the positive class was on average 80%. The precision relative to the negative class varied across each noise level, but was on average 99%. The average accuracy was 93%. The decent same-task classification is why the three models at the top of Figure 6.9 are so flat.

Experiment 5. For Experiment 5, we looked at the effect of sanitization on query recommendation performance. We re-ran each of the earlier experiments, but using a TQGraph made from sanitized crowd logs. For Experiments 1 and 2, we found the same relationship held in most cases, on-task context producing a boost in performance over using only the reference query, and off-task context hampering performance. The exceptions were when all values of MRR were zero, which was the case for all of the ZEALOUS sanitizations, for DP_a and DP_u with $d \geq 8$, and for FT_u at $k = 50$. For the other values of d for DP_a and DP_u, the average performance never peaked greater than 0.0005, making the findings practically useless. Sanitization under frequency thresholding proved to be more useful. With DP_a at $k = 2$, performance dropped to under MRR= 0.06 and under MRR= 0.05 for DP_u. However, by $k = 5$—the value of k for which we have approval from the University of Massachusetts Institutional Review Board to use when releasing data—maximum performance of on-task context drops to MRR= 0.01. As with the differential privacy mechanisms, this is not practically useful performance. Figure 6.10 show the degredation in performance under the frequency thresholding mechanisms as k increases. One very interesting observation is that performance actually increases under DP_a going from $k = 5$ to $k = 10$. One speculative explanation for this is that $k = 10$ causes less frequent, noisy queries to be eliminated from the TQGraph. Perhaps the same is not seen under FT_u because noise is more likely to affect FT_a in the sense that a single user entering a query many times can cause it to appear in a TQGraph sanitized under FT_a.

For Experiments 3 and 4, we found a similar tend: the relationships are similar to those found when an unsanitized TQGraph is used, but the MRR is so low as to be practically useless. We did find that at higher values of d, the importance of context improved over using just the reference query. For example, Figure 6.11 shows that using context information triples the performance; however, effectively it doesn't matter because MRR does not exceed 0.0013. Under FT_a and FT_u, performance is slightly more practical, as we saw before. Figure 6.12 shows performance under these models for $k = \{2, 5, 10\}$. Again, we see the same jump in performance under FT_a going from $k = 5$ to $k = 10$.



$k = 2$

$k = 5$

$k = 10$

Figure 6.10. Query recommendation MRR when using on- and off-task context and using only the reference query. On the left, the TQGraph was sanitized using FT_a, and on the right with FT_u.

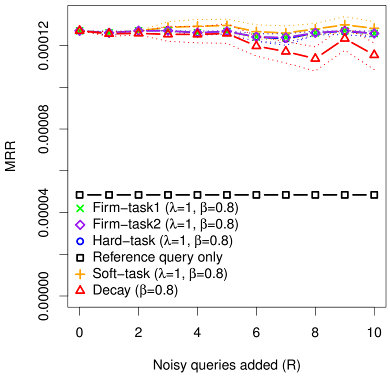


Figure 6.11. Query recommendation MRR in a mixed-context situation when the TQGraph is sanitized using DP_u with $d = 4$.

6.5.2 Discussion

The results of our experiments demonstrate not only the usefulness of on-task context, but also the extreme impact of off-task and mixed contexts. The results from Experiment 4 suggest that the appropriate model is one that balances a hard threshold to remove any influence from context queries predicted to be off-task, and to weight the importance of the remaining queries by both their distance to the reference query and by the confidence of the same-task classification. Based on the results, we recommend the $\mathbf{r}_{\text{firmtask2}}$ model, since its performance will be consistent regardless of how far back in a user's history we go, unlike $\mathbf{r}_{\text{firmtask1}}$.

We did not see any substantial effects from using task clustering, as Liao et al. (2012) used. However, other task identification schemes may perform differently; after all, as we saw in Experiment 4, our task identification method actually caused slight improvements over using the true tasks.

To get a feel for the quality of the recommendations produced generally with the AOL query logs and specifically by different models, consider the randomly generated mixed context in Figure 6.13 (also Figure 6.1). The top five recommendations from three methods for this context are shown in Figure 6.14. Notice that using only the

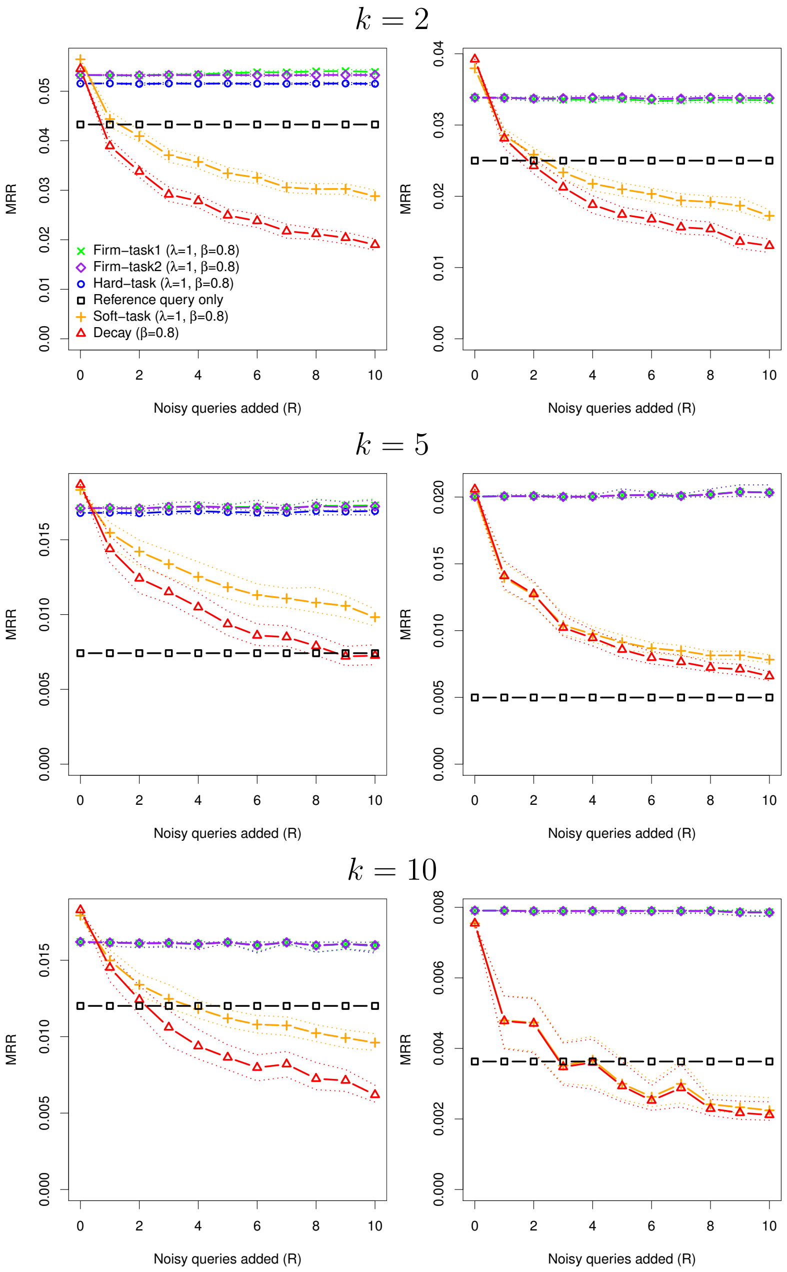


Figure 6.12. Query recommendation MRR when using mixed-context and using only the reference query. On the left, the TQGraph was sanitized using FT_a, and on the right with FT_u.

No.	Query context	sametask$_1$
5.	**satellite internet providers**	**1.00**
4.	**hughes internet**	**0.44**
3.	ocd beckham	0.02
2.	reviews xc90	**0.04**
1.	buy volvo semi trucks	0.04

Figure 6.13. An example of a randomly generated mixed context along with the same-task scores. The top query (No. 5) is the reference query. The bolded queries are on-task.

Rank	Reference query	RR	Decay	RR	Hard task ($\lambda = 1$)	RR
1.	alabama satellite internet providers	0.08	2006 volvo xc90 reviews		hughes internet.com	1.00
2.	sattelite internet	0.25	volvo xc90 reviews		hughes satelite internet	1.00
3.	satellite internet providers	1.00	hughes internet.com	1.00	alabama satellite internet providers	0.08
4.	satelite internet for 30.00	0.02	hughes satelite internet	1.00	satellite high speed internet	0.17
5.	satellite internet providers northern california	0.06	alabama satellite internet providers	0.08	sattelite internet	0.25

Figure 6.14. The top five suggestions generated from three of the models for the randomly generated context shown in Figure 6.13. Reciprocal rank (RR) values of 0 are left blank.

reference query produces suggestions with non-zero MRR, though the suggestion with the highest MRR is acutally the same as the reference query. Meanwhile, blindly using the context produces suggestions at the top that are swamped by the off-task queries, even though they occur much earlier in the context, and therefore are significantly down-weighted. A couple high-MRR queries are suggested, but not until the third rank. This is in stark contrast to using the hard task model, which recommends queries similar in nature to the reference query, but with higher MRR, and ranks the suggestions with the highest MRR at the top.

Many of the tasks for which task-aware recommendation performed well involved specialization reformulations. Some examples include: *heart rate→slow fast heart* *rate*, *bobcat tractor→bobcat tractor attachment*, *disneyland hotel→disneyland hotel reviews*, and *elliptical trainer→elliptical trainer benefits*. One possible reason for this is that incorporating recommendations for a context query that is a subset of the reference query focuses the final recommendations on the most important concepts.

None of the experiments used notions of temporal sessions or complete user histories. We did this mainly because the mixed contexts were generated and not from actual user logs where context windows could be tied to temporal boundaries, e.g., two-day windows. We believe that by focusing on factors such as on- and off-task queries, we struck at the core questions in this space. We leave testing whether the results from these experiments apply equally to real user data as future work, but we believe they will, especially given the results of related studies, such as those conducted by Liao et al. (2012) and Filali et al. (2010).

Finally, it appears that sanitization using differentially private mechanisms is not practical for this application with these numbers of users. Using FT_a and FT_u with low values of k can provide performance that is more on par with that of using an unsanitized data set, however, performance is still low. It seems that more users would improve the results. An alternative artifact representation, one that makes more artifacts publishable and is still useful (e.g., contains query text) may also help. For example, query templating, where specific nouns are replaced with temples, e.g., *bars in boston* would be templated as *bars in* $\langle$*city*$\rangle$.

6.6 Summary

We investigated several research questions concerning the effects of personalization and sanitization on query recommendation. We introduced a model for integrating query recommendations across on-task searches and found that on-task context is more helpful than using the reference query or blindly using context that contains noisy queries for several test sessions. We also used an off-the-shelf same-task classifier which helped us effectively detect and use on-task searches automatically.

We found that the relative improvements of on-task context are still present when the global data is sanitized under most models, but that the absolute performance is quite low. With the exception of $k = 2$ for FT_a and FT_u, sanitization appears to render global data useless. A larger user base and perhaps a different artifact representation may lead to more fruitful results.

CHAPTER 7
SEARCHER FRUSTRATION DETECTION

In this chapter, we investigate detecting searcher frustration. In Chapter 1 we defined searcher frustration as the self-reported level of frustration a user experiences while engaged in a search. There are many reasons why one would like to have a model able to detect how frustrated a user is. For example, a retrospective frustration classification is useful as a system effectiveness measure. In a real-time setting, automatic detection could be used to trigger system interventions to help frustrated searchers, hopefully preventing users from leaving for another search engine or abandoning the search altogether. Depending on the level of frustration, we may wish to change the underlying retrieval algorithm or user interface. For example, one source of difficulty that could cause frustration in retrieval is a user's inability to sift through the results presented for a query (Lawrie, 2003; Xie & Cool, 2009; Kong & Allan, 2013). If the system can detect that the user is frustrated, it could offer an alternative interface, such as on that shows a conceptual breakdown of the results: rather than listing all results, group them based on the key concepts that best represent them (Lawrie, 2003) or attempt to diversify by other attributes (Aktolga & Allan, 2013). Using a well worn example, if a user enters 'java', they can see the results based on 'islands', 'programming languages', 'coffee', etc. Adaptive systems based on user models have been used elsewhere in IR. For example, White et al. (2006) used implicit relevance feedback to detect changes in users' information needs and alter the retrieval strategy based on the degree of change.

