A STUDY OF LANGUAGE MODELS FOR EXPLOITING USER FEEDBACK IN INFORMATION RETRIEVAL

BY

BIN TAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Doctoral Committee:

Associate Professor ChengXiang Zhai, Chair
Professor Jiawei Han
Associate Professor Kevin Chen-Chuan Chang
Professor Marianne Winslett

Abstract

Feedback is an important technique in Information Retrieval to have users provide contextual information about their search needs, with the goal of improving retrieval accuracy and achieving personalization. Relevance feedback has been studied extensively, and in recent years new types of feedback such as implicit feedback and collective feedback have attracted much research interest. However, there are not many works exploring language modeling techniques for feedback. In this thesis, I study how to use language models to exploit user feedback, including long-term implicit feedback and short-term explicit term-based feedback. I show that language models have unique advantages in modeling users' search interests and preferences in the long-term, as well as capturing term and sub-topic relevance in the short-term.

In particular, I first study exploiting implicit feedback from a user's long-term search log. Language models are constructed to represent both information needs in past searches and history context for new searches. These history language models capture the user's search interests and preferences, thus can help personalize search results for new queries. Not only that, by selecting topics in a user's long-term search history that represent their long-lasting exploratory interests and building language models for these topics, the user can receive personalized recommendation of new information without a query.

I also study term-based explicit feedback that deviates from traditional document-based relevant feedback. By modeling query sub-topics using language models and asking the user for term-level relevance judgments, both term and sub-topic level relevance can be incorporated into a new query model that improves retrieval accuracy.

Finally, I have designed the UCAIR system to support development, deployment and evaluation of feedback algorithms for personalized search and recommendation. The system

not only implements some of the previously proposed algorithms but also provides a highly reusable and easily extensible platform for designing and testing new feedback algorithms in the language modeling framework. It is hoped that this system will help reduce the difficulty of building personalized feedback systems and generating feedback evaluation data sets.

To Father and Mother.

Acknowledgments

I would like to express my deepest gratitude to my adviser, Professor ChengXiang Zhai, without whom this thesis would not have been possible. Cheng introduced to me the wonderful world of Information Retrieval, inspired me with his dedication to research and high standard on it, and led me onto the correct track of scientific study. Every time I came back from a discussion with Cheng, I would benefit not only from his insight on the research topics, but also his teachings on the right way of doing research, from how to identify research problems with impact, to how to approach them using scientific methodology, and to how to make effective writings and presentations, all with great clarity and patience. Cheng also treated me like an equal peer, encouraging me to speak out with my own ideas and freely explore directions I am interested in, for which I am really grateful.

I would also like to thank my other thesis committee members, Professor Jiawei Han, Professor Marianne Winslett, and Professor Kevin Chang for their constructive suggestions and feedback that helped me make this thesis more complete and accurate.

I want to say thanks to my friends from the DAIS group, who accompanied me through my years in graduate school and collaborated me on many projects, especially: Tao Tao, Xuehua Shen, Hui Fang, Jing Jiang, Azadeh Shakery, Hong Cheng, Zheng Shao, Deng Cai, Xu Ling, Xin He, Qiaozhu Mei, Xuanhui Wang, Tao Cheng, Shui-Lung Chuang, Alexander Kotov, Maryam Karimzadehgan, Vinod Vydiswaran, Younhee Ko, Hyun Duk Kim, Yue Lu, Yuanhua Lv, and Duo Zhang.

Finally, I would like to thank my parents, for their unconditioned love, for their patience and encouragement that support me through my graduate study. This thesis is dedicated to them.

Table of Contents

Chapter 1 Introduction **1**

Chapter 2 Literature Review **6**

Chapter 3 Long-term Implicit Feedback for Personalized Search **12**
3.1 Search Log Concepts and Notations 13
3.2 Context-Sensitive Information Retrieval 14
3.3 History Language Models 16
3.4 Data Collection 22
3.5 Experiment Results 24

Chapter 4 Long-term Implicit Feedback for User Modeling and Recommendation **28**
4.1 Discovering Long-lasting Exploratory Interest Patterns 30
4.2 Recommending New Information Using Search Patterns 37
4.3 Experiment Results 38

Chapter 5 Explicit Term Feedback **48**
5.1 Motivation 48
5.2 General Approach 51
5.3 Presentation Term Selection 52
5.4 Estimating Query Models from Term Feedback 55
5.5 Experiment Results 57

Chapter 6 Towards a Personalized Retrieval System **68**
6.1 General Architecture of Personalized Retrieval Systems 70
6.2 UCAIR Design Goals and Features 73
6.3 UCAIR Data Model 77
6.4 UCAIR Architecture and Components 80
6.5 Evaluation in UCAIR 85

Chapter 7 Summary **87**

References **89**

Chapter 1

Introduction

Recent years have seen widespread use of information retrieval systems represented by web search engines. Their popularity is partially due to the simple keyword search user interface: the user formulates a query composed of one or more keywords to summarize their information need, in response the retrieval system returns a ranked list of documents that best match these keyword terms. As queries are usually short (average length is reported to be 2.2 terms [38]), it requires little user effort. However, if the query fails to provide enough information to clarify the user's need, the retrieval system will have a hard time guessing what the user really wants. For example, for the ambiguous query "jaguar", the result list is typically a mix of documents matching different meanings of the query (big cat, car, or Mac OS), so the user has to manually sift through them or come up with a more refined query, neither of which is a trivial task.

Feedback has been a standard technique in information retrieval to reduce the information gap between user queries and their true intentions by acquiring additional contextual information from the user. With traditional *relevance feedback* [70, 73, 34], if the initial round of retrieval yields unsatisfactory results due to a non-optimal query, the user may clarify their intention by giving relevance judgment (either positive or negative) on some retrieved documents. The retrieval system modifies the original query by including/excluding/reweighting representative terms extracted from the feedback documents, and uses the refined query to execute a second round of retrieval. Feedback generally improves retrieval accuracy because of the improved query, which is true even in the case of *pseudo* or *blind feedback* [17, 13], where the top-ranked documents are assumed to be relevant.

Despite its effectiveness, relevance feedback is not widely adopted in production retrieval

systems. This is mainly due to its complexity: users have to spend extra effort to provide relevance judgment, and using the feedback interface can take time to learn. Therefore researchers have increasingly shifted their attention to *implicit feedback*: by passively observing users' contextual activities during their interaction with the retrieval system, we can often uncover clues about their search intentions and piece together more complete search contexts. For example, if a user spends a lot of time reading a document and make notes on it, it is highly probable that the document is relevant to the user's information need. Here the feedback information obtained from users for inferring their information needs is acquired by means of implicit observation, rather than explicit solicitation, thus there is no burden on users.

The dominant source of implicit feedback being studied and used today is probably users' query and click history recorded in their web search logs. As people become more and more reliant on web search engines for information seeking tasks in their lives (the average user conducted more than 80 searches in August 2007 [1]), search engine companies have accumulated massive amount of search log data, which is invaluable for mining search patterns that are good for implicit feedback. On one hand, by studying the collective search behavior of many other users on a query, we can aggregate their search preferences for collaborative filtering to help a new user on the same query. For example, if the majority of users who issue the jaguar query tend to click on the car manufacturer's website rather than a car owner's blog, this preference is likely also true for a new user who issue this query. This is *collective feedback* and has been studied in works such as [40, 4]. On the other hand, by examining a single user's search behavior on past queries, we can model their search interests and preferences to help the same user on a new query. For example, if a user made a lot of searches on Apple products in the past, then for a new query of jaguar the user is likely to prefer the Mac OS meaning of jaguar. This is *personalized feedback* and has been explored in works such as [56, 87]. Unlike collective feedback, here the search context relies on feedback from a user's personal search history, thus different users may get different results personalized to their unique needs.

Besides the disadvantage compared to implicit feedback, relevance feedback has another

drawback: feedback is usually on the document or passage level, and a user seldom has control of which terms are used for feedback: the use of large text units such as documents often introduces irrelevant terms, and the user has little opportunity to specify relevant terms. To overcome this, we can consider feedback on the term level: present individual terms to the user for feedback and use the term relevance judgments to improve queries accordingly. Since the user has more direct control of the feedback terms, the refined query is hopefully closer to the user's information need.

These various new types of feedback have sparked strong research interests in recent years, but surprisingly few works are based on statistical language modeling, which is a probabilistic framework for quantifying uncertainties in text [65, 52] and can bring superior retrieval performance with its modeling power. Although it has been successfully applied to relevance and pseudo feedback [53, 102, 85], there are few existing works studying how it can be extended to non-traditional areas of feedback. This deficiency is addressed in this thesis, as we attempt to apply language modeling techniques on new types of feedback, including long-term implicit feedback from personal search history and short-term explicit term-based feedback. We find language modeling makes it easy to represent various text constructs such as queries, documents, user search context and user interest topics in a unified way and manipulate them with ease. It allows us to model users' search interests and preferences in the long-term, as well as to capture term and sub-topic relevance in the short-term. We have developed algorithms showing that implicit feedback from a user's long-term personal search log can be successfully exploited to provide personalization in search and recommendation, and term-based feedback is able to achieve favorable retrieval performance compared to relevance feedback.