Our goal is to detect whether a user is frustrated at the end of each search. Recall that we define a search as a query and all of the search behavior occurring afterwards until either the next query or the end of the session is encountered. At the end of a search, we ask, "Is the user frustrated at this point of the search task?" To detect frustration, we derive features from the search just completed as well as other searches conducted in the task so far. We refer to these feature sets as search and task features, respectively. We compute additional features from a user's other tasks, which we call user features.

In this chapter, we consider frustration detection a binary task. However, multi-class detection may be useful, such as the intensity or type of frustration, e.g., detecting each of the fifteen types of frustration outlined by Xie and Cool (2009). We leave multi-class detection for future work.

In our previous work (Feild, Allan, & Jones, 2010), we found the most effective model for detecting searcher frustration was one based on the features that White and Dumais (2009) found were useful for predicting when users would switch search engines. The features in the model are: the most recent query's length in characters, the average token length of the most recent query, the duration of the task in seconds, the number of user actions in the task, and the average number of URLs visited per task for the current user.

In this chapter, we extend our previous work to consider the effects of personalization and privacy on detection quality. We pose the following questions regarding searcher frustration:

RQ1. How does personalization affect the quality of detection?

RQ2. What impact does sanitizing global data have on the quality of detection?

To address these questions, we use the best performing model from our previous work and extend it to include personalization. We also consider the baselines of always or never considering the user frustrated. We then explore sanitization by using the FT_a, FT_u, DP_a, DP_u, and ZEALOUS privacy mechanisms to produce global models of searcher frustration. We use a data set collected during a user study with 30 participants (also used in our previous work).

Our key contributions are (1) an analysis of personalized searcher frustration detection and (2) an analysis of the effect of sanitized global data on searcher frustration detection.

Figure 7.1 outlines the implementation we will follow for personalized frustration detection throughout this chapter. In Section 7.1 we describe how global data will be mined, collected and processed. We then describe how local data is used and combined with global data in Sections 7.2 and 7.3. We present our experimental setup and describe the data we use in Section 7.4. We present our findings in Section 7.5 before wrapping up with a discussion of limitations in Section 7.6 and a summary in Section 7.7.

7.1 Global data

To extract global data, we must first ask users to explicitly inform us when they are frustrated. When we detect that a user is ending a search (i.e., they have moved on to a new search), we ask them if they were frustrated with the search. This annotation is then coupled with a feature vector for the search. As mentioned previously, the feature vector includes the most recent query's length in characters, the average token length of the most recent query, the duration of the task in seconds, the number of user actions in the task, and the average number of URLs visited per task for the current user. The last feature can be calculated at the time of the annotation or computed just prior to uploading the data, giving a better estimate of the user's behavior.

Personalized frustration detection

▷ **Preprocessing**

1. *What artifacts will be extracted?*
   ◇ Feature vectors generated from a user's search history, each corresponding to a search conducted by the user; features are binary and each vector is annotated as *frustrated* or *not-frustrated* by the user (see Section 7.1 for details).
2. *What preprocessing will take place on the global data?*
   ◇ A logistic regression model will be trained (see Section 7.1 for details).
3. *What preprocessing will take place on the local data?*
   ◇ A logistic regression model will be trained on the raw feature vectors for any instances that have been annotated by the user (see Section 7.2 for details).

▷ **On input**

4. *What is the input?*
   ◇ A feature vector for an un-annotated search.
5. *What request will the client send to the server?*
   ◇ A request for the globally-trained logistic regression model (no data is sent).
6. *What processing will the server perform for the client request?*
   ◇ None.
7. *What response will the server send to the client?*
   ◇ The trained logistic regression model.
8. *How will the server data be integrated with the local data?*
   ◇ The score produced from the local logistic regression classifier will be multiplied with the score produced from the global logistic regression classifier (see Section 7.3 for details).
9. *What is the output of the application?*
   ◇ A boolean: *frustrated* or *not-frustrated.*

Figure 7.1. Implementation specifications for personalized frustration detection.

Because all of these features are continuous, it would be very difficult to create a sanitized data set. To overcome this issue, we binarize the features. One way to do this is to find the mean of a feature across either a user or a group of users and use that as the threshold. User-based means are more private since no data has to be aggregated across users, and moreover, as we will show, it actually out-performs using a group-level mean.

To train a detection model, we rely on logistic regression. This is appropriate since we have a binary classification problem: frustrated/not-frustrated. Other methods would likely work as well, such as decision trees or random forests.

7.2 Local data

We treat local data very similarly to global data. First, it is collected in the same way, i.e., we ask users to explicitly provide frustration labels to search instances. However, we do not binarize the feature values. In experiments on a development set, we found that raw features provide more useful detection than binarized ones.

Just as in the global model, we train a logistic regression model. This is easily implemented via JavaScript and so can be included in a platform such as CrowdLogger to be trained client-side.

7.3 Integrating local and global data

There are many ways to combine local and global data for classification models, as we saw in Chapter 5. For a particular test instance and a global and local score, we consider four integration methods: maximum, minimum, mean, and product. Using a development set, we found the product to give the best results when smaller amounts of local data were used, while performing similarly to other models when higher levels of local data were used. For that reason, we concentrate on the product. The variant of logistic regression we use outputs scores in the range [0,1], and so the product of any two scores also falls within that range. We use a threshold of 0.5, which means that for an instance to be classified as *frustrated*, at least one the local or global scores must be high (e.g., both equal to $\sqrt{0.5} = 0.71$ or one score equal to 1 and the other 0.5).

7.4 Experimental setup

In this section, we outline our experiments, the data we use, and our method of evaluation.

7.4.1 Experiments

To answer our two research questions, we consider three experiments.

Experiment 1. Our first research question asks how personalization affects frustration detection. To answer this, we look at the performance of using a local model built from n user instances and then testing on several other instances from that user. We vary n from 1 to the maximum possible number. For our experiments, we will randomly sample n labeled user instances and perform r trials per value of n. The models to compare against are: always assume the user is frustrated, never assume the user is frustrated, and a global model. If personalization helps, we expect the local model to out-perform the others.

Experiment 2. Our second experiment extends the first and evaluates the effect of integrating local and global data. We look at several methods of combining the two sources of data, varying n just as in Experiment 1. As we mentioned in Section 7.3, our analysis focuses on the product of the local and global scores for a given instance, but we also considered taking the maximum, minimum, and mean of the two scores. If integrating local and global information is beneficial, we expect the integrated model to outperform the models examined in Experiment 1.

Experiment 3. Our final experiment explores the effect of sanitization on frustration detection. We consider all privacy mechanisms: FT_a, FT_u, DP_a, DP_u, and ZEALOUS. We then show the performance of the sanitized global and integrated models applying the same procedures used in Experiments 1 and 2. One issue we must address has to do with our data size: there are too few users for the sanitization methods to work well. To overcome this, we created a synthetic pool of data using the following procedure, varying i from 1 to N: 1) randomly select a user; 2) randomly sample with replacement $m = [1, 50]$ instances from that user to create a new pseudo user u_i; 3) add u_i to the set of pseudo users U. The instances added in step 2 are raw, meaning that the average required for binarization is computed over the pseudo user, not the original user. If sanitization is harmful, the performance of the global and integrated models should drop compared to the unsanitized global models used in the previous experiments.

7.4.2 Data

In Fall 2009, we conducted a user study with thirty participants from the University of Massachusetts Amherst. The mean age of participants was 26. Most participants were computer science or engineering graduates, others were from English, kinesiology, physics, chemical engineering, and operation management. Two participants were undergraduates. Twenty-seven users reported a '5' (the highest) on a five-point search experience scale; one reported a '4' and two a '3'. There were seven females and twenty-three males.

Each participant was asked to complete seven[1] tasks from a pool of twelve (several with multiple versions) and to spend no more than seven minutes on each, though this was not strictly enforced. The order of the tasks was determined by four 12×12

[1] The first two participants completed eight tasks, but it took longer than expected, so seven tasks were used from then on.

1. What is the average temperature in [Dallas, SD/Albany, GA/Springfield, IL] for winter? Summer?
2. Name three bridges that collapsed in the USA since 2007.
3. In what year did the USA experience its worst drought? What was the average precipitation in the country that year?
4. How many pixels must be dead on a MacBook before Apple will replace the laptop? Assume the laptop is still under warranty.
5. Is the band [Snow Patrol/Greenday/State Radio/Goo Goo Dolls/Counting Crows] coming to Amherst, MA within the next year? If not, when and where will they be playing closest?
6. What was the best selling television (brand & model) of 2008?
7. Find the hours of the PetsMart nearest [Wichita, KS/Thorndale, TX/Nitro, WV].
8. How much did the Dow Jones Industrial Average increase/decrease at the end of yesterday?
9. Find three coffee shops with WI-FI in [Staunton, VA/Canton, OH/Metairie, LA].
10. Where is the nearest Chipotle restaurant with respect to [Manchester, MD/Brownsville, OR/Morey, CO]?
11. What's the helpline phone number for Verizon Wireless in MA?
12. Name four places to get a car inspection for a normal passenger car in [Hanover, PA/Collinwood, TN/Salem, NC].

Table 7.1. The information seeking tasks given to users in the user study. Variations are included in brackets.

Latin squares, which removed ordering effects from the study. Users were given tasks one at a time, so they were unaware of the tasks later in the order. Most of the tasks were designed to be difficult to solve with a search engine since the answer was not easily found on a single page. The complete list of tasks is shown in Table 7.1.

The study relied on a modified version of the Lemur Query Log Toolbar[2] for Firefox.[3] To begin a task, participants had to click a 'Start Task' button. This prompted them with the task and a brief questionnaire about how well they understood the task and the degree to which they felt they knew the answer. They were asked to use any of four search engines: Bing, Google, Yahoo!, or Ask.com and were allowed

[2] http://www.lemurproject.org/querylogtoolbar/

[3] http://www.mozilla.com/en-US/firefox/firefox.html

to switch at any time. Links to these appeared on the toolbar and were randomly reordered at the start of each task. Users were allowed to use tabs within Firefox.

For every query entered, users were prompted to describe their expectations for the query. Each time they navigated away from a non-search page, they were asked the degree to which the page satisfied the task on a five point scale, with an option to evaluate later. At the end of a search (determined by the user entering a new query or clicking 'End Task'), users were asked what the search actually provided relative to their expectations, how well the search satisfied their task (on a five point scale), how frustrated they were with the task so far (on a five point scale), and, if they indicated at least slight frustration (2–5 on the five-point scale), we asked them to describe their frustration.

When users finished the task by clicking 'End Task', they were asked to evaluate, on a five point scale, how successful the session was, what their most useful query was, how they would suggest a search engine be changed to better address the task, and what other resources they would have sought to respond to the task.

A total of 211 tasks were completed (one participant completed one fewer task because of computer problems), feedback was provided for 463 queries, and 711 pages were visited. On the frustration feedback scale, '1' is *not frustrated at all* and '5' is *extremely frustrated*. In Table 7.2 we see that users collectively reported some level of frustration for about half of their queries. The most common reasons given for being frustrated were: (1) off-topic results, (2) more effort than expected, (3) results that were too general, (4) un-corroborated answers, and (5) seemingly non-existent answers.

For our experiments, we selected 20 users for training and development and the remaining 10 users for final testing—these are the same training and test sets used in our previous work. We split each user's data into two parts: testing and local training. The local training partition consists of search instances for the first four

Query Frustration	None				Extreme
Feedback value:	1	2	3	4	5
Frequency:	235	128	68	25	7
Percentage:	51%	28%	15%	5%	1%

Table 7.2. Distribution of user-reported frustration for searches.

tasks the user performed. The testing partition contains the other three tasks. One of the factors we vary is the number of instances used from the local training partition. To binarize the frustration levels, we map the frustration level of 1 to *not-frustrated* and all other levels to *frustrated*.

Before building the final models, we conducted an exploratory analysis on the training and development set using a leave-one-user-out approach, training global models over 19 users and testing on the remaining user. This allowed us to settle on parameters for the models we ultimately tested. When a user is part of the global training set, data from both the local training and testing partitions are used. The final models were trained over the entire training set and evaluated on the 10-user test set.

The statistics for the exploratory and final evaluation phases of our experiments are shown in 7.3. In the final evaluation, the maximum number of instances within any of the ten local training sets is twelve.

7.4.3 Evaluation

In this section, we describe the metrics that we use to evaluate frustration models. Our ultimate goal is to use frustration models to decide when to intervene to help the user during the search process. Since many interaction methods with which we would like to intervene are not typically used because of their undesirable, frustration-causing attributes (i.e., interaction and latency), we are interested in minimizing our false-positives (non-frustrated searchers that our models say are frustrated), potentially at the cost of recall. For that reason, our predominant evaluation metric is a

Exploratory phase on development set		
Training users	19	
Training instances	287–313	
(labeled *frustrated*)	151–165	(49–53%)
Total local training instances	184	
(labeled *frustrated*)	108	(59%)
Evaluation users	20	
Evaluation instances	139	
(labeled *frustrated*)	57	(41%)

Final evaluation		
Global training users	20	
Global training instances	324	
(labeled *frustrated*)	165	(51%)
Total Local training instances	77	
(labeled *frustrated*)	33	(43%)
Evaluation users	10	
Evaluation instances	62	
(labeled *frustrated*)	29	(47%)

Table 7.3. Statistics about the data used in the exploratory phase over the 20 training/development users (top) and for the final training and evaluation (bottom).

macro (across users) F-score with $\beta = 0.5$, which gives increased weight to precision over recall. Specifically, we calculate $\mathrm{F}_{0.5}$ using the average over the precision and recall values calculated for each user, rather than using the average $\mathrm{F}_{0.5}$ calculated per user. Using a macro rather than a micro approach avoids one frustrated searcher with many search instances in the test data skewing the results. Un-weighted macro-averaging treats all users equally. A desirable model is one that performs well across all users, not just on one specific user.

Formally, precision, recall, and $\mathrm{F}_{0.5}$ are defined as follows. First, assume that we have four counters that keep track of the number of instances we: correctly classify as *frustrated* (true positives, TP); incorrectly classify as *frustrated* (false positives, FP); correctly classify as *not-frustrated* (true negatives, TN); and incorrectly classify as *not-frustrated* (false negatives, TN). Then,

$$\text{Precision} = TP/(TP + FP), \tag{7.1}$$

$$\text{Recall} = TP/(TP + FN), \tag{7.2}$$

$$\text{F}_\beta = \frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}. \tag{7.3}$$

In the cases where a user has no instances of frustration, we set recall to 1.0; if a model classifies all of a user's instances as *not-frustrated*, we set precision to 1.0. These defaults avoid edge cases in which undefined values are possible, and do so in a practical manner.

Our evaluation is off-line. When calculating features such as the average number of URLs visited per task for a given user, we average over all tasks, including those in the test partition. If this were an on-line evaluation, then user features could be calculated up until the searching being tested. We chose the off-line option due to the limited data—only seven or eight tasks per user. Given more user data (unlabeled, even), we believe that the two evaluations would be nearly equivalent.

7.5 Results and analysis

For Experiments 1 and 2, we first turn to Figure 7.2. In this figure, we show the performance of various models as a function of the number of local training instances on the training and development set of 20 users. For the local and integrated (local×global) models, 10 samples are shown for each level of training instances used. We can see that as more instances are used for local training, performance becomes less variable, though part of this is that there are only so many instances contributed by each user. While the local model by itself is not very stable, especially at lower levels of local training data, it on average out-performs the global model. Integrating local and global scores appears to provide greater, though not statistically significant, improvements over using either local (for small levels of local training data) or global data by itself.

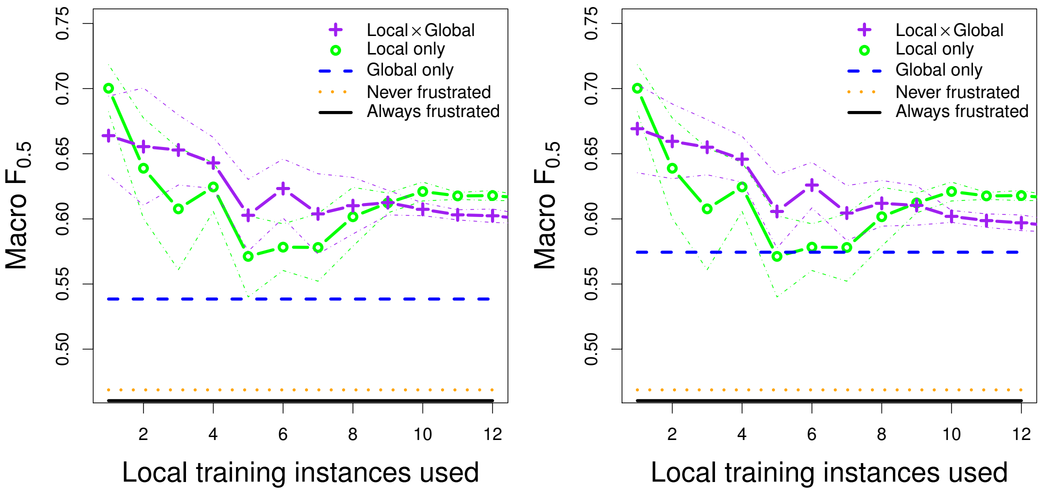


Figure 7.2. The performance of various models as a function of the number of local training instances used on the training set of 20 users. On the left, we use a globally calculated mean to binarize the global feature vectors; on the left, we rely on user-level means. The local and integrated model lines are an average over 10 samples per x value, with dashed lines representing the 95% confidence intervals for the mean.

Figure 7.2 demonstrates the effect that using means of global-level features for binarization has on the performance of the global only model. We had originally assumed that global-level means (left graph) would increase performance. However, it turns out that user-level means (right graph) perform substantially better. This is useful to know as collecting a global mean is more challenging when privacy is taken into account

In Figure 7.3, we show the performance of the various models on the test set of 10 users. Performance is quite different from that seen with the training and development set. For instance, the macro $F_{0.5}$ substantially jumped for all models. Also, the advantage of local data over global is missing for lower levels of local training data and is much less pronounced for higher values. We see an upward trend rather than a downward trend for the local and integrated models. Finally, the integrated model always out-performs the local model, not just when there is less local training

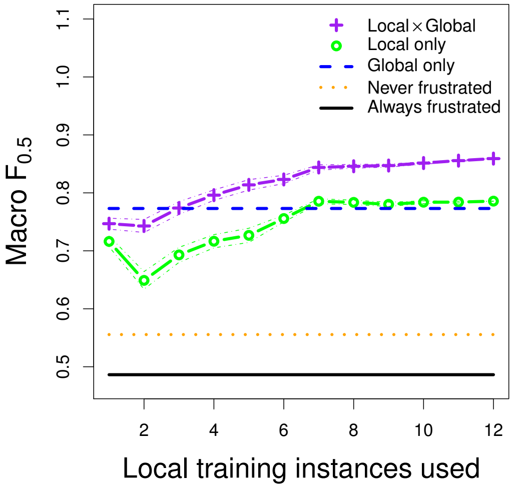


Figure 7.3. The performance of various models as a function of the number of local training instances used on the test set of 10 users. The global model was trained on the 20 training users and feature binarization relied on user-level means. The local and integrated model performance lines are an average over 100 samples per x value, and the dotted lines indicate the 95% confidence interval of the mean.

data as was the case in Figure 7.2. It seems that, even with small amounts of local training data—four or more training instances—it is advantageous to integrate local and global score to achieve the highest performance. When all the local training data is used, the integrated model achieved a macro F score of 0.86—an almost 9% relative improvement over the local-only model (F= 0.79) and a 12% relative improvement over the global-only model (F= 0.77). The differences between the mean performance of models is statistically significant (t-test, $p < 0.001$). However, a Fisher's Randomization test only showed significant differences in macro $F_{0.5}$ between each model and the baselines (always frustrated and never frustrated) when all local training data was used (the performance reported at the far right of the plot). This may be due to in part to the small scale of the test set.

For Experiment 3, we constructed a large pseudo user collection following the procedure outlined in Section 7.4. This resulted in 100,000 pseudo users and 2,499,416 feature vectors, 51% of which have the *frustrated* label. All 64 distinct feature vectors—2 values raised to the (5 features + 1 label) = 2^6 = 64—are present in the pseudo user set. Interestingly, this resulted in the same performance as in the leave-one-user-out evaluation, validating our technique.

For the differentially privacy mechanisms, we use $N = 100,000$, $\epsilon = \ln(10)$, and $\delta = 1/100,000$. We considered several values of d ($d = \{1, 2, 4, 8, 16, 32, 64\}$) and took 10 samples at each level. For the frequency thresholding models, we looked at several values of k ($k = \{2, 5, 10, 50\}$). We found no effect from the privacy mechanisms on performance. The artifact space is small and even at the high threshold required for $d = 64$ under ZEALOUS, 57 of the 64 distinct artifacts were released. While the impression coverage changed, the coverage of distinct artifacts remained high for all levels of d and k. This result suggests that frustration can be modeled privately for large user bases.

7.6 Limitations

The data used in this chapter is both small and biased. Among the biases is the overlap in tasks given to users—tasks in the training set overlap with tasks in the test set. However, the coarseness of the feature values used in the global models may mitigate this to some extent. Nonetheless, an in-the-wild experiment would provide valuable insights into the generalizability of the results found here.

The small sample size makes it difficult to know how representative the results are. Large differences appeared between the cross-validation experiments on the training and development set and the final experiments on the test set. A larger study is needed to better understand the impact of these models in a real system.

We analyzed one global model here based on the best performing classifier from previous work. There are many other models, such as ones that rely on additional features of the type we used, which would increase the feature space. Other models rely on sequences of actions, such as "SCVC", meaning the user issued a search, click on a result, viewed the page, and then clicked on a link. Further work is needed to understand the effects that sanitization has on these and similar models.

7.7 Summary

We considered two research questions: 1) how does personalization affect frustration detection and 2) how does sanitization affect frustration detection. We considered local data and global data by themselves as well as several integrated models. Using data from a user study we conducted for previous work, we demonstrated that local data can add substantial and statistically significant benefits. We also found that, using a simulated training set of 100,000 pseudo users, privacy mechanisms have no effect on the performance, even for large thresholds. For a practitioner interested in deploying a privacy-preserving frustration detection system, our results suggest that they gather labels from many users and use the integrated model to detect frustration for users that have provided labels, and the global only model for users without annotations.

CHAPTER 8
CROWDLOGGER

In this chapter, we describe CrowdLogger, an open source browser extension for Firefox and Google Chrome.[1] To evaluate the applications we described earlier in this thesis, we used a combination of the AOL search logs, TREC data, and data collected during a laboratory study. A complimentary evaluation, which we leave for future work, is to collect data from users in-situ and have them use the applications we described. In order to launch such studies, we require a platform on which to conduct them. To this end, we have implemented Crowdlogger, which serves as a generalized platform for mining and collecting user data (privately or not) both passively and interactively, prototyping IR applications, evaluating IR applications, and conducting user studies.

CrowdLogger works as follows. Consenting participants download the browser extension, which then provides them with a list of available *Apps* and *Studies*, collectively known as *CrowdLogger Remote Modules*, or *CLRMs*. CLRMs are JavaScript and HTML code modules that provide a service, such as an IR application prototype (in the case of Apps), or collect data for research (in the case of Studies). They have access to an application programming interface (API) provided by CrowdLogger. This API, described in Section 8.1, allows CLRMs to access user data, store new data, interact with users, and upload data. Figures 8.1, 8.2 and 8.3 show screen shots of the extension in action.