More specifically, the contributions of this thesis are:

1. We study how to exploit implicit feedback from a user's long-term search log for search personalization using language models. Given a new query, a history language model is estimated from the user's search log to capture the query's historical context. The history language model, which serves as implicit feedback, is combined with the original query to rerank the search results to match the user's search preferences.

Evaluation on search logs from real users shows implicit feedback from personal search history substantially improves retrieval accuracy for both recurring and fresh queries, and works best when the history language model is estimated using clickthrough data and a discriminative weighting scheme.

2. We show implicit feedback from a user's long-term search log can also be exploited to model the user's search interests and perform personalized recommendation of new information. Using text clustering, related searches in the log are grouped into topics. Since a search interest topic is most interesting when it is both long-lasting and exploratory, frequency and exploratoriness measures are proposed to filter the topics. Language models are built to summarize the user's search interest topics and used to recommend new information matching the user's interests. The recommendation is solely based on implicit feedback, as the topics are constructed from the search log without any help from the user. Evaluation is performed using a simulated study on real web search logs.

3. We study term-based explicit feedback using language models. With term feedback the user directly judges the relevance of individual terms without interaction with feedback documents, thus taking full control of the query expansion process. The terms presented to the user for feedback are selected to cover sub-topics in the search results, and the refined query language model is constructed using direct term relevance judgment and indirect sub-topic relevance inference. Evaluation on the TREC HARD track shows term feedback can achieve favorable retrieval performance compared to relevance feedback.

4. The UCAIR (User-Centered Adaptive Information Retrieval) system has been designed to aid the development, deployment and evaluation of personalized information retrieval systems. In the past, the development of feedback algorithms is hindered by the lack of real usage data such as personal search logs, while the difficulty of deploying these algorithms in production systems to attract enough users makes it hard to collect real usage data. It is UCAIR's goal to break this vicious circle. Designed for

client-side personalization, it includes various data entities and components that are highly reusable and extensible to support personalized search and recommendation with language models. UCAIR easily implements the previously proposed feedback algorithms and provides a platform for easy development of new algorithms and collecting of feedback data.

The rest of this thesis is organized as follows. Chapter 2 provides an overview and categorization of existing feedback works in information retrieval literature. Chapter 3 studies exploiting implicit feedback from long-term search log for personalized search using language models. Chapter 4 is focused on modeling user interest topics in long-term search log for recommendation of new information. Term-based feedback is studied in Chapter 5. The UCAIR system for developing feedback algorithms is presented in Chapter 6. Chapter 7 concludes the thesis.

Chapter 2

Literature Review

This chapter aims to provide an introduction to some recent works on feedback. As there are a wide range of feedback techniques, the material is organized according to feedback types, going through explicit feedback, implicit feedback, collective feedback, personalized feedback, short-term feedback, long-term feedback, adaptive feedback, non-document-based feedback and hybrid feedback. Within this big picture of feedback the positioning of this work is discussed.

In a typical information retrieval setting, the user describes their information need with a query Q, in response the retrieval system returns a ranked list of documents $D_1, D_2, D_3, \cdots$ as results, as shown in Figure 2.1. If the initial ranking is poor, one way for improvement is to ask the user to provide feedback, i.e., to evaluate the relevance of some top-ranked documents. According to the cluster hypothesis [89], which states that relevant documents tend to be more similar to each other than to irrelevant ones, if some example relevant documents are identified from user feedback, one may find more relevant documents by seeking similar ones. The traditional relevance feedback has been studied in works such as [70, 73, 34, 102, 53]. As can be seen in Figure 2.2, a feedback loop is introduced: relevance judgment on the initial results are fed back to the system to perform a second-round retrieval, presumably generating a better ranking. Typically this is done by extracting informative terms from the feedback documents and adding them to the original query, producing a refined query Q' that better represents the user's information need. Thus relevance feedback is usually tied with *query expansion*. There are other ways to use feedback, though. For example, in [90], documents with a short web graph distance to the feedback documents receive boost to their relevance scores, without a query modification stage. Finally, some feedback approaches do not involve user judgment, but seek search context

Figure 2.1: No feedback

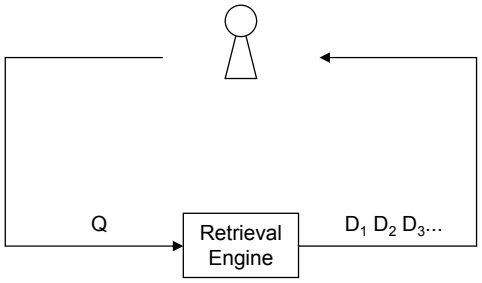


Figure 2.2: Relevance feedback

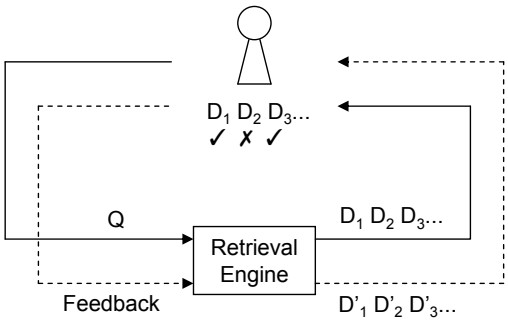


Figure 2.3: Collective feedback

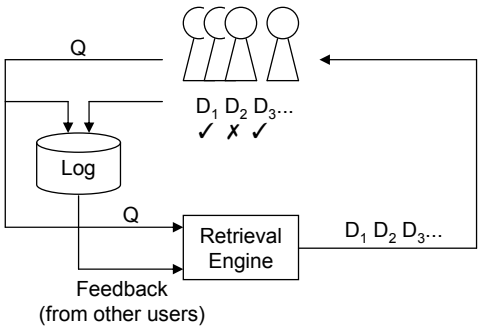
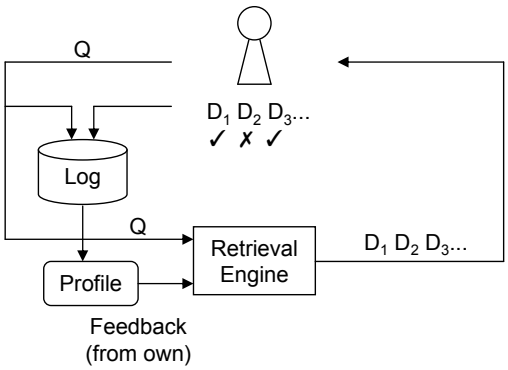


Figure 2.4: Personalized history-based feedback



from external knowledge like word co-occurrence (e.g., [11]). Particularly, if one assumes all the top documents to be relevant and extracts terms from them for expansion, it becomes *pseudo or blind feedback* [17, 13, 18].

In relevance feedback, the user has to *explicitly* give relevance ratings on documents, and the toll of extra user efforts has largely prevented its wide adoption in production retrieval systems such as search engines. In contrast, *implicit feedback* infers relevance judgment silently from observed user actions when they are interacting with a retrieval system, without explicitly asking for input. For example, one might assume a user is interested in a search result if the user clicks on it and spends long time viewing it. [61] classifies different types of implicit feedback in recommender systems into the categories *examination* (e.g., clickthrough, duration), *retention* (e.g., bookmarking) and *reference* (e.g., copy&paste). Duration (time spent on a document) [25, 47, 94] and clickthrough (user

clicking on a link to view the content) [31, 41, 28] are the two most studied implicit feedback measures and are found to correlate well with explicit ratings. Combining multiple types of implicit feedback further improves the accuracy of predicting explicit relevance judgment [31, 59, 5]. Joachims et al. [41] proposes to interpret clickthrough as relative relevance preference instead of absolute judgment, as clickthrough is biased by initial ranking. An adaption of this is in [28], where relative preference is extracted for URL pairs that have large difference in aggregated click count.