[1] Source code: `https://code.google.com/p/crowdlogger/`; Web site: `http://crowdlogger.cs.umass.edu/`

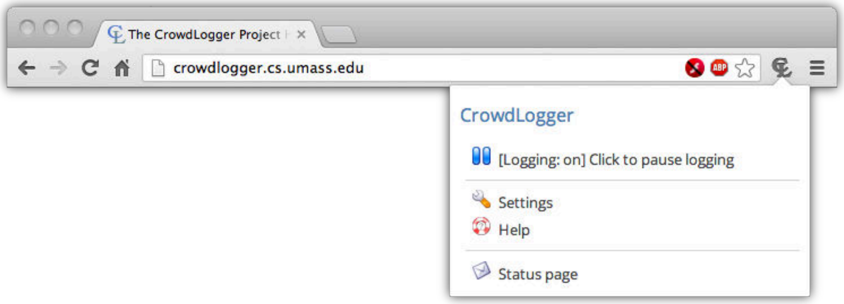
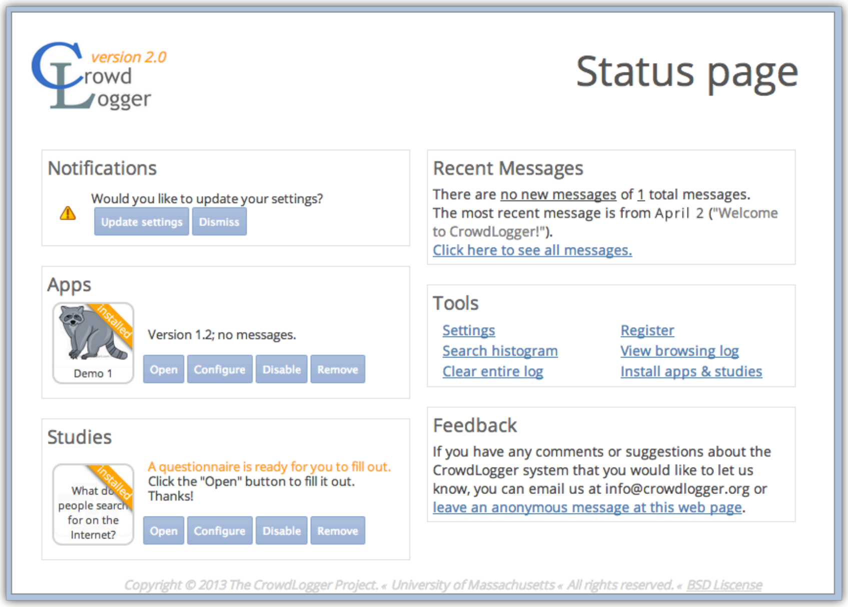


Figure 8.1. A screen shot of the CrowdLogger menu in Google Chrome. It is non-obtrusive and lets users quickly toggle logging on and off.

From a researcher's point of view, CrowdLogger provides an easy way to prototype a new system, evaluate it using real user data, and conduct studies over a potentially large number of users. CrowdLogger takes care of logging, data input/output, server communication, and pre-defined privacy policies. We implemented CrowdLogger with the goal of maintaining a large user base, which can be shared by multiple researchers conducting independent studies. If users downloading CrowdLogger for a particular study decide to keep it installed, they might see another study advertised in the CLRM listings and elect to participate in that one, as well. Our hope is that the CrowdLogger user network will provide researchers with a pool of potential participants for user studies, in addition to those they would normally advertise to.

For the purposes of this thesis, the important aspects of CrowdLogger are as follows. First, it implements the CrowdLogging framework, allowing for the private aggregation of user data under several definitions of privacy. Second, it allows client-side logging and user modeling. Third, it allows for user interaction, e.g., users can be asked to label their data. Fourth, it allows for IR applications to be evaluated either implicitly through user behavior or explicitly through feedback from the user. Earlier in this thesis, we described three IR applications; while we did not conduct user studies with CrowdLogger to evaluate these applications, we have either implemented them as a Study module or they could be implemented as one.



Status page

Notifications

Would you like to update your settings?

Recent Messages

There are no new messages of 1 total messages. The most recent message is from April 2 ("Welcome to CrowdLogger!"). Click here to see all messages.

Apps

Version 1.2; no messages.

Tools

Settings
Register
Search histogram
View browsing log
Clear entire log
Install apps & studies

Studies

What do people search for on the Internet?

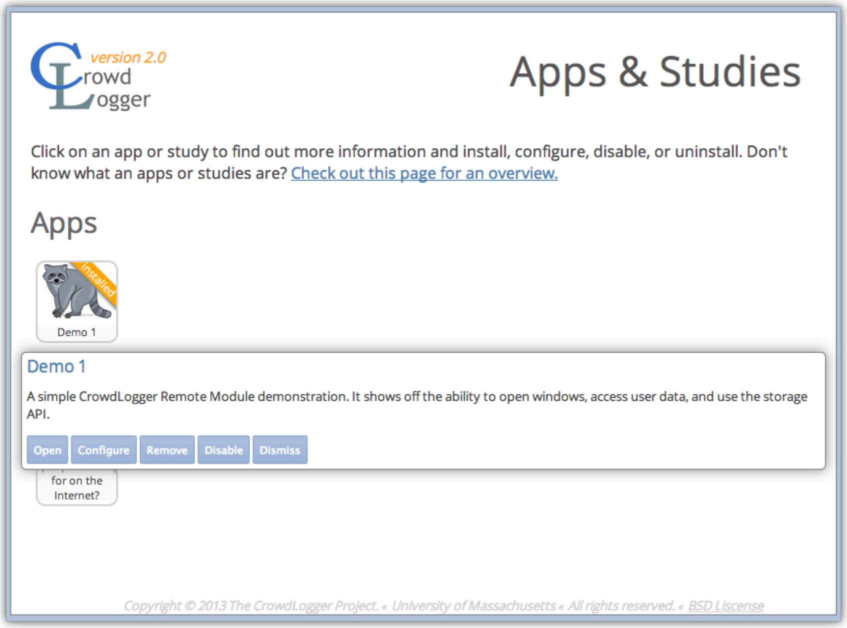
A questionnaire is ready for you to fill out. Click the "Open" button to fill it out. Thanks!

Feedback

If you have any comments or suggestions about the CrowdLogger system that you would like to let us know, you can email us at info@crowdlogger.org or leave an anonymous message at this web page.

Copyright © 2013 The CrowdLogger Project. • University of Massachusetts • All rights reserved. • BSD Liscense

Figure 8.2. A screen shot of the CrowdLogger Status Page. This is where users can set their preferences, find notifications from researchers, update CLRMs, and manage their search history.



Apps & Studies

Click on an app or study to find out more information and install, configure, disable, or uninstall. Don't know what an apps or studies are? Check out this page for an overview.

Apps

Demo 1

A simple CrowdLogger Remote Module demonstration. It shows off the ability to open windows, access user data, and use the storage API.

Copyright © 2013 The CrowdLogger Project. • University of Massachusetts • All rights reserved. • BSD Liscense

Figure 8.3. A screen shot of the CrowdLogger App and Study page, listing CRLMs available for install.

Other tools exist for performing some of the functionalities encapsulated in CrowdLogger. For instance, the Lemur Query Log Toolbar[2] was developed and released with the goal of collecting a sizable search log for use within the IR research community. Implemented as a Firefox and Internet Explorer extension, it logged many user interactions, including web search queries, viewed page text, copied and pasted text and scrolls. This data was uploaded once a week to a centralized location. The primary purpose of the Lemur Toolbar was to collect data and it did not allow interactions with users as CrowdLogger does. Also, the only notion of privacy in the Lemur Toolbar was that each week's worth of data was stored under a unique, anonymized user identifier. Another piece of software similar in spirit to CrowdLogger is the HCI Browser introduced by Capra (2011). The HCI Browser was made to be used for interactive, task-based IR studies. While it does log user behavior similar to that logged by the Lemur Toolbar, it was not made to be a passive logger, as CrowdLogger allows. Both of these loggers are open source and can be modified by researchers for other ends, but doing so requires understanding the internals, reworking them, and then releasing the software to a new batch of participants. CrowdLogger allows extensions to be programmed as CLRMs, which can be installed, uninstalled, and updated on the fly and share a common user base. At the same time, since the software is open source, researchers can modify the underlying system to suit their particular needs (though they will lose access to the CrowdLogger user base), just as with the Lemur Toolbar and the HCI Browser.

The remainder of this chapter begins in Section 8.1 with an overview of the API exposed by CrowdLogger that allows researchers to leverage the CrowdLogging framework, implement IR application prototypes, perform user studies, and evaluate sys-

[2] `http://www.lemurproject.org/querylogtoolbar/`

tems. We describe some example CLRMs that we have implemented for CrowdLogger in Section 8.2. Finally, we end with a summary in Section 8.3.

8.1 CrowdLogger API

In this section, we outline the API exposed to developers in CrowdLogger. The API is meant to abstract many common functionalities as well as provide an easy way to provide privacy. There are several major API categories:

- User behavior (historic and real-time)
- Aggregate user data
- User interface
- Local storage
- Privacy
- Server-side access

We discuss each of these below.

User behavior. This API provides access to the locally stored user log, which includes the following user interactions: web searches on Google, Bing, and Yahoo! and the displayed results; page loads, focuses, and blurs (when the page loses focus); clicks on search page results; clicks on links; browser tab additions, removal, and selections; and logging status (when logging is turned on and off). In addition to having access to the user's interaction history, the API also provides a function that CLRMs can use to register for a real-time stream of user interactions. One example of using this feature is to prompt users with a questionnaire when they enter a new search.

Aggregate user data. This API gives CLRMs access to crowd logs, if they exist, or create new ones otherwise. The motivation is that if an IR application implemented as a CLRM requires global data, it can use this API function to access it if that data already exists on the CrowdLogger server or gather it otherwise. The API is agnostic to privacy policies, and therefore developers can use this function to access/create sanitized or unsanitized crowd logs.

User interface. One difficulty of implementing logging software is dealing with cross platform issues. As a Firefox and Google Chrome browser extension, CrowdLogger works across non-mobile Windows, Macintosh, and Linux/Unix system. However, programming for both Firefox and Chrome can be troublesome as CLRMs run on both browsers (there is no notion of a Firefox or Chrome only CLRM). To prevent developers from having to include many switches in their CLRMs to deal with Firefox or Chrome idioms, the user interface API provides wrapper functions to open windows, get favicons, and interact with CrowdLogger's messaging service.

Local storage. While CrowdLogger takes care of logging many important user interactions, CLRMs will likely still need to log information to a user's machine. This can be achieved with the local storage API. For example, a CLRM for gathering page quality annotations from users may want to save that information locally. Local storage is also necessary for maintaining state between browser restarts.

Privacy. This API provides a number of components used in CrowdLogging, including basic encryption, encryption with Shamir's Secret Sharing, and anonymized data uploading.

Server-side access. This API provides functions for communicating with a server to send data, receive data, or both. This does not access the CrowdLogger server, but rather allows communication between the CLRM and any arbitrary server that a developer needs access to. One reason for requiring server access is to upload user data. Another is to perform a server-side computation on some user data. Developers may also need access to crowd logs not stored on the CrowdLogger server.

	Query freq.		Query reform.		Query-click	
k	Impressions	Users	Impressions	Users	Impressions	Users
1	4905	4905	6194	6194	2816	2816
2	1803	46	1783	12	830	9
3	490	13	118	1	288	1
4	402	6	89	1	209	0
5	156	1	36	0	126	0

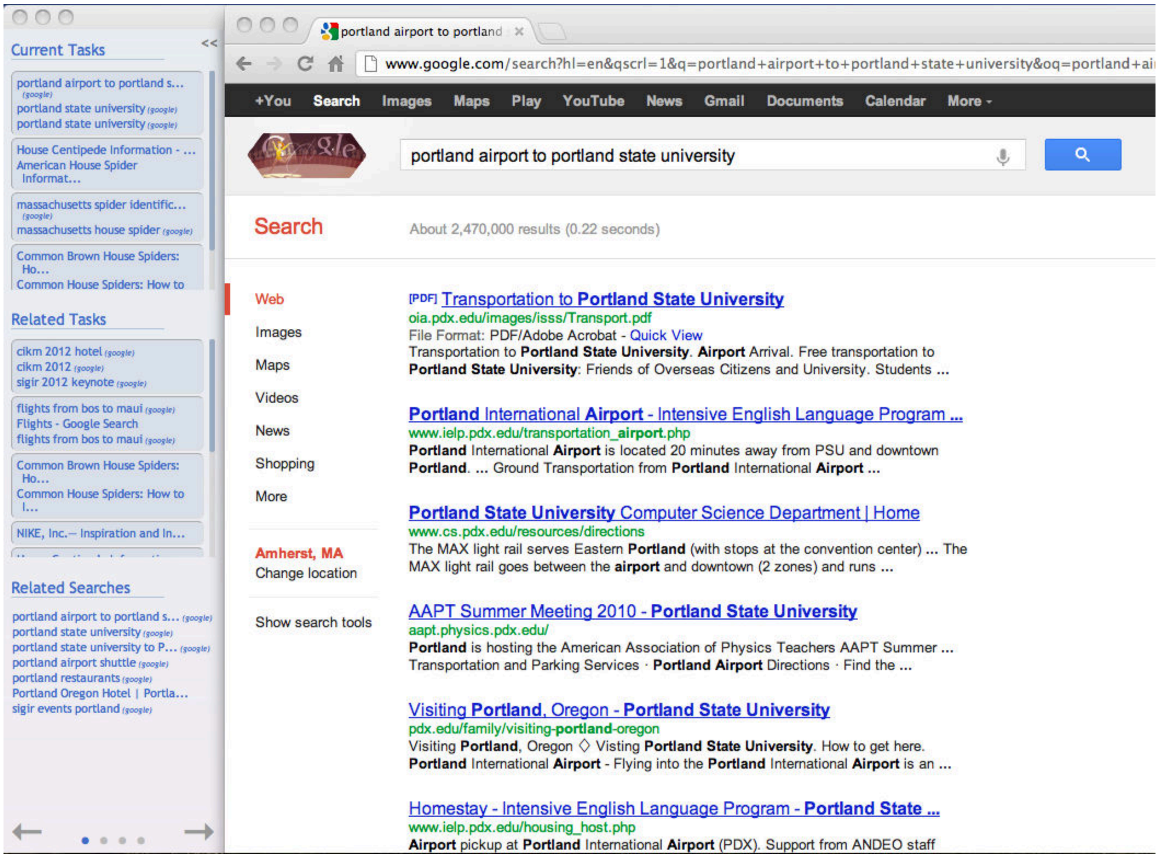
Table 8.1. Number of distinct queries, query pairs, and query-click pairs with at least k impressions or k users. Data was collected with CrowdLogger over two weeks in January 2011.

CLRMs are required to list the APIs they want access to so that users can be informed during the installation. For example, if a CLRM need server-side access, users should be aware as their data could be uploaded.

8.2 Examples

We have implemented two studies and one prototype using CrowdLogger. The first, which preceded the CrowdLogger API and CLRMs, evaluated the coverage of extracting queries, query pairs, and query-click pairs using different privacy policies in the CrowdLogging framework. We ran the study for two weeks in January 2011 with 16 anonymous users. Not much significant data was collected—only a single query was shared by at least five of the users—but the system implementation was a success. Table 8.1 shows the number of distinct queries, query pairs, and query-click pairs consisting of at least 1–5 instances or shared by at least 1–5 users. More details of the study can be found in our previous work (Feild et al., 2011).

We also implemented a prototype of a system called the Search Task Assistant (Feild & Allan, 2012), the left window in Figure 8.4. This system automatically groups a user's search history—queries and page visits—into search tasks. It is not quite as advanced as the search task identification techniques we describe later (there is no personalization yet), but it does allow users to re-group tasks, search their task


Current Tasks
<<
portland airport to portland s... (google)
portland state university (google)
portland state university (google)
House Centipede Information - ...
American House Spider Informat...
massachusetts spider identific... (google)
massachusetts house spider (google)
Common Brown House Spiders: Ho...
Common House Spiders: How to
Related Tasks
cikm 2012 hotel (google)
cikm 2012 (google)
sigir 2012 keynote (google)
flights from bos to maui (google)
Flights - Google Search
flights from bos to maui (google)
Common Brown House Spiders: Ho...
Common House Spiders: How to I...
NIKE, Inc.— Inspiration and In...
Related Searches
portland airport to portland s... (google)
portland state university (google)
portland state university to P... (google)
portland airport shuttle (google)
portland restaurants (google)
Portland Oregon Hotel | Portla...
sigir events portland (google)
portland airport to portland
www.google.com/search?hl=en&qscrl=1&q=portland+airport+to+portland+state+university&oq=portland+ai
+You
Search
Images
Maps
Play
YouTube
News
Gmail
Documents
Calendar
More
portland airport to portland state university
Search
Web
Images
Maps
Videos
News
Shopping
More
Amherst, MA
Change location
Show search tools

About 2,470,000 results (0.22 seconds)

[PDF] Transportation to Portland State University
oia.pdx.edu/images/isss/Transport.pdf
File Format: PDF/Adobe Acrobat - Quick View
Transportation to Portland State University. Airport Arrival. Free transportation to Portland State University: Friends of Overseas Citizens and University. Students ...

Portland International Airport - Intensive English Language Program ...
www.ielp.pdx.edu/transportation_airport.php
Portland International Airport is located 20 minutes away from PSU and downtown Portland. ... Ground Transportation from Portland International Airport ...

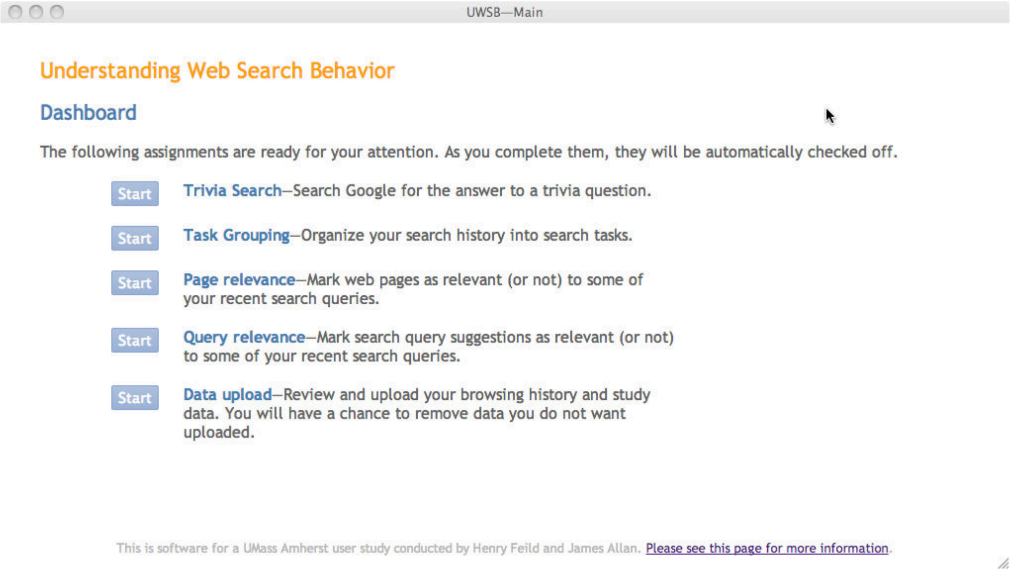
Portland State University Computer Science Department | Home
www.cs.pdx.edu/resources/directions
The MAX light rail serves Eastern Portland (with stops at the convention center) ... The MAX light rail goes between the airport and downtown (2 zones) and runs ...

AAPT Summer Meeting 2010 - Portland State University
aapt.physics.pdx.edu/
Portland is hosting the American Association of Physics Teachers AAPT Summer ... Transportation and Parking Services · Portland Airport Directions · Find the ...

Visiting Portland, Oregon - Portland State University
pdx.edu/family/visiting-portland-oregon
Visiting Portland, Oregon ◇ Visting Portland State University. How to get here. Portland International Airport - Flying into the Portland International Airport is an ...

Homestay - Intensive English Language Program - Portland State ...
www.ielp.pdx.edu/housing_host.php
Airport pickup at Portland International Airport (PDX). Support from ANDEO staff

Figure 8.4. A screen shot of the Search Task Assistant, an application implemented using the CrowdLogger API.



Understanding Web Search Behavior

Dashboard

The following assignments are ready for your attention. As you complete them, they will be automatically checked off.

Trivia Search—Search Google for the answer to a trivia question.

Task Grouping—Organize your search history into search tasks.

Page relevance—Mark web pages as relevant (or not) to some of your recent search queries.

Query relevance—Mark search query suggestions as relevant (or not) to some of your recent search queries.

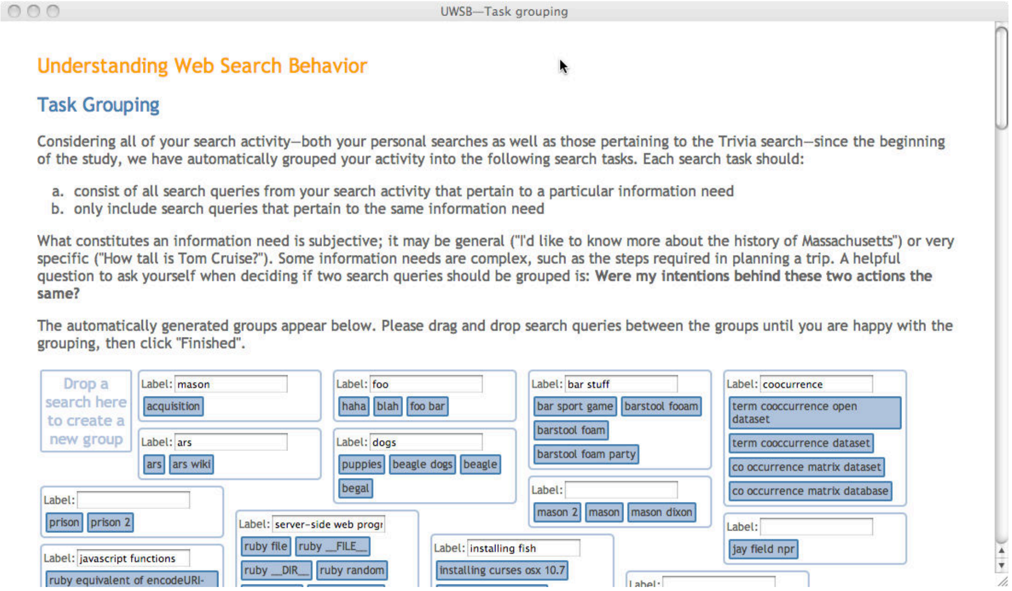
Data upload—Review and upload your browsing history and study data. You will have a chance to remove data you do not want uploaded.

This is software for a UMass Amherst user study conducted by Henry Feild and James Allan. Please see this page for more information.

Figure 8.5. A screen shot of the Understanding Web Search Behavior study, a CLRM implemented using the CrowdLogger API.

and search history, and displays tasks and searches that appear to be related to their current task. This implementation of the Search Task Assistant demonstrates the usefulness of CrowdLogger as a platform for prototyping. While CrowdLogger Apps are a great way for researchers to demonstrate a system, we believe they may also be used as an avenue to maintain users between studies. If CrowdLogger is providing a service to users, they may be less likely to uninstall it at the end of a study.

We have also implemented (but have not carried out) a study in CrowdLogger called Understanding Web Search Behavior. The goal of the study is to acquire user annotations for a number of IR applications. Specifically, groupings of search tasks and relevance labels for web pages and query recommendations. Screen shots are shown in Figures 8.5 and 8.6. The study makes use of many of the APIs, but does not consider privacy in the uploading phase; the University of Massachuestts Institutional Review Board protocol for the study allows us to collect raw data for our use only, but crowd logs can be released from the collected data. This demonstrates that



UWSB—Task grouping

Understanding Web Search Behavior

Task Grouping

Considering all of your search activity—both your personal searches as well as those pertaining to the Trivia search—since the beginning of the study, we have automatically grouped your activity into the following search tasks. Each search task should:

a. consist of all search queries from your search activity that pertain to a particular information need
b. only include search queries that pertain to the same information need

What constitutes an information need is subjective; it may be general ("I'd like to know more about the history of Massachusetts") or very specific ("How tall is Tom Cruise?"). Some information needs are complex, such as the steps required in planning a trip. A helpful question to ask yourself when deciding if two search queries should be grouped is: **Were my intentions behind these two actions the same?**

The automatically generated groups appear below. Please drag and drop search queries between the groups until you are happy with the grouping, then click "Finished".

Drop a search here to create a new group

Label: mason
acquisition

Label: ars
ars | ars wiki

Label: foo
haha | blah | foo bar

Label: dogs
puppies | beagle dogs | beagle | begal

Label: bar stuff
bar sport game | barstool fooam | barstool foam | barstool foam party

Label:
mason 2 | mason | mason dixon

Label: coocurrence
term cooccurrence open dataset | term cooccurrence dataset | co occurrence matrix dataset | co occurrence matrix database

Label:
jay field npr

Label:
prison | prison 2

Label: javascript functions
ruby equivalent of encodeURI-

Label: server-side web progr
ruby file | ruby __FILE__ | ruby __DIR__ | ruby random

Label: installing fish
installing curses osx 10.7

Label:

Figure 8.6. A screen shot of the task grouping assignment in the Understanding Web Search Behavior study. All of a user's searchers are automatically grouped and displayed in drag-and-drop interface. The user can then drag searches (the filled boxes) between task groups (the larger, unfilled boxes) to correct mistakes made by the automatic algorithm.

an interactive study can be implemented as a CLRM and that the platform allows flexibility in the study parameters, e.g., an empty privacy policy.