An early bibliography on implicit feedback is given in [50]. Implicit feedback becomes more popular with the availability of large amount of search log in the search engine companies, which provides an abundant yet cheap source of implicit feedback (especially in the form of clickthrough). The search log is often modeled as a bipartite graph between queries and documents, where clicks establish edges connecting them [14, 32, 64, 58]. Search log has been exploited for many tasks, such as query suggestion [45, 93, 58], query classification [55], generating ad keywords [32], document representation [82, 64] and clustering search results [14, 91]. For the very task of document retrieval, Figure 2.3 shows how *collective feedback* from a large number of users can help improve ranking. Compared to relevance feedback (Figure 2.2), there are two main differences. First, there is only one retrieval round: collected user feedback (usually in the form of clickthrough) is not consumed immediately to help the current search, but retained to help other users in the future. Second, there is a log database that stores queries and feedback from all users. It can be used to tweak the retrieval function in an offline phase (e.g., fitting the function parameters to log data), or to adjust the ranking for an incoming query in an online fashion (e.g., boosting results clicked most by other users). In this setting, user feedback from a search instance is not very useful per se, but when millions of users have their feedback pooled together, the aggregated statistics represent the collective intelligence of the user population. Representative works include [40, 67, 19], which train a ranking SVM [40] on preference pairs extracted from search log, and [4, 6, 15], which use implicit feedback features as direct or indirect input to RankNet [69]. Both approaches are able to learn ranking functions that incorporate implicit feedback from search log and yield improved retrieval accuracy.

In collective feedback (Figure 2.3), when a user issues a query, the log database is usually looked up for feedback from other users on the same query, rather than from the same user on other queries in his/her history. The collective feedback approach is most useful for queries that are popular and unambiguous, but less effective for long-tail queries (i.e., infrequent ones) or queries whose intention differ by user (e.g., "jaguar"). It is able to improve overall retrieval accuracy, but does not personalize the search results to fit particular users. In contrast, Figure 2.4 shows single-user feedback from past search history. Its subtle difference from Figure 2.3 is that only one user is involved, thus the log database only stores feedback from that user only. This has several implications: compared to a gigantic search log containing millions of users' records, a single-user one is small enough to reside on the user's client side, which alleviates privacy concerns, and more importantly, allows for more computation-intensive feedback algorithms. In particular, we can afford not only feedback from the same session (*short-term feedback*) [76, 75, 97], but also feedback from long ago (*long-term feedback*) as long as it helps the current query. For long-term feedback, most personalized history-based feedback algorithms build some kind of user profile, which could be, for example, word usage (term frequency or representative phrases and sentences) from a local desktop index [87, 22], weighted term vectors covering different lengths of history [81], a weighted vector or hierarchy of predefined or summarized categories [56, 23, 66, 27, 79, 99], or a set of clusters [21, 57]. Some profiles involve not only search history, but also indexed personal documents like emails and web pages [81, 87, 22] as they are readily available on the client side. The different user profiles, which are all derived from implicit feedback in the user's local context, aim to capture essential aspects of the user's search preferences and/or interests that distinguish the user from others. The personalization algorithms would then adjust the document ranking to reflect the user profile as search context. For example, [87, 21] use a modified BM25 [43] formula to give special weights to frequent local terms, [22] adds feedback terms to the original query, [81, 57, 23, 27, 79] score the result documents against the profile based on their distance, and [39, 66] compute topic-sensitive PageRank [35] using the learned topic distribution.

Web search queries often have different goals and may require different handling. The

queries can be categorized into navigational, informational, transactional or resource-seeking according to [16] and [71]. [20] distinguishes between finding and refinding tasks and [86] discusses how to detect refinding queries. For personalized implicit feedback algorithms, it is often wise to make adaption according to whether feedback is necessary and/or whether enough feedback is available. For queries whose intentions are very clear and vary little by user (e.g., "wikipedia"), there is little room for disambiguation using the user's own history. Also, for some new queries the user may not have accumulated enough feedback, especially at the beginning of use of a system (known as "cold start"). Click entropy is explored in [27] to select appropriate personalization strategies. In [22], the number of expansion terms is controlled by query clarity [26]. [88] builds predictive models using many query, document and history features to identify queries that can benefit from personalization.

My work on long-term implicit feedback for search personalization [83] distinguishes itself from previous works in several aspects. First, following our previous works on short-term implicit feedback [76, 75], it is based on the statistical language modeling framework, which allows a uniformed and easy way to represent and manipulate queries, documents, search history and user information needs. Second, the memory-based approach records in a user profile all past searches and their associated language models, thus it retains more detailed feedback information than other approaches that use a few term vectors (e.g., in [81, 87]) or categories (e.g., in [23, 66]) to represent user preferences, which inevitably cause information loss. Third, the use of past history to personalize search is adaptive, in the sense that the amount of implicit feedback incorporated depends on how much related information can be found in the history. The performance on two types of queries (fresh and recurring) is discussed separately.

As found in an analysis on Yahoo! search log [92], a large number of users own a long search history, and a user's topical interests as reflected by the search log tend to be very consistent, but differ with other users. This offers the opportunity to mine the user's personal search log to model their interests. This work studies how to extract long-lasting yet exploratory patterns from the search log using implicit feedback. The search patterns are shown to represent different search interest topics, and can be used to recommend new

information to user. In some ways this is similar to adaptive information filtering with feedback [106, 108, 96], where a user provides ratings to a document stream to train a filter. The major difference is that search log is more structured than a document stream (e.g., there is the presence of queries and sessions) and more noisy (e.g., there can be many ad-hoc queries and clicks that have little bearing on user interests).

The above feedback methods are mostly document-based. But when document is used as the unit for feedback, many irrelevant terms can be introduced along with the relevant ones. To overcome this, people have proposed passage feedback [7, 98], which is shown to improve performance with its smaller feedback unit. A more direct solution is to ask the user for their relevance judgment of feedback terms. For example, in some relevance feedback systems such as [51], there is an interaction step that allows the user to add or remove expansion terms after they are automatically extracted from relevant documents. This is categorized as *interactive query expansion*, where the original query is augmented with user-provided terms, which can come from direct user input (free-form text or keywords) [80, 37, 48] or user selection of system-suggested terms (using thesauri [33, 80] or extracted from feedback documents [33, 80, 51, 10, 37]). My work [84] differs from previous ones in that it selects terms carefully to cover multiple clusters in the feedback documents, and use language models to construct a refined query model that captures both term and cluster feedback.

Finally, there is potential to combine various types of feedback into a hybrid one and make the feedback algorithm adaptive to different situations. For example, [81, 27, 96] combine short-term profile and long-term profile, which generally makes feedback more reliable than using only one of them. In another direction, [81, 27, 63] incorporate both personal feedback from single user and collective feedback from a group of users, thus taking advantage of both personalization and collaborative filtering.

Chapter 3

Long-term Implicit Feedback for Personalized Search

In this chapter, we study how to exploit implicit feedback from a user's search log for personalized search (the work was first published in [83]). Unlike our previous works on short-term implicit feedback [76, 75], here the focus is on long-term implicit feedback. In contrast to short-term search history, which only involves a single search session and has a coherent information need, long-term search history is unlimited in time scope and may include all search activities in the past. As such, it provides a more complete record on the user's search preferences and interests. For example, one might be able to discover long-term preference for results from the .edu domain or long-term interest on computer science topics, which are "macro" patterns unavailable if we only examine short-term feedback in isolated sessions. However, the inclusive nature also makes long-term search history more noisy: there could be many queries and clicks irrelevant to the user's true interests; blindly applying long-term feedback for personalization could yield undesirable results. For example, searches such as "iraq war" that are irrelevant to the current search "jaguar" would not be helpful, and such noise can even overshadow the signal of truly relevant past searches. Unfortunately, many works on long-term implicit feedback (e.g., [81, 87]) are not query-adaptive: they apply personalization without distinguishing between queries with strong or weak history support, and do not handle relevant and irrelevant portions of search history differently.

Here we systematically study how to exploit a user's implicit feedback from their long-term search history to improve retrieval accuracy. Our method follows the language modeling approach, by casting the feedback problem into a problem of how to estimate an improved contextual query model from user feedback. We build hierarchical history models to capture user's search context through implicit feedback. We propose a principled mixture

model algorithm to estimate weights in the history model using search history as evidence, which allows feedback to be query-adaptive. We evaluate the algorithm on a test set of Web search logs collected from some real users, and find implicit feedback from long-term search history can substantially improve retrieval performance for both recurring and fresh queries. It works best when clickthrough data is used with a discriminative weighting scheme for past searches. We also find that although recent history tends to be more useful than remote history (especially for fresh queries), all of the entire history is helpful for improving the retrieval accuracy of recurring queries.

The chapter is organized as follows. In Section 3.1, we describe search log data with some concepts and notations. In Section 3.2, we introduce a context-sensitive information retrieval paradigm that allows us to incorporate personal search history as search context, and cast the task of exploiting long-term implicit feedback for personalized search as a query model estimation problem. In Section 3.3, we develop several algorithms based on statistical language models to exploit long-term implicit feedback. We describe how a test set is built by collecting users' web search history in Section 3.4 and present the experiment results in Section 3.5.

3.1 Search Log Concepts and Notations

In a typical scenario of information retrieval, after a query is submitted, the retrieval system will return a set of result documents with titles and summaries displayed. The user can then select to view the full texts of some results (usually by clicking on them). Thus the search history generally includes three components: past queries, their search results, and the information on which results were clicked/viewed (also known as clickthrough data).

Formally, let q_i be a query, D_i be the set of its result documents, $C_i (C_i \subseteq D_i)$ be the set of clicked ones, and t_i be q_i's submission time. A search instance is represented by a tuple $S_i = \langle q_i, t_i, D_i, C_i \rangle$ (if needed, we can include more details like document view duration). If q_k is the current query, its search history H_k consists of all previous searches $S_1, S_2, \cdots, S_{k-1}$ ordered by time.

In our work we represent a document d using a concatenation of its title and summary

instead of its full content. These "captions" are readily available from a search engine's result page and have been shown in [24] to have a strong influence on user click behavior. Even if the actual content is irrelevant, the captions that attracts the user into a clickthrough are still likely to contain relevant terms. Also, we only use absolute clickthough as implicit feedback. Although one can combine multiple implicit feedback measures [59] or use relative preference [41], absolute clickthrough is easier to use and already has good performance in our experiments.

Sometimes a user's information need (especially if it is complex) cannot be satisfied just after one query; multiple rounds of query formulation and document retrieval need to be performed. Thus queries are usually grouped into sessions, where a session is one or more of queries issued with a short time span and sharing a coherent information need. Traditionally people use a fixed time threshold to decide session boundaries. For example, if two adjacent queries have a gap of ore than 30 minutes, they are often considered to belong to different sessions. [67] uses more features (e.g., cosine distance) and learn an SVM classifier to identify query chains in sessions. [44] proposes to go beyond simple time-based segmentation and organize queries into a hierarchy of missions and goals. Sessions help group related queries together, but in this chapter we just use queries as atomic containers of implicit feedback.

3.2 Context-Sensitive Information Retrieval

Traditional retrieval models take the retrieval problem as matching a query with a set of documents, and thus are inadequate for modeling personalized search having feedback as additional search context. Following the approach taken in our previous works ([76, 75]), we implement context-sensitive information retrieval with statistical language modeling, and use it to incorporate feedback from search history. Below we briefly describe how this can be achieved.

A language model provides a probabilistic way to quantify the uncertainties in text. In information retrieval, the simplest unigram model is most popular, where texts are assumed to be generated by sampling a multinomial distribution on terms. In the Kullback-Leibler

(KL) divergence retrieval method [102], if we have computed a query language model θ_q for the query q and a document language model θ_d for each document d, we can score the documents according to their KL divergence $D(\theta_q||\theta_d)$ to the query, which is defined as

$$D(\theta_q||\theta_d) = \sum_{w \in V} p(w|\theta_q) \log \frac{p(w|\theta_q)}{p(w|\theta_d)} \tag{3.1}$$

where w is a word in the vocabulary set V.

Clearly, with this method, our main task is to estimate θ_d and θ_q. We estimate the document model θ_d using Bayesian smoothing with Dirichlet prior [103]:

$$p(w|\theta_d) = \frac{c(w;d) + \mu p(w|\theta_C)}{|d| + \mu} \tag{3.2}$$

where $c(w;d)$ is the count of w in d, $|d|$ is the document length, $p(w|\theta_C)$ is the collection language model and μ is the Dirichlet prior. In our study, we use the TREC web corpus WT2G[1] to estimate the collection language model $p(w|\theta_C)$ and set μ to 10, which is optimized for the contextless baseline and suitable for short snippet texts.

When no other evidence is available (i.e., being contextless), the query model θ_q can be estimated solely based on the query text, using the Maximum Likelihood Estimator (MLE):

$$p(w|\theta_q) = \frac{c(w;q)}{|q|} \tag{3.3}$$

where $c(w;q)$ is the count of w in q and $|q|$ is the query length. But when there is feedback from the search history, we can incorporate it as additional evidence to improve our estimate of the query language model. This natural incorporation of extra information is why language models are particularly suitable for modeling context-sensitive search.

Specifically, given search history H, we can estimate the context-sensitive query model $p(w|\theta_{q,H})$ as a mixture:

$$p(w|\theta_{q,H}) = \lambda p(w|\theta_q) + (1-\lambda) p(w|\theta_H) \tag{3.4}$$

[1] http://ir.dcs.gla.ac.uk/test_collections

where $p(w|\theta_q)$ is the context-independent language model estimated using query text only, $p(w|\theta_H)$ is a history language model learned from feedback in search history, and $\lambda \in [0, 1]$ is the interpolation weight. The key is to estimate the best history model θ_H from q's history H, i.e., the model that most accurately captures the user's implicit feedback from search history and can hopefully bring greatest increase in retrieval accuracy. There are several challenges in this task: First, a past search can contain different components (query, results and clickthrough), and we should find the best way to combine these pieces of contextual information. Second, as discussed, not all past queries are equally relevant, and we need to identify past queries related to the current query and weight them appropriately. Third, when the search history has hundreds or thousands of entries, efficiency may become a concern. These issues will be addressed in the next section.

3.3 History Language Models

In this section, we discuss how to compute the history language model θ_{H_k}, which is regarded as the search context of the current query q_k and to be interpolated with θ_{q_k}. The strategy is to first generate a unit history model for each history query, and then combine them to get the overall history model.

3.3.1 Unit History Model

For each past query $q_i \in H_k$, we will estimate a language model θ_i that captures the user's information need at that particular moment. We call this a unit history model, because it represents a basic unit of search history that can be integrated to produce the overall history model. We use the following formula to compute it:

$$p(w|\theta_i) = \lambda_q p(w|\theta_{q_i}) + (1 - \lambda_q)\frac{\sigma_C \sum_{d_j \in C_i} p(w|\theta_{d_j}) + \sigma_{NC} \sum_{d_j \in NC_i} p(w|\theta_{d_j})}{\sigma_C |C_i| + \sigma_{NC} |NC_i|} \tag{3.5}$$

where λ_q is the interpolation weight on the original query model, θ_{q_i} and θ_{d_j} are the query and document language models, and $NC_i = D_i - C_i$ is the set of non-clicked results. The fraction in the above formula is essentially a weighted average of result document models,

with σ_C and σ_{NC} being the weights on clicked and non-clicked results, and $|C|$ and $|NC|$ being the number of clicked and non-clicked results.

Below we discuss three special degenerated cases for setting the parameters, each involving a single component of search history (namely, query, documents and clickthrough):

- $\lambda_q = 1$: (3.5) simplifies to

$$p(w|\theta_i) = p(w|\theta_{q_i}) \tag{3.6}$$

  The unit history model is based on query text only.

- $\lambda_q = 0, \sigma_C = \sigma_{NC}$: (3.5) simplifies to

$$p(w|\theta_i) = \frac{\sum_{d_j \in D_i} p(w|\theta_{d_j})}{|D_i|} \tag{3.7}$$

  The unit history model makes use of result documents only by averaging document models. Because query texts are usually short, the user's information need can often be better inferred from these result documents. This resembles pseudo feedback, and we expect the formula to give higher performance than the previous one.

- $\lambda_q = 0, \sigma_C \neq 0, \sigma_{NC} = 0$: (3.5) simplifies to

$$p(w|\theta_i) = \frac{\sum_{d_j \in C_i} p(w|\theta_{d_j})}{|C_i|} \tag{3.8}$$

  The unit history model is generated from clicked result documents only. Because clicked documents reflect a user's implicit feedback, the constructed language model should be more accurate than the one in (3.7), where clickthrough information is not used. If there are no clicks ($C_i = \emptyset$), the query (and its unit history model) is ignored when this formula is used.

The general form of (3.5) combines different components of search history. Typically, we can set $\sigma_C > \sigma_{NC} > 0$, so that clicked results receive more weight than non-clicked ones. On the data set, we find that the setting $\lambda_q = 0, \sigma_C = 20, \sigma_{NC} = 1$ achieves good performance.

3.3.2 Overall History Model

We use a weighted average of the unit history models of past queries as the overall history model:

$$p(w|\theta_{H_k}) = \frac{\sum_{q_i \in H_k} \lambda_i p(w|\theta_i)}{\sum_{q_i \in H_k} \lambda_i} \tag{3.9}$$

We discuss two general weighting schemes below.

3.3.3 Equal Weighting

With equal weighting, unit history models of different past queries are assigned equal weights:

$$\lambda_i = 1, \forall q_i \in H_k \tag{3.10}$$

If the unit history models only rely on query texts (3.6), and queries are assumed to be of equal length, the probability of a term in the overall history model is proportional to its global frequency in all queries of the history. Similar things can be said about (3.7) and (3.8) for search results and those clicked ones.