8.3 Summary

In the chapter, we introduced a novel system called CrowdLogger for collecting user data and performing user studies. The system implements the CrowdLogging framework, demonstrating the feasibility of CrowdLogging in a live system. CrowdLogger is cross-platform and open source, making it a good candidate for distributed research studies. We gave several examples of CrowdLogger in action, including a study of mining artifacts, an App called the Search Task Assistant, and a study for collecting user annotations.

CHAPTER 9
CONCLUSIONS

The two themes of this thesis are privacy and personalization. Personalization at the user level allows information retrieval system implementors the opportunity to tailor system behavior to each individual user, providing a unique, and hopefully superior, experience. In this thesis, we considered privacy an orthogonal issue, but one that is of increasing interest. Users want to protect their privacy while still receiving the benefits of a service, like web search. Researchers want to share data sets of user behavior with the research community at large, but must provide a policy for protecting user privacy in order to satisfy the demands of users, lawyers (in the case of industry researchers), and institutional review boards (in the case of academic researchers).

It is within this context that we considered ways for collecting data privately and the effects of personalization and privacy on three information retrieval applications. In this chapter, we summarize the contributions of this thesis and give an overview of future work.

9.1 Summary of contributions

We introduced a novel framework call CrowdLogging for collecting and mining search behavior from a distributed group of users in a privacy-preserving manner. This framework is useful for (1) researchers that must provide some mechanism to protect users for approval by their institutional review boards and to gain the trust of users, as well as for (2) large web search companies that want to release data to the research community. The framework logs user behavior on each user's computer, where the user has full control over it. Data is mined at the consent of users and sent encrypted to a centralized server. The framework depends on secret sharing, which allows the server to decrypt data only if it receives a sufficient number of distinct keys for a given piece of data. For example, CrowdLogging can be used to decrypt the set of queries that have been submitted by at least five users, without exposing queries entered by fewer than five users, resulting in sanitized data.

We considered several privacy mechanisms for sanitizing data that work within the CrowdLogging framework, formalizing two naïve mechanisms (FT_a and FT_u) and introducing a novel (ϵ, δ)-indistinguishable mechanism (DP_u). We described several weaknesses of each of the mechanisms, including an empirical experiment on the AOL query log data demonstrating the ability to infer unsupported artifacts using FT_a and FT_u. We explored the parameter settings necessary under the DP_a and DP_u mechanisms in order to achieve comparable query artifact coverage to that of FT_a and FT_u using the AOL search log. We found that the parameters settings, namely of ϵ, required to reach approximately the same coverage as frequency thresholding are unreasonable—ϵ values of 10 or more for values of $k < 200$. Our findings pertain to query artifacts released using the AOL search log and do not necessarily generalize to other data sets, though the process we use can be applied to other data sets and artifact classes.

We introduced a template for describing how local user data and global data aggregated from many users are collected, processed, and combined for use within an application. Leveraging local data, either by itself or in combination with global data, allows us to provide a personalized experience for each user. We looked at how personalization could be applied to three IR applications and analyzed the effect of both personalization and sanitization on performance.

The first application we explored was search task identification. We demonstrated the variability in individuals' perceptions of what constitutes a search task, finding a Fleiss' Kappa as low a 0.53 among six annotators' labels across ten user histories. This is the first such analysis in the search task identification literature that we know of. We gathered annotations for over 500 user histories extracted from the 2006 AOL search log, labeled by ten annotators—38 times as many user histories as used by the current state of the art research. With this data, we introduced several models for providing personalization, but found they perform similarly to using a non-personalized random forest classifier, all achieving between 94% and 95% macro accuracy across users. Our experiments showed the random forest classifier significantly out-performs the current state of the art model. We further demonstrated that sanitization has an overall mild effect on performance and can even improve performance under certain conditions, namely when the FT_u mechanism is used with $k = 100$.

The second application we considered was task-aware query recommendation, for which we introduced a novel model that combines query recommendations for each query within a task, giving more weight to queries based on either temporal distance or same-task likelihood. It relies on personalized search task identification, as described above. We found that leveraging on-task search context provides a large boost in MRR for many evaluation queries—more than 25% on average. However, our CrowdLogging approach to privacy has a substantial impact on recommendation performance, rendering the quality so low as to be useless in a real system in most cases we consider.

The third and last application we considered was searcher frustration detection—detecting when users become frustrated as they search for information. We explored the effects of personalization on frustration detection and showed that it can provide substantial performance improvements—as much as 9% in $\mathrm{F}_{0.5}$. We also demonstrate that with a simulated user base of 100,000 users performance of global models is not affected by sanitization.

Overall, we found mixed results for sanitization. When used for classification problems that rely on feature vectors as artifacts, sanitization has less of an impact (and can even boost performance in the case of STI). However, when artifacts cover a large space, such as query pair text used for query recommendation, sanitization has a substantial, negative impact on performance—at least when aggregating across 617,000 users. One possible way to reduce this effect is to modify the artifact representation, which we demonstrated can at the very least improve artifact coverage in crowd logs. Another is to aggregate over many more users, something that is difficult to achieve without access to a large-scale commercial product. If sanitization is still an issue even after modifying the representation and increasing the user base, it may be prudent to use an algorithm that does not depend on global data.

Finally, we described an open source system that implements the CrowdLogging framework. Available as a browser extension for Firefox and Google Chrome, it also serves as a platform with which to perform in-situ studies of user search behavior, evaluate IR applications, and provide research prototypes to interested parties. CrowdLogger allows researcher-defined JavaScript/HTML modules to be remotely loaded and exposes an API for: accessing a user's real-time or past browsing behavior; uploading data to a server privately, anonymously or in the clear (as approved by the user); interacting with the user via HTML and JavaScript; accessing a remote server for computational purposes; and saving data on a user's computer. We described two studies and one prototype that have been implemented with CrowdLogger.

9.2 Future work

There are many interesting avenues of future work stemming from this thesis. For example, using CrowdLogger to perform user studies for each of the IR applications we considered here. Such user studies would be valuable for several reasons. First, global data would be up-to-date, as opposed to the dated queries present in the AOL search log. Second, in situ studies allow us to measure the performance of our systems in their intended environment rather than simulations. Third, in user studies, users are the annotators of their own data, so we do not have to worry about third-party annotators guessing a user's intentions.

One of the limitations of some of the experiments we conducted is that the AOL search log is outdated and potentially biased (AOL was past its peak as a popular search engine by 2006). Experiments with more recent search logs would provide additional data points and allow us to draw more general conclusions from our results. One avenue for accomplishing this is to partner with major web search companies, such as Microsoft Bing or Google. Another is to conduct large scale user studies, e.g., with CrowdLogger.

In our analysis of task-aware query recommendation, we discovered that search task context can be both helpful and harmful; extending this work, it would be interesting to devise a method for automatically detecting when search task context will be helpful. Another direction is to consider the effects of using other recommendation algorithms besides term-query graphs—including methods that do not rely on search logs—and whether search context helps there, as well.

We found that privacy was prohibitively expensive in terms of utility for task-aware query recommendation. One hypothesis is that forming a term-query graph from a much larger log—millions or billions of users' worth of data—would allow for more reasonable performance when sanitized. The most likely path to accomplishing this would be to use a log from a large web search company.

When quantifying the effect of privacy on searcher frustration detection performance, we used a large simulated training set. An interesting question is whether the simulated training set is realistic. Future work should address this by obtaining frustration labels from a larger set of users. Other future work includes developing a multi-class detector that can classify the cause of a user's frustration within the search process. Doing so may allow adaptive system to intervene in helpful ways.

In considering privacy, we explored the coverage of privacy mechanisms over different artifact types. However, we did not perform an analysis of the content differences between the data released under various privacy mechanisms. It is possible that two mechanisms provide nearly identical coverage, but have substantial differences in terms of the artifact revealed. Extending this line of thought, it is also unclear what the relationship is between coverage and performance. That is, two data releases have large differences in content, but similar coverages, will their performance be drastically different, or very similar? We leave these explorations for future work.

When quantifying privacy, we relied on k for frequency thresholding and the theoretical bounds guaranteed under differential privacy, ϵ and δ. Future work should consider other means for discussing, qualifying, and quantifying privacy loss—what does it mean to a user that $\epsilon = \ln(10)$? What exactly are the risks associated with participating in a data collection with a given level of ϵ or k? Without lay descriptions, it is difficult to explain the risks to users, lawyers, and institutional review boards.

We only considered batch mining operations: given a search log spanning some amount of time, here is how it can be mined with respect to a privacy policy. However, one of our long-term goals is to mine data from a live stream of user interactions over a prolonged period of time. Future research questions surrounding this include how to sample from users and how to handle older data. The latter question is interesting because data that is relatively old—e.g., older than year—may be less sensitive or less correlated with a user's current search activity. These may have implications for the differentially private mechanisms we considered, the result of which may reduce the privacy loss associated with released data under those mechanisms.

Another question we leave for future work is how much data does one need for an application in order for privacy to not have a significant negative impact on effectiveness. From our experiments, we know that this number will vary across applications and artifact types—the effects of sanitization over feature vectors extracted from 352 users were mild for search task identification, while sanitizing query reformulations from 617,000 users provided virtually no utility for task-aware query recommendation. In addition, there are many kinds of artifacts that one can extract from user data and many applications that can use them; we have only scratched the surface in this thesis, and a good vein of future work is to establish how different classes of artifacts are affected by sanitization in different situations and how best to represent artifacts to maximize performance in a wider variety of applications. Taken together, a useful contribution would be a taxonomy of applications and artifact types with which an implementor could estimate the minimum number of users required in order to provide some level of performance.

There are many ways that personalization can be applied to a given information retrieval application. In this thesis, we only looked at certain ones. Future work should consider other methods of leveraging a user's local information to improve for the three applications we considered. For example, we could consider group-level personalization (the privacy implications of this would need to be explored).

We have several long-term goals for CrowdLogger, one of which is to encourage wide adoption of the platform. One of our intentions when designing CrowdLogger was that it be used by a large number of users for prolonged durations, allowing researchers to recruit participants from the existing user base. For this to happen, we need to demonstrate the utility of CrowdLogger. One way to demonstrate usefulness is to conduct a variety of user studies and remote evaluations of information retrieval system using CrowdLogger. Another is to ensure it is easy for participants to use and easy for researchers to develop Study and App modules. Future work includes further developing CrowdLogger to meet these requirements.

Appendices

APPENDIX A
GLOSSARY

Artifact: See *search log artifact.*

Crowd log: A histogram of artifacts extracted from one or more user logs.

Global model: A model built using data from a crowd log.

Information retrieval application (IR application): A component of an IR system. Examples include: ranking, re-ranking, relevance feedback, and query auto-completion. We specifically consider three additional applications: query recommendation, search task identification, and frustration detection. In the literature, this concept is often described as an *information retrieval task.* However, as a major piece of this dissertation concerns search tasks, we use the phrase *information retrieval application* to avoid confusion between the two.

Local model: A model built using data from a user log.

Personalization: The act of incorporating information about a user into the processing of an IR application.

Privacy policy: A protocol governing how artifacts must be treated with respect to user privacy to form a crowd log. If a privacy policy does not specify any actions to reduce privacy loss, we call it an *empty privacy policy.*

Sanitized global model: A global model built from a crowd log constructed with a non-empty privacy policy.

Search: A query and all of the search behavior occurring afterwards until either the next query or the end of the session is encountered (Feild, Allan, & Jones, 2010).

Search log: A set of entries, each describing a search event. Entries often include an identifier, a time stamp, an event descriptor, and additional information about the event.

Search log artifact (Artifact): A piece of information mined from a search log. An artifact can involve the composition of multiple entries within a log. Examples of artifacts include: individual queries, query-click pairs, query reformulations, and feature vectors.

Search session (Session): All search behavior—queries, URL clicks, page views, etc.—immediately following the end of the previous search session or the beginning of the log if no previous search session exists until there is a period of inactivity. We use a period of 26 minutes, as measured empirically by Lucchese et al. on a sample of the AOL search log (Lucchese et al., 2011).