This simple weighting scheme suffers from the problem that, as it tries to assign equal weights to every piece of search history, none of them obtains much weight to be influential. It produces a global but weak description of the user's long-term interests.

3.3.4 Discriminative Weighting

As we have discussed in the start of this chapter, out of all past searches, only those that are related to the current query are important as its context. We should therefore concentrate the weight mass on them, and ignore other random, noisy parts of the search history. We call this approach discriminative weighting, as we are selective about which parts of search history to use according to the current query.

Generally, the more similar to the current query a previous query is, the more weight it should have in the computation of the overall history model. Below we describe several methods for calculating similarity scores between two queries, which can be used as interpolation weights in (3.9).

Cosine Similarity

For each query q_i, we compute a TF-IDF vector v_i that corresponds to the concatenation of all its result documents:

$$v_i[w] = \sum_{d_j \in D_i} c(w; d_j) \log \frac{N+1}{DF(w)+0.5}, \tag{3.11}$$

where $v_i[w]$ is the element in the TF-IDF vector that corresponds to term w, N is the number of documents in the background corpus (WT2G), and $DF(w)$ is w's document frequency. We choose to use concatenation of result documents rather than query text because query text is usually very short, so there may not be enough overlapping between two queries, even if they are related.

The cosine similarity between two vectors is defined as

$$\cos(\mathrm{v_i}, \mathrm{v_j}) = \frac{v_i \cdot v_j}{|v_i||v_j|} \tag{3.12}$$

and is always in $[0, 1]$.

Since $\cos(v_i, v_k)$ measures how close q_i is related to q_k, we can naturally use it for λ_i in (3.9).

EM Estimation

Here we present a more principled approach, in which λ_i is derived from mixture weights in a generative model.

Suppose there is a mixture model θ_{mix}:

$$p(w|\theta_{\text{mix}}) = \mu_C p(w|\theta_C) + \mu_q p(w|\theta_{q_k}) + \sum_{i=1}^{k-1} \mu_i p(w|\phi_i), \tag{3.13}$$

where θ_C is the background language model estimated from the corpus (WT2G), θ_{q_k} is MLE from the current query q_k's text, and ϕ_i is MLE from the concatenation of result documents

of q_i, a past query in H_k:

$$p(w|\phi_i) = \frac{\sum_{d_j \in D_i} c(w; d_j)}{\sum_{d_j \in D_i} |d_j|} \tag{3.14}$$

μ_C, μ_q and μ_i are mixture weights and constrained by

$$\mu_C + \mu_q + \sum_{i=1}^{k-1} \mu_i = 1 \tag{3.15}$$

Let Λ denote the set of mixture weights (μ_C, μ_q, μ_i). We want to choose Λ^* to maximize the log likelihood for the mixture model to generate the result documents of q_k:

$$\begin{aligned} \Lambda^* &= \text{argmax}_\Lambda \log p(D_k|\Lambda) \\ &= \text{argmax}_\Lambda \sum_{d_j \in D_k} \sum_{w \in d_j} c(w; d_j) \log p(w|\theta_{\text{mix}}) \end{aligned} \tag{3.16}$$

From (3.13) and (3.16), it can be easily seen that, the closer q_i is related to q_k, the larger mixture weight (μ_i) ϕ_i will have in the mixture model (because it fits D_k better). Indeed, μ_i reaches its maximum when D_i is identical to D_k. Therefore, we can use μ_i for λ_i in (3.9).

To estimate these mixture weights, we use the EM algorithm. Let w_j be the j-th word in the concatenation of all result documents in D_k. The Q-function is

$$\sum_{j=1}^{L} \Big(p(Z_{Cj}|\Lambda^{(n)}) \log \mu_C p(w_j|\theta_C) \; + \; p(Z_{qj}|\Lambda^{(n)}) \log \mu_q p(w_j|\theta_{q_k}) \; + \sum_{i=1}^{k-1} p(Z_{ij}|\Lambda^{(n)}) \log \mu_i p(w_j|\phi_i) \Big)$$

where $L = \sum_{d_j \in D_i} |d_j|$ is the sum of q_k's result document lengths, $\Lambda^{(n)}$ is the set of parameters in the n-th iteration, and Z_{Cj}, Z_{qj}, Z_{ij} are the hidden variables, indicating the events of w_j being generated by θ_C, θ_{q_k}, ϕ_i respectively.

In the E-step, we have

$$p(Z_{Cj}|\Lambda^{(n)}) = \frac{\mu_C^{(n)} p(w_j|\theta_C)}{\mu_C^{(n)} p(w_j|\theta_C) + \mu_q^{(n)} p(w_j|\theta_{q_k}) + \sum_{i=1}^{k-1} \mu_i^{(n)} p(w_j|\phi_i)}$$

$$p(Z_{qj}|\Lambda^{(n)}) = \frac{\mu_q^{(n)} p(w_j|\theta_{q_k})}{\mu_C^{(n)} p(w_j|\theta_C) + \mu_q^{(n)} p(w_j|\theta_{q_k}) + \sum_{i=1}^{k-1} \mu_i^{(n)} p(w_j|\phi_i)}$$

$$p(Z_{ij}|\Lambda^{(n)}) = \frac{\mu_i^{(n)} p(w_j|\phi_i)}{\mu_C^{(n)} p(w_j|\theta_C) + \mu_q^{(n)} p(w_j|\theta_{q_k}) + \sum_{l=1}^{k-1} \mu_l^{(n)} p(w_j|\phi_l)}$$

In the M-step, we have

$$\mu_C^{(n+1)} = \frac{\sum_{j=1}^{L} p(Z_{Cj}|\Lambda^{(n)})}{L}$$

$$\mu_q^{(n+1)} = \frac{\sum_{j=1}^{L} p(Z_{qj}|\Lambda^{(n)})}{L}$$

$$\mu_i^{(n+1)} = \frac{\sum_{j=1}^{L} p(Z_{ij}|\Lambda^{(n)})}{L}$$

Because we have computed μ_q, the mixture weight on q_k, we may estimate λ_{q_k} in (3.4) based on it, instead of using a fixed value:

$$\lambda_{q_k} = \frac{\mu_q}{\mu_q + \sum_{i=1}^{k-1} \mu_i} \tag{3.17}$$

This way, the weighting in the final contextual model θ_k is very flexible: when there is a rich amount of relevant search history (reflected by a large value of $\sum_{i=1}^{k-1} \mu_i$ compared to μ_q), there will be significant weight on the history model θ_{H_k}; on the other hand, when the search history is mostly irrelevant, the MLE model θ_{q_k} from query text will dominate. Moreover, all the weighting parameters (i.e., λ_{q_k} and λ_i's) will be estimated rather than manually set.

Hybrid Method

The EM estimation method, although shown to produce more accurate weights, runs much slower than the cosine similarity method, due to the fact that the EM algorithm usually needs many iterations to converge, and each iteration is generally more complex than just computing a cosine similarity value. This will be a big concern for longer search history, when there are hundreds or thousands of queries.

We observe on the data set that, with discriminative weighting, only a small number of previous queries are most related to the current query and receive non-insignificant weights (which is exactly what we intend to see). Motivated by this, we first run the cosine similarity method, identify the queries with highest similarity scores, and keep them in a *working set*.

We then run the EM estimation method only on the queries in this working set, and assign zero weights to other queries in the search history. We find this approach yields similar retrieval accuracy as the original EM method, yet runs almost as fast as the cosine method.

3.4 Data Collection

At the time of experiment, there was no publicly available collections of search logs that contain reasonably long period of users' search history with implicit feedback information. The later AOL search log [62], though containing search history for a large number of users, provides little chance for conducting explicit user evaluation. Therefore, we chose to create our own data set in the web search domain by making a plug-in for the Firefox browser to record a user's long-term search history. Specifically, the plug-in saves to a log file all user search activities that are captured from the browser, including queries issued to the Google search engine, search results (with titles, summaries and URLs) returned, and the information of which results are clicked on. The plug-in collects search history in the background and is intentionally kept transparent from the user so that it will not interfere with her normal search activities.

Four computer science students kept the plug-ins installed on their personal computers for over a month and then submitted their individual search logs to us (they were free to delete any sensitive queries that they do not want to disclose). Next the users were asked to pick at least 15 queries from their own search logs, starting from the back (the most recent history). The queries selected from the search logs would be evaluated to create a test data set. They must satisfy the following conditions, so that there is room for potential improvement of retrieval accuracy with long-term context.

1. A selected query should have at least one relevant document. Thus misspelled queries and queries issued just to check for the existence of something are excluded.

2. A selected query should either match the person's interests and background (e.g., computer science, pop music, football) or belong to a search session (a chain of queries for the same information need), being a reformulation of some previous query.

For each query, we chose the top 20 results retrieved from Google as the collection of documents (with Google's ranking information removed) to be scored by our retrieval methods. We only use their snippet texts (title + summary). To evaluate these result documents, the users were presented with the set of top 20 results retrieved from Google and asked to judge whether each document was relevant or not. If a query was a known-item search, i.e., if the user knew exactly what the needed result would look like, then only that result should be deemed relevant. Otherwise, if the user was exploring some topic, then they should mark all results matching that topic as relevant. For example, if the user has visited Python language's website before and is searching for it again, only this result should be considered relevant; if the user does not know Python and searches to get some ideas about it, then a tutorial on Python is also relevant.

We distinguish two types of queries due to their different nature and retrieval performance. If a query has occurred before in the search history (in exact form or with keywords' order changed) and there are clicks associated with its earlier occurrence(s), it belongs to the category of *recurring* queries (known as repeat queries in [86]). Otherwise, we call the query *fresh*. Usually, the purposes of recurring queries is to refind information [20], and are more likely to be navigational than informational or transactional [16]. Recurring queries are also more likely to reflect the user's long-term interests. It tends to be easier to improve the retrieval performance of recurring queries, as the user is very likely to choose exactly those results clicked on in an earlier search.

Table 3.1 shows some statistics of the collected log data. The large difference in the number of queries is due to some users not using Firefox for all of their web searches.

Table 3.1: Statistics of search log data

	user1	user2	user3	user4
# days in search history	65	44	69	64
# queries	1255	355	376	136
# queries with ≥ 1 clicks	607	238	207	79
avg. # clicks for query with ≥ 1 clicks	1.26	1.48	1.56	1.37
# testing queries	71	63	19	17
# fresh/recurring queries	54/17	59/4	12/7	13/4
avg. # rel. results per query	2.09	4.14	3.58	6.59

3.5 Experiment Results

In this section, we empirically evaluate the performance of the proposed methods on our data set of personal web search logs. We will also study the influence on retrieval accuracy of individual components and different time cutoffs in search history.

We use the standard TREC mean average precision (MAP) and precision at top 5 documents (Pr@5) as our evaluation metrics, which respectively measure the system's overall retrieval accuracy and its performance for those documents that are most viewed. We pool together the queries and judgments of all four users, so that the evaluation result will be a weighted average over these users, with the number of testing queries of each user as weights. We also report the performance for fresh and recurring queries separately, because they display very different behaviors.

3.5.1 Effects of Using Different Search History Components

We first study how useful each search history component (i.e., query, documents and clickthrough) are as search context. Table 3.2 shows the performance of using only certain components with the equal weighting and EM estimation methods. The rows "Contextless", "Equal", "EM" correspond respectively to the baseline method of using only query text, equal weighting and discriminative weighting with EM estimation. The text in parentheses indicate which component is used. For equal weighting, we set λ_{q_k} to 0.1 for fresh queries and 0.02 for recurring queries, which perform better than other values.

We find that from the search history, queries alone usually does not help (except in the case of recurring queries with equal weighting), probably because query texts are too short to make useful search context. In contrast, result documents are able to improve retrieval performance, especially for recurring queries. Finally, clicked results yield the highest increase in search accuracy, suggesting the usefulness of clickthrough as implicit feedback.

The fact that both result documents and clickthrough bring improvement in retrieval performance prompts us to combine them by setting $\sigma_C = 20\sigma_{NC}$ in (3.5), so that both result documents and clickthrough are used in the computation of the history model. How-

Table 3.2: Effects of using different search history components

	Fresh		Recurring	
	MAP	pr@5	MAP	pr@5
Contextless	0.371	0.233	0.265	0.138
Equal (query)	0.355	0.228	0.309	0.206
Equal (docs)	0.387	0.264	0.396	0.231
Equal (clickthrough)	0.394	0.258	0.470	0.250
Equal (combination)	0.391	0.261	0.460	0.244
EM (query)	0.368	0.237	0.257	0.138
EM (docs)	0.404	0.275	0.390	0.244
EM (clickthrough)	0.425	0.274	0.772	0.331
EM (combination)	0.430	0.271	0.766	0.325

ever, we do not observe performance gain over using only clickthrough.

3.5.2 Comparison of Contextual Models

Table 3.3: Retrieval accuracy of different methods

	Fresh		Recurring	
	MAP	pr@5	MAP	pr@5
Contextless	0.371	0.233	0.265	0.138
Equal	0.391	0.261	0.460	0.244
Cosine	0.409	0.265	0.705	0.325
EM	0.430	0.271	0.766	0.325
Hybrid	0.420	0.268	0.802	0.331

Table 3.3 shows the retrieval accuracy of different methods. The rows "Contextless", "Equal", "Cosine", "EM", "Hybrid" correspond, respectively, to the baseline method of using query text only, equal weighting, discriminative weighting with cosine similarity, discriminative weighting with EM estimation and the hybrid method. In the methods that use search history, we combine result documents and clickthrough by setting $\sigma_C = 20\sigma_{NC}$ as before.

We observe that all contextual methods perform better than the contextless one, indicating that long-term history indeed provides helpful search context. We also find that recurring queries get a lot more improvement from the use of search context than fresh queries, because by nature recurring queries have more relevant search history available as

implicit feedback. As we have expected, the discriminative weighting methods outperform the equal weighting one, proving the advantage of selective use of search history. Finally, we note that the EM estimation method achieves higher retrieval accuracy than the cosine similarity method, and the hybrid method has comparable performance.

3.5.3 Effects of Using Different Search History Lengths

To find out whether recent search history is most useful and whether remote search history helps, we truncate the search history to different lengths (e.g., one day back from the current query), and plot the change of MAP (of the EM weighting method) with respect to time cutoff in Figure 3.1.

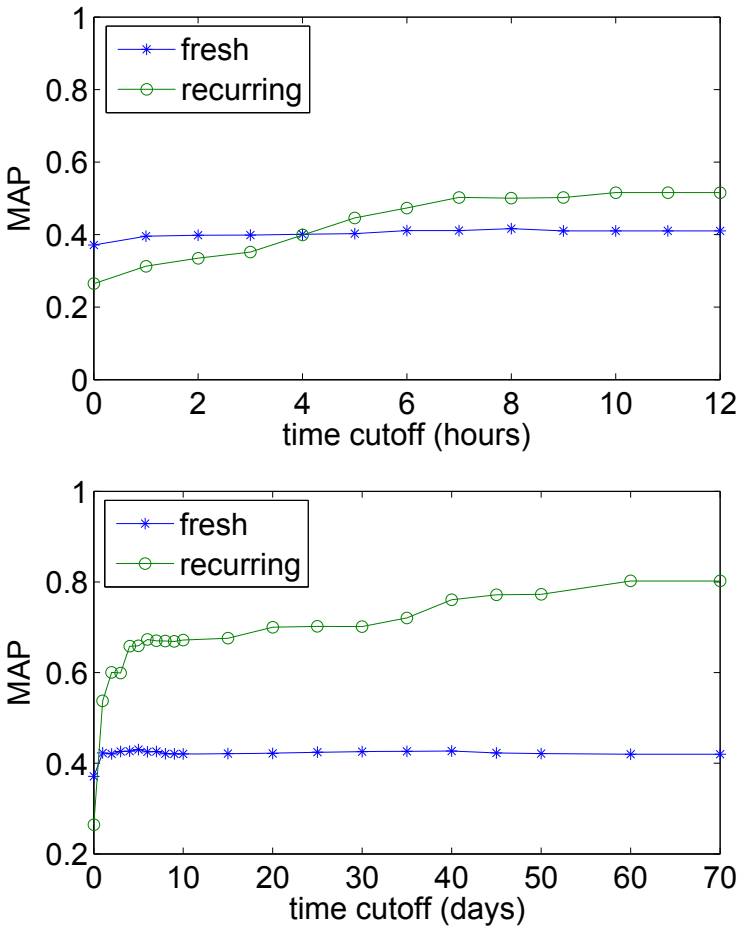


Figure 3.1: Effects of different search history lengths: within 12 hours (top) & within 70 days (bottom)

We see that for fresh queries, the dominant increase in MAP comes from the most recent history (especially within one hour), while for recurring queries, although recent history is clearly more important, remote history also contributes to the improvement in retrieval accuracy. We believe the difference is because recurring queries are more likely to reflect a user's long-term interests, and thus have more relevant history to leverage.

Chapter 4

Long-term Implicit Feedback for User Modeling and Recommendation

As discussed earlier, implicit feedback in a user's long-search search history contains rich clues about the user's interests and preferences, thus are potentially an important source for learning about the user. User interests discovered from search history not only directly facilitate understanding of the user's search behavior, but also can be leveraged to improve future search results, or make recommendations on new information items like news and products. In this chapter, we propose to mine a user's long-term search log to discover Long-lasting Exploratory Interest Patterns (LEIP).