Search task: A search task consists of one or more searches that share a common information need. In previous research, the terms *query chains* (Boldi et al., 2008; Radlinski & Joachims, 2005) and *goals* (Jones & Klinkner, 2008) are also used to describe tasks. Our definition is different from that used in the information seeking literature, which considers a task to consist of several facets, including its goal, source, and the process by which it is carried out (Li, 2009).

Searcher frustration: The level of agitation experienced by a user due to the mismatch of search expectations and outcomes.

Sufficiently supported: An artifact is sufficiently supported by a privacy policy if it exists in the sanitized crowd log.

User search log (User log): A search log in which all events share the same identifier. This assumes that identifiers correspond to individual users.

APPENDIX B
DETAILS OF DIFFERENTIAL PRIVACY

In this appendix, we provide the technical details for the three privacy mechanisms based on differential privacy: DP_u, DP_a, and ZEALOUS. In order provide a provably private mechanism, we can consider relaxations of differential privacy, such as *(ϵ, δ)-indistinguishability*, introduced by Dwork et al. (Dwork, Kenthapadi, et al., 2006). This is defined as follows:

Definition B.0.1 ((ϵ, δ)-indistinguishability (Dwork, Kenthapadi, et al., 2006))**.** A randomized algorithm $\mathcal{A}$ is (ϵ, δ)-indistinguishable if for all data sets D and D' that differ in at most one individual's data, and all $S \subseteq \text{Range}(\mathcal{A})$:

$$Pr[\mathcal{A}(D) \in S] \leq \exp(\epsilon) \cdot Pr[\mathcal{A}(D') \in S] + \delta.$$

With this relaxation, we have introduced a slack variable, δ, which allows more privacy to be leaked. Generally, it is recommended that $0 < \delta < \frac{1}{U}$, where U is the number of users contributing to the input data sets (Korolova et al., 2009).

Korolova et al. (Korolova et al., 2009) , Götz et al. (Götz et al., 2011), and we (Feild et al., 2011) established (ϵ, δ)-indistinguishable algorithms for releasing search log data, which we describe next. First, we introduce some vocabulary that will be used by the various algorithms. Let U be the number of users contributing to an input set, *Count*(a, A) be a function that counts the number of instances of a that occur in set A, *Count*$_u(a, A)$ be a function that counts the number of distinct

```
Algorithm 1: Release private query click graph (Korolova et al., 2009)
  Input  : A search log $D$; the maximum number of queries and clicks each user
           may contribute: $d, d_c$, respectively; Laplacian noise parameters:
           $b, b_q, b_c$ (these stand for: noise for thresholding on $k$, noise for releasing
           queries, and noise for releasing clicks); the query threshold, $k$.
  Output: A query-click graph.
1 Set $Q$ and $C$ to be the first $d$ queries and $d_c$ clicks from each user in $D$,
  respectively
2 $Q' \leftarrow \{q : \mathit{Count}(q, Q) + \mathit{Lap}(b) > k\}$
3 For each $q \in Q'$, output $\langle q, \mathit{Count}(q, Q) + \mathit{Lap}(b_q)\rangle$
4 For each URL $u$ in the top ten results for $q \in Q'$, output
  $\langle q, u, \mathit{Count}(u, C) + \mathit{Lap}(b_c)\rangle$
```

users that contributed a in set A, and $\mathit{Lap}(b)$ be a function that randomly samples a number from the Laplacian distribution with $\mu = 0$ and variance b.

The goal of the algorithm introduced by Korolova et al. is to release a sanitized query-click graph from a search log. The process is described in Algorithm 1.

We tailor this slightly to deal with arbitrary artifacts, allow any set of d artifacts to be used rather than the first d, and we remove the steps related to building a query click graph, yielding the artifact thresholding differential privacy algorithm (DP_a), shown in Algorithm 2. Assuming the experimenter sets the privacy parameters ϵ and δ, Korolova et al. suggest setting:

$$k = d\left(1 - \frac{\ln\left(\frac{2\delta}{d}\right)}{\epsilon}\right), \tag{B.1}$$

$$b = \frac{d}{\epsilon}. \tag{B.2}$$

This setting of k minimizes the noise added in step 3 of the algorithm and assumes that $\exp\left(\frac{1}{b}\right) \geq 1 + \frac{1}{2\exp\left(\frac{k-1}{b}\right)-1}$.

Götz et al. describe the ZEALOUS algorithm, which is described in Algorithm 3. Two important distinctions between the ZEALOUS and DP_a algorithms are that where DP_a allows each user to contribute d total artifacts and defines k in terms of

```
Algorithm 2: Artifact thresholding differential privacy ($DP_a$)
  Input : A set of artifacts, $D$; the maximum number of artifacts each user may
          contribute $d$; Laplacian noise parameters $b, b_r$ (these stand for: noise
          for thresholding on $k$ and noise for releasing); the artifact threshold $k$.
  Output: A histogram of artifacts with noisy impression frequencies.
1 Set $A$ to be $d$ artifacts submitted in $D$ by each user
2 $A' \leftarrow \{a : Count(a, A) + Lap(b) > k\}$
3 For each $a \in A'$, output $\langle a, Count(a, A) + Lap(b_r) \rangle$
```

the number of instances of an artifact, ZEALOUS restricts each user's contribution to d *distinct* artifacts and defines k in terms of the number of *distinct* users that contributed each artifact. ZEALOUS can achieve (ϵ, δ)-indistinguishability with the following settings:

$$b = \frac{2d}{\epsilon}, \tag{B.3}$$

$$k = d\left(1 - \frac{\log\left(\frac{2\delta}{d}\right)}{\epsilon}\right), \tag{B.4}$$

$$k' = 1, \tag{B.5}$$

where k, k', d, and b are defined as in Algorithm 3. Note that these settings are very similar to those proposed by Korolova et al. However, the noise parameter b is given twice the weight. This is because of how Götz et al. define the *sensitivity* of ZEALOUS. Let us take a moment to understand what sensitivity is and how it can be used. Dwork et al. (Dwork, McSherry, et al., 2006) define sensitivity as follows:

Definition B.0.2 (Sensitivity (Dwork, McSherry, et al., 2006))**.** The L_1 sensitivity of a function $f : D^n \rightarrow \mathbb{R}^m$ is the smallest number $S(f)$ such that for all search logs $D, D' \in D^n$ that differ in a single individual's d artifacts,

$$||f(D) - f(D')||_1 \leq S(f).$$

```
Algorithm 3: ZEALOUS (Götz et al., 2011)
  Input  : A set of search artifacts, $D$; the maximum number of distinct
           artifacts each user may contribute $d$; Laplacian noise parameter $b$; the
           user thresholds $k, k'$.
  Output: A histogram of artifacts with noisy user frequencies.
Set $A$ to be some set of $d$ distinct artifacts submitted in $D$ per user
$A' \leftarrow \{a : Count_u(a, A) > k'\}$
For each $a \in A'$, output $\langle a, u \rangle : u = Count_u(a, A') + Lap(b) \wedge u > k$
```

In the case of the algorithms listed above, f is a function that builds a histogram of artifacts and their counts. The confusion arises around how we define search logs that differ. If we think of D has having a particular user's data and D' as not, then the sensitivity of building a histogram is d, since a user's data can only effect at most d elements of the histogram. If, however, we consider D and D' to differ by the contents of the single user—rather than use one subset of d artifacts from the user, we use a *different* set of d artifacts—then the sensitivity of creating a histogram is $2d$, since in the edge case there is no overlap between the two sets of d artifacts. According to Dwork et al. (Dwork, McSherry, et al., 2006), the noise parameter b should be set as follows:

$$b = \frac{S(f)}{\epsilon}. \tag{B.6}$$

Thus, if we consider the latter variation of sensitivity, then we can update Equation B.2 to match Götz:

$$b = \frac{2d}{\epsilon}. \tag{B.7}$$

We introduced an algorithm called user thresholding differential privacy (DP_u) (Feild et al., 2011), a modification of the DP_a algorithm that maintains that each user can contribute d total artifacts, but where the threshold k is based on the number of users that contributed an artifact, as shown in Algorithm 4.

Algorithm 4: User-thresholding differential privacy (DP_u)

```
Input : A set of search artifacts, $D$; the maximum number of artifacts each
        user may contribute $d$; Laplacian noise parameters $b, b_r$ (these stand
        for: noise for thresholding on $k$ and noise for releasing); the user
        threshold $k$.
Output: A histogram of artifacts with noisy impression frequencies.
Set $A$ to be $d$ artifacts submitted in $D$ by each user
$A' \leftarrow \{a : \mathit{Count}_u(a, A) + \mathit{Lap}(b) > k\}$
For each $a \in A'$, output $\langle a, \mathit{Count}(a, A) + \mathit{Lap}(b_r) \rangle$
```

To prove that this technique is (ϵ, δ)-indistinguishable, we need only slightly alter a portion of the proof presented by Korolova et al. Specifically, we can update Equation 9 (Korolova et al., 2009), which in its original form[1] is as follows:

$$\frac{1}{2}\sum_{i=1}^{n_y} \exp\left(\frac{\mathit{Count}(y_i, D') - k}{b}\right) \leq \frac{d}{2}\exp\left(\frac{d-k}{b}\right), \tag{B.8}$$

where D' is a dataset with a single user added, versus D, which is the same dataset but with that user removed and $y_1, \ldots, y_{n_y}$ is the set of queries which are unique to D'. $\mathit{Count}(y_i, D')$ can return at most d, since a user can only contribute at most d artifacts. However, under the user frequency model, $\mathit{Count}_u(x, D)$ is used in place of $\mathit{Count}(x, D)$. Thus, $\mathit{Count}_u(y_i, D')$ is at most 1, since y_i is unique to the one user whose data is in D' but not D. This yields the following, tighter bound:

$$\frac{1}{2}\sum_{i=1}^{n_y} \exp\left(\frac{\mathit{Count}_u(y_i, D') - k}{b}\right) \leq \frac{d}{2}\exp\left(\frac{1-k}{b}\right). \tag{B.9}$$

This has ramifications for δ, giving us the following optimal user frequency threshold:

$$k = 1 - \frac{d\ln(\frac{2\delta}{d})}{\epsilon}. \tag{B.10}$$

[1] We have changed the notation slightly to be consistent with the notation used in this thesis.

DP_u has the same privacy guarantees as DP_a, but because of the change in the definition k, if we set k constant, d under the DP_u model is always the same or greater than d under the DP_a model. In short, DP_u allows for more artifacts to be contributed per user than with the DP_a algorithm. However, at the same time, the threshold criteria are tougher for DP_u—an artifact must be issued by k distinct users, not issued k times by potentially fewer than k users, as is allowed with the DP_a algorithm.

Because of their differences, each of these (ϵ, δ)-indistinguishable algorithms have different utility with respect to the number of artifacts they release given a common input search log D.

Going beyond (ϵ, δ)-indistinguishability, Götz et al. prove that with the right parameter configuration, the ZEALOUS algorithm is also *(ϵ, δ)-probabilistically differentially private* (Götz et al., 2011). This is a more conservative relaxation of differential privacy, yielding stronger privacy guarantees than (ϵ, δ)-indistinguishability, defined as follows:

Definition B.0.3 ((ϵ, δ)-probabilistic differential privacy (Götz et al., 2011; Machanavajjhala et al., 2008))**.** A randomized algorithm $\mathcal{A}$ is (ϵ, δ)-probabilistically differentially private if for all data sets D we can divide the output space Ω into two sets Ω_1, Ω_2 such that

$$Pr[\mathcal{A}(D) \in \Omega_2] \leq \delta,$$

and for all data sets D' that differ from D by at most one individual's data and all $S \subseteq \Omega_1$:

$$\exp(-\epsilon) \cdot Pr[\mathcal{A}(D') = S] \leq Pr[\mathcal{A}(D) = S] \leq \exp(\epsilon) \cdot Pr[\mathcal{A}(D') = S].$$

As Götz et al. explain, this definition says that $\mathcal{A}$ satisties ϵ-differential privacy with high probability. All of the outputs that breach ϵ-differential privacy are included in Ω_2, and the probability of an output being placed there is controlled by δ.