A long-lasting interest pattern is a sequence of related searches spanning a long period of time. Note that queries in the sequence do not need to be consecutive; all that is required is they share a similar information need and be ordered by time. Intuitively, a long-lasting, or recurring interest pattern represents a strong, stable interest of a user, which can be a hobby that inspires much exploration or just a frequent need to navigate to specific web pages. This is in contrast to a short-term, passing information need, such as finding information about a city that the user is visiting, which generally only lasts for a short period of time, and with a low likelihood to be repeated. Since short-term information needs are often ad-hoc, we are more interested in long-lasting patterns, which are more reliable for us to learn about the user's interests, and to make robust recommendation that the user would presumably be interested in.

Furthermore, we are particularly interested in mining long-lasting exploratory interest patterns. An exploratory interest pattern is one that represents an exploratory information need, such as learning about a topic or category, either in depth or wide coverage, or keeping track of an event or item. For such an information need, a user can be expected to explore the information space broadly, and the scope of relevant documents tends to

enlarge as time goes by, which is often reflected by a diverse array of queries and clicks. This is in contrast to a focused interest pattern, where the user's information need is fixed on a narrow subject. For example, users often issue queries as a convenient "bookmark" to repeatedly visit a certain web page like "Yahoo mail". This pattern provides little value for learning about the user's interest, in contrast to truly exploratory patterns like one generated by a video game player searching for game information. Clearly, in the latter case, the user can significantly benefit from automatic recommendation of new games. The main motivation for mining LEIPs is to enable recommending new relevant information to a user proactively based on the discovered LEIPs. Discovering such interest patterns has also other applications such as understanding user behavior and improving search results of related queries by the user to achieve personalized search, which are not the focus of this paper.

This chapter builds upon many results from Chapter 3, as both of them rely on implicit feedback from a user's long-term search history and aim to capture the user's search interests and preferences using language models. For example, we continue to represent the information need in individual searches using the unit history model from Chapter 3, and still find using clickthrough works much better than only query. However, unlike search tasks, recommendation is queryless, in that the built user model is not combined with a new query to rerank its results, but used directly to filter new documents without a query. The memory-based method in Chapter 3 to compute a query-specific history model will not work; instead we represent the user's interests using many topic language models, corresponding to different aspects of user interests. Also, we conduct a simulated user study, which we show is a feasible evaluation strategy alternative to real user studies like the one we did in Chapter 3.

The rest of the chapter is organized as follows. In Section 4.1, we formally describe the task and present our methods to discover LEIPs. We discuss how to recommend new information using LEIP in Section 4.2. We describe our experiment design and analyze the evaluation results in Section 4.3.

4.1 Discovering Long-lasting Exploratory Interest Patterns

Given search history from a user (notations are same as in Section 3.1), we would like to discover search patterns that represent the user's interests. More specifically, a search interest pattern P_k is represented as a set of relevant searches $S_{k1}, S_{k2}, \ldots$ that together capture the user's interest in a particular aspect such as auto or computer science. Additionally, we associate with it a language model LM_k that models word usage in searches matching the pattern. Thus, we write $P_k = (\{S_{ki}\}, LM_k)$.

Aside from being long-lasting and exploratory, a well-formed LEIP should also possess the basic properties of *completion* and *cohesion*. For completion, we would like the search patterns to cover as many aspects of user interests as possible, so that nothing important is left behind. For cohesion, searches in a pattern should be similar to each other, as they are about the same interest. We would like similar searches to be in the same pattern and dissimilar searches to be in different ones, like what is required in clustering. So naturally we base our approach on clustering of searches, which tries to make the discovered patterns complete and coherent.

It is important to recognize the challenges in our task of LEIP discovery that can cause a casual application of clustering algorithm to perform poorly:

First, there are many ad-hoc searches that are found in a user's search history that constitute "outliers" not belonging to any cluster. Thus the clustering algorithm should be able to identify outliers; including them in output clusters can hurt the coherence of the pattern.

Second, search log is a very special type of literature, as the text associated with each search is very sparse: the query, which directly reflects a user's information need, tends to be very short, usually just 2-3 words long. This makes it difficult to compare the similarity between two searches. For example, 'auto dealer" and "used cars" have no overlapping words, though they are probably related. To solve this problem, we need to use other evidences to provide a smoother model of user intent than the one constructed only from query. In our approach, we use all results for pseudo feedback and clicked results for implicit feedback.

Third, queries usually group into sessions, which consist of one or more related queries issued near each other in a short time span. They provide a hint to the clustering algorithm that some queries should be assigned to the same cluster. Note that this session membership is like time locality, but not the same. Two queries with short time gap are more likely to be related, but since we are looking for long lasting interest patterns, even if two queries are far apart, they should still be grouped together as long as they are closely related. To detect session boundaries, we use a simple approach of setting time and query similarity thresholds.

4.1.1 Clustering Algorithm

We represent the user information need behind a single search S by a TF-IDF vector v [74], in which the i-th element records the TF-IDF weight of term w_i:

$$v(w_i) = \text{tf}(w_i) * \text{idf}(w_i) \tag{4.1}$$

where $\text{tf}(w_i)$ is the normalized frequency of w_i in S, and $\text{idf}(w_i)$ is the inverse document frequency of w_i. Here the choice to represent search as a TF-IDF vector rather than a unit history language model (as in Section 3.3) is solely for the convenience of computing cosine similarity in the clustering algorithm.

As discussed earlier, in addition to query text, we use all results for pseudo-feedback, and the clicked ones for implicit feedback, so as to construct a smoothed user search need model. More specifically, we let

$$\text{tf}(w) = \#(w, q) + \sum_{d \in NC} \#(w, d) \cdot \alpha + \sum_{d \in C} \#(w, d) \cdot \beta \tag{4.2}$$

where $\#(w, q)$ and $\#(w, r)$ is word w's frequency in query q and result d, NC and C are sets of non-clicked and clicked results, and α, β are weights that are set to 0.5 and 0.01 empirically. This formula is very similar to the unit history model in Section 3.3.

For *idf*, we calculate it using the formula from [30].:

$$idf(x) = \left(\frac{N+s}{\#(w)+s}\right)^{0.35} \tag{4.3}$$

where N is the total number of documents in the WT2G corpus, and $\#(w)$ is the number of documents containing w. s is needed to cover those cases where the word is not found in the background corpus, and we set it to 100 as WT2G is not very large.

We use agglomerative clustering, as this is a simple algorithm with reasonable performance, and more importantly, it does not require us to predefine a certain cluster number. More sophisticated clustering algorithms such as [29, 105] can be used, but improving it along this line is orthogonal to other techniques in this paper, and does not affect our conclusions.

We calculate the similarity between two searches using cosine similarity:

$$\cos(S_i, S_j) = v_{S_i} \cdot v_{S_j} / (|v_{S_i}| \cdot |v_{S_j}|) \tag{4.4}$$

where v_S is the TF-IDF representation of S as calculated above. And we use average-link [95] to compute similarity between two clusters:

$$\text{sim}(C_k, C_l) = \sum_{S_i \in C_k} \sum_{S_j \in C_l} \cos(S_i, S_j) / (|C_k| \cdot |C_l|) \tag{4.5}$$

We stop merging clusters when it falls below 0.1.

The clustering algorithm is an offline clustering algorithm, with a computational complexity of $O(ln^2 \log n)$, where n is the number of searches, and l is the maximum vector size (we truncate search vectors to $n = 50$ terms). On a P4 2.8G machine, it takes about 5 seconds to cluster 1000 queries. In a real production system, however, a user's search history comes in a stream, and from time to time (when enough new search history has been accumulated), the user's interest patterns need to be updated with the newly available information. To avoid re-computation from scratch, an incremental, on-line clustering algorithm is desired. One could modify the original algorithm by only using existing sum-

mary information for past searches, instead of keeping all of them in memory. [107, 12] are examples of building online clustering algorithms based on offline versions.

We use sessions as seed clusters for the agglomerative clustering algorithm, i.e., queries in the same session are initially grouped in the same cluster already. Without session information, these queries may not end up in the same cluster due to the average-link based similarity computation. Also, a reduction of seed cluster number helps to reduce computation. We also join recurring appearances of the same query in a single cluster. Because there are usually a large number of recurring queries (e.g., the user may use "Yahoo mail" to reach the site frequently), this constitutes a nontrivial saving of computation.