In ZEALOUS, (ϵ, δ)-probabilistically differential privacy can be attained using the following parameter configuration (Götz et al., 2011):

$$b \geq \frac{2d}{\epsilon}, \text{and}$$

$$k - k' \geq \max\left(-b \ln\left(2 - 2\exp\left(\frac{-1}{b}\right)\right), -b \ln\left(\frac{2\delta}{U \cdot d/k'}\right)\right).$$

In order to minimize the second threshold, k, the authors show that k' should be set as:

$$k' = \left\lceil \frac{2d}{\epsilon} \right\rceil.$$

REFERENCES

Adar, E. (2007). User 4xxxxx9: Anonymizing query logs. In *Proceedings of the Query Log Analysis Workshop, International Conference on World Wide Web.*

Aktolga, E., & Allan, J. (2013). entiment diversification with different biases. In *Proceedings of the 36th annual ACM SIGIR conference on Research and development in information retrieval.*

Barbaro, M., & Zeller Jr., T. (2006). A Face Is Exposed for AOL Searcher No. 4417749. *New York Times.* (`http://www.nytimes.com/2006/08/09/technology/09aol.html`)

Bar-Ilan, J. (2007). Position paper: Access to query logsan academic researchers point of view. In *Proceedings of the 16th International World Wide Web Conference.*

Bilal, D., & Kirby, J. (2002). Differences and similarities in information seeking: children and adults as Web users. *Information Processing and Management*, *38*(5), 649–670.

Boldi, P., Bonchi, F., & Castillo, C. (2009). From Dango to Japanese Cakes: Query Reformulation Models and Patterns. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01* (pp. 183–190).

Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., & Vigna, S. (2008). The query-flow graph: model and applications. In *Proceedings of the 17th ACM conference on Information and knowledge management* (pp. 609–618).

Boldi, P., Bonchi, F., Castillo, C., Donato, D., & Vigna, S. (2009). Query suggestions using query-flow graphs. In *Proceedings of the 2009 workshop on Web Search Click Data* (pp. 56–63).

Bonchi, F., Perego, R., Silvestri, F., Vahabi, H., & Venturini, R. (2012). Efficient query recommendations in the long tail via center-piece subgraphs. In *Proceed-*

ings of the 35th international ACM SIGIR conference on Research and development in information retrieval (pp. 345–354).

Cao, H., Jiang, D., Pei, J., Chen, E., & Li, H. (2009). Towards context-aware search by learning a very large variable length hidden markov model from search logs. In *Proceedings of the 18th international conference on World Wide Web* (pp. 191–200).

Cao, H., Jiang, D., Pei, J., He, Q., & Liao, Z. (2008). Context-aware query suggestion by mining click-through and session data. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 875–883).

Capra, R. (2011). HCI Browser: A tool for administration and data collection for studies of web search behaviors. In A. Marcus (Ed.), *Design, user experience, and usability* (Vol. 6770, p. 259-268). Springer Berlin Heidelberg.

Ceaparu, I., Lazar, J., Bessiere, K., Robinson, J., & Shneiderman, B. (2004). Determining Causes and Severity of End-User Frustration. *International Journal of Human-Computer Interaction*, *17*(3), 333–356.

Cooper, A. (2008). A survey of query log privacy-enhancing techniques from a policy perspective. *ACM Transactions on the Web*, *2*(4), 19.

Cormack, G., Smucker, M., & Clarke, C. (2011). Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, *14*(5), 441–465.

Downey, D., Dumais, S., & Horvitz, E. (2007). Models of searching and browsing: languages, studies, and applications. In *Proceedings of the 20th international joint conference on artifical intelligence* (pp. 2740–2747).

Druin, A., Foss, E., Hatley, L., Golub, E., Guha, M. L., Fails, J., et al. (2009). How children search the Internet with keyword interfaces. In *Proceedings of the 8th International Conference on Interaction Design and Children* (pp. 89–96).

Dwork, C. (2006). Differential privacy. *Automata, languages and programming*, 1–12.

Dwork, C. (2008). Differential privacy: A survey of results. *Theory and Applications of Models of Computation*, *4978*, 1-19.

Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., & Naor, M. (2006). Our data, ourselves: Privacy via distributed noise generation. *Advances in Cryptology-EUROCRYPT 2006*, 486–503.

Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography*, 265–284.

Feild, H., & Allan, J. (2012). Task-aware search assistant. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval* (p. 1015).

Feild, H., & Allan, J. (2013). Task-aware query recommendation. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (p. To appear).

Feild, H., Allan, J., & Glatt, J. (2011). CrowdLogging: Distributed, private, and anonymous search logging. In *Proceedings of the 34rd international ACM SIGIR conference on Research and development in information retrieval* (p.).

Feild, H., Allan, J., & Jones, R. (2010). Predicting searcher frustration. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (pp. 34–41).

Feild, H., Velipasaoglu, O. E., Dumoulin, B., Churchill, E. F., Jones, R., & Bardzell, J. (2010, December 23). *Systems and methods for providing search assistance technologies based on user self-efficacy and search frustration.* (US Patent App. 12/978,261)

Filali, K., Nair, A., & Leggetter, C. (2010). Transitive history-based query disambiguation for query reformulation. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (pp. 849–850).

Fleischer, P. (2007). *Taking steps to further improve our privacy practices.* (`http://googleblog.blogspot.com/2007/03/taking-steps-to-further-improve-our.html`)

Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological bulletin*, *76*(5), 378–382.

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems*, *23*(2), 147–168.

Götz, M., Machanavajjhala, A., Wang, G., Xiao, X., & Gehrke, J. (2011). Publishing search logs—a comparative study of privacy guarantees. *IEEE Transactions on Knowledge and Data Engineering*, *99*.

Hassan, A., Jones, R., & Klinkner, K. L. (2010). Beyond DCG: User Behavior as a Predictor of a Successful Search. In *Proceedings of the third ACM international conference on Web search and data mining* (pp. 221–230).

He, Q., Jiang, D., Liao, Z., Hoi, S. C. H., Chang, K., Lim, E. P., et al. (2009). Web query recommendation via sequential query prediction. In *IEEE International Conference on Data Engineering* (pp. 1443–1454).

Ho, T. K. (1995). Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition* (Vol. 1, pp. 278–282).

Hong, Y., He, X., Vaidya, J., Adam, N., & Atluri, V. (2009). Effective anonymization of query logs. In *Proceeding of the 18th ACM conference on Information and knowledge management.*

Howe, D., & Nissenbaum, H. (2009). TrackMeNot: Resisting Surveillance in Web Search. In *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society.* Oxford Univ. Press.

Huang, C. K., Chien, L. F., & Oyang, Y. J. (2003). Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, *54*(7), 638–649.

Huffman, S., & Hochster, M. (2007). How well does result relevance predict session satisfaction? In *Proceedings of the 30th annual international acm sigir conference on research and development in information retrieval* (pp. 567–574).

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 133–142).

Jones, R., & Klinkner, K. L. (2008). Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. *Proceedings of CIKM 2008*.

Jones, R., Kumar, R., Pang, B., & Tomkins, A. (2007). I know what you did last summer: query logs and user privacy. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management* (pp. 909–914).

Jones, R., Kumar, R., Pang, B., & Tomkins, A. (2008). Vanity fair: privacy in querylog bundles. In *Proceeding of the 17th ACM conference on Information and knowledge management* (pp. 853–862).

Jones, R., Rey, B., Madani, O., & Greiner, W. (2006). Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web* (pp. 387–396).

Kanoulas, E., Carterette, B., Hall, M., Clough, P., & Sanderson, M. (2011). Overview of the trec 2011 session track. In *Proceedings of text retrieval conference.*

Kanoulas, E., Clough, P., Carterette, B., & Sanderson, M. (2010). Session track at trec 2010. In *Proceedings of the SIGIR Workshop on the Simulation of Interaction* (pp. 13–14).

Kifer, D., & Machanavajjhala, A. (2011). No free lunch in data privacy. In *Proceedings of the 2011 international conference on Management of data* (pp. 193–204).

Kodeswaran, P., & Viegas, E. (2009). Applying differential privacy to search queries in a policy based interactive framework. In *Proceeding of the ACM first international workshop on Privacy and anonymity for very large databases* (pp. 25–32).

Kong, W., & Allan, J. (2013). Extracting query facets from search results. In *Proceedings of the 36th annual ACM SIGIR conference on Research and development in information retrieval.*

Korolova, A., Kenthapadi, K., Mishra, N., & Ntoulas, A. (2009). Releasing search queries and clicks privately. In *Proceedings of the 18th international conference on World Wide Web* (pp. 171–180).

Krause, A., & Horvitz, E. (2010). A utility-theoretic approach to privacy in online services. *Journal of Artificial Intelligence Research*, *39*, 633–662.

Kuhlthau, C. C. (1991). Inside the search process: Information seeking from the user's perspective. *Journal of the American Society for Information Science and Technology*, *42*(5), 361–371.

Kumar, R., Novak, J., Pang, B., & Tomkins, A. (2007). On anonymizing query logs via token-based hashing. In *Proceedings of the 16th international conference on World Wide Web* (pp. 629–638).

Lawrie, D. J. (2003). *Language models for hierarchical summarization.* Unpublished doctoral dissertation.

Li, Y. (2009). Exploring the relationships between work task and search task in information search. *Journal of the American Society for information Science and Technology*, *60*(2), 275–291.

Liao, Z., Song, Y., He, L.-w., & Huang, Y. (2012). Evaluating the effectiveness of search task trails. In *Proceedings of the 21st international conference on World Wide Web* (pp. 489–498).

Lucchese, C., Orlando, S., Perego, R., Silvestri, F., & Tolomei, G. (2011). Identifying task-based sessions in search engine query logs. In *Proceedings of the fourth ACM international conference on Web search and data mining* (pp. 277–286).

Machanavajjhala, A., Kifer, D., Abowd, J., Gehrke, J., & Vilhuber, L. (2008). Privacy: Theory meets practice on the map. *Data Engineering, International Conference on*, *0*, 277-286.

Mei, Q., Zhou, D., & Church, K. (2008). Query suggestion using hitting time. In *Proceeding of the 17th ACM conference on Information and knowledge management* (pp. 469–478).

Metzler, D., & Croft, W. (2004). Combining the language model and inference network approaches to retrieval. *Information Processing & Management*, *40*(5), 735–750.

Microsoft. (2006). *Winners Announced for Microsoft Live Labs Search RFP.* (`http://www.microsoft.com/presspass/press/2006/jun06/`

06-01MSRLabSearchPR.mspx)

Radlinski, F., & Joachims, T. (2005). Query chains: learning to rank from implicit feedback. In *Proceedings of the eleventh acm sigkdd international conference on knowledge discovery in data mining* (pp. 239–248).

Schwartz, A., & Cooper, A. (2007). Search privacy practices: A work in progress. *Center for Democracy and Technology report (August 2007).*

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, *22*(11), 612–613.

Sweeney, L. (2002). *k*-Anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty Fuzziness and Knowledgebased Systems*, *10*(5), 557–570.

Szpektor, I., Gionis, A., & Maarek, Y. (2011). Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th international conference on World Wide Web* (pp. 47–56). New York, NY, USA: ACM.

White, R. W., & Dumais, S. T. (2009). Characterizing and predicting search engine switching behavior. In *Cikm '09: Proceeding of the 18th acm conference on information and knowledge management* (pp. 87–96). New York, NY, USA: ACM.

White, R. W., Jose, J. M., & Ruthven, I. (2006). An implicit feedback approach for interactive information retrieval. *Information Processing & Management*, *42*(1), 166-190.

Xie, I., & Cool, C. (2009). Understanding help seeking within the context of searching digital libraries. *Journal of the American Society for Information Science and Technology*, *60*(3), 477–494.

Xiong, L., & Agichtein, E. (2007). Towards privacy-preserving query log publishing. In *Query Log Analysis: Social And Technological Challenges Workshop in World Wide Web.*

Xu, Y., Wang, K., Zhang, B., & Chen, Z. (2007). Privacy-enhancing personalized web search. In *Proceedings of the 16th international conference on World Wide Web* (pp. 591–600).

Zhang, Z., & Nasraoui, O. (2006). Mining search engine query logs for query recommendation. In *Proceedings of the 15th international conference on World Wide Web* (pp. 1039–1040).

Zhou, B., & Xu, J. (2013). Privacy protection in personalized web search: A peer group-based approach. In A. Greenberg, W. Kennedy, & N. Bos (Eds.), *Social Computing, Behavioral-Cultural Modeling and Prediction* (Vol. 7812, p. 424-432). Springer Berlin Heidelberg.

Zhu, Y., Xiong, L., & Verdery, C. (2010). Anonymizing user profiles for personalized web search. In *Proceedings of the 19th international conference on World Wide Web* (pp. 1225–1226). New York, NY, USA: ACM.