4.1.2 Filtering and Ranking Interest Patterns

At the finish of the clustering algorithms, we usually have a lot of clusters, many of which just contains a single unique query, being either an outlier or a series of recurring navigational queries. Even among the truly exploratory patterns, there is a need to rank them by how frequently the user shows interest in them. Therefore, following clustering we have a postprocessing step that filters clusters to only keep long-lasting and exploratory ones as interesting patterns

Long-lastingness Measures

To characterize how long-lasting an interest pattern is, the most straightforward measure is the number of queries in the pattern. This one has the drawback that navigational queries, which are less interesting, produce high scores, so an alternative is to measure the number of unique queries in the pattern to penalize recurring queries. Also, since a pattern tends to last longer if its queries are distributed in many sessions than concentrated in few ones, we would like to penalize queries in large sessions. We can measure the number of sessions in the pattern, which gives each session a frequency weight of 1 no matter how many queries they contain.

Finally, we propose a measure that penalizes both recurring queries and queries in large

sessions, which we call normalized query count:

$$\sum_{q_i \in Q} \frac{1}{\max(M(q_i) + \alpha, N(q_i) + \beta)} \tag{4.6}$$

where $M(q_i)$ is the number of queries that are identical to q_i, and $N(q_i)$ is the number of queries in q_i's search session. This is like a TF scoring function in which we favor higher frequencies, but the marginal score increase decreases when we increase the frequency ($M(q_i)$ or $N(q_i)$). In practice we find setting α and β to 0 works best, which effectively ignores recurring queries and gives each session a weight of 1.

Exploratoriness Measures

Here we measure how exploratory an interest pattern is. For an exploratory pattern, the user tends to explore many aspects, rather than focusing on a single item. Most likely, the user will continue to issue new queries and click on new urls. Subsequently, the user's queries and clicked urls tend to be diverse. This gives us some insight into how to capture this exploratoriness.

Query novelty: The more exploratory a search interest pattern is, the more likely the user is to issue new queries. Thus we can use the likelihood for a user to issue new queries as a measure of query exploratoriness.

Suppose the user has already issued N queries $Q_1, Q_2, \cdots Q_N$, out of which M are unique. Let the binary indicator variable $U(Q_k)$ be 1 if Q_k is different from all previous queries $Q_1, Q_2, \cdots Q_{k-1}$ and 0 otherwise. If we make the assumption that the probability of Q being new is independent of its occurrence order and its textual content, then the total probability of seeing M unique queries out of N is $\delta^M(1-\delta)^{N-M}$, where δ is the probability of a query being novel (given the independence assumption). It is easy to infer that the value of δ that maximizes the above likelihood is M/N, if we use a uniform prior for δ.

The independence assumption has flaws. For example, we may think a user is more likely to repeat a query when there have been a larger number of past queries. Therefore, a pattern with a large number of queries may get penalized. One might use a more complicated

model such as the Chinese Restaurant Process (in which we make identical queries to sit at the same table), but in reality we find that Pearson's Correlation Coefficient between division-based query novelty and query number is only -0.045 for patterns with at least 5 unique queries, so the current formula is sufficient.

Query term novelty: Similar to query novelty, if we consider each query term as a unit instead of a whole query, we can define query term novelty as

$$\#\text{ unique query terms} \,/\, \#\text{ total query terms}$$

This formula gives the probability of each query term being novel. Pearson's correlation coefficient between query term novelty and query term number is 0.022, so the independence assumption should not pose a serious problem.

Query term novelty is very similar to query novelty, except that query novelty tends to over-estimate but query term novelty tends to under-estimate. This is because in the computation of query novelty, a query is considered new whether it differs entirely or just by a term from previous queries. While for query term novelty, it is measured at term level, but sometimes it reports no novelty when there is. For example, for the three queries "NCAA", "NBA schedule" and "NCAA schedule", the last query does not introduce any new terms, but it still reveals new information need.

Clicked url novelty: If a search interest pattern is exploratory, not only will the user issue many different queries, but also the urls clicked by the user tend to be diverse. For a navigational query, the user may always click on the same url, but for an exploratory one, she is likely to try out many different urls. Thus, we can also define clicked url novelty and use it to capture the exploratoriness of a search pattern.

As before, clicked urls can be defined as

$$\#\text{ unique clicked urls} \,/\, \#\text{ total clicked urls}$$

Note that clicked url novelty is less direct than query novelty in characterizing exploratoriness. While the action of issuing a query is directly motivated by the information

need, the action of clicking on a certain url not only depends on the user's information need, but is also affected by the quality of the results returned by the search engine. It is possible that an exploratory pattern contains many different queries, but not so many different clicked urls simply because the result ranking is poor. Also, the correlation with regard to clicked url number, however, is quite strong (-0.355), so large patterns tend to be penalized.

Query term entropy: Entropy is a standard measure for characterizing the uncertainty in a probability distribution. Thus we can use it to measure the diversity of user actions in a search pattern. We pool all the query words in an interest pattern and then compute entropy from the histogram. Generally speaking, if an interest pattern is exploratory, the word use in the member queries tends to be rich and diverse, yielding a high query term entropy. A larger entropy usually means higher diversity of the interest pattern. However, entropy tends to characterize the "evenness" of a distribution. Let A be a pattern having two queries with frequency 20-20, and B be a pattern having 5 queries with frequency 1-1-1-1-20. B is more exploratory than A, because both queries in A are repeated a lot (likely to be navigational), while only one query in B is repeated. But in this case, A will have a larger entropy.

Click url entropy: We can also compute the entropy of the clicked urls as a more principled way to measure their diversity. Given a pattern, we build a histogram which gives click frequency for each clicked url, and use it compute its entropy. It has similar problems as query term entropy.

There are some existing works [46, 54] on predicting if a query is navigational or not. For example, [54] uses click distribution as one of the features. But the click distribution in that case is calculated from collaborative filtering data from different users for the same query, while here we use (possibly) different queries from a single user. [100] uses a set of history features to predict whether the user's interest on a query is high enough to have the query registered to receive new results. Its difference with this work is that it tries to find queries that are interesting yet unfulfilled, rather than LEIPs.

4.2 Recommending New Information Using Search Patterns

Given a set of pattern $\{P_k\}$ extracted from search history, we can use them to recommend items for the same user. But first of all, for each pattern, it is desirable to summarize the individual searches using a language model, which is a distribution of terms, more concisely representing the user interest as reflected by these queries. The general model can then be used for all kinds of information retrieval or text mining tasks. Here we can use it to score a set of new information items to produce a ranked recommendation list.

In order to produce the pattern language model θ_P, we would like to first estimate a separate language model θ_i for each search S_i in the pattern P, which can then be combined to generate the overall θ_P.

θ_i can be computed using the unit history model (3.5) in the previous chapter. Alternatively, we may assume S_i is generated from a topic model θ_T and a background model θ_B, so the log likelihood of S_i is

$$\sum_w c(w) \log \Big(\mu P(w|\theta_B) + (1-\mu) P(w|\theta_T) \Big)$$

where μ is the mixture coefficient, and we tune it to 0.9 in the experiments. We estimate θ_B from the WT2G corpus, and we should select θ_T to optimize the log likelihood of S_i. An expectation-maximization (EM) algorithm for this is provided in [102]. We can then just let θ_T be the language model for S_i.

The pattern language model can be the weighted average of the search language models:

$$\theta_P = \frac{\sum_{S_i \in P} \theta_{S_i} \cdot \lambda_i}{\sum_{S_i \in P} \lambda_i} \tag{4.7}$$

where λ_i is the weight assigned to S_i. For example, we can use a time-decaying scheme to give recent queries more weight. We tried this but failed to see improvement, probably because our search log does not span a long enough period for time weighting to show advantages.

4.3 Experiment Results

4.3.1 Data Collection

We use a web search log data set from the AOL search engine [62]. The log spanned three months. Each query is associated with a user id, so we can recover the complete search history of a particular user within this period. Because the search log does not come with results, we crawled the search engine for up to 20 result snippets per query[1]. We did this for 100 users with highest number of clicked sessions (using the previous definition of session), and got 122487 queries, which means these users on average had 1200+ queries in the three-month period. The result snippets have an average length of 29 words, and are indexed together as a collection C, which can be considered as a small sample of the web. We divide the user's search history into two equal halves, a "training" set and a "test" set for the purpose of simulated studies, as detailed later. For each user, the training set has the same number of queries as the test set. We use the training set to mine the long-lasting exploratory interest patterns. More specifically, we cluster the searches, and compute various scores on them using the measures defined in Section 4.1.2. We discard patterns with fewer than 5 unique queries as they are unlikely to be very informative.

In our experiments, we first evaluate the effectiveness of our methods to discover interesting LEIPs, then examine whether useful recommendation of new information can be made using these LEIPs.

4.3.2 Evaluating LEIP Discovery

We have proposed many measures to evaluate whether a pattern is long-lasting / exploratory. Each measure produces a ranking of patterns based on how they score against it. We want to know which measure is more accurate, i.e., which of them better orders clusters in accordance to their long-lastingness / exploratoriness. One way is to recruit people to manually label whether a pattern is long-lasting or exploratory, e.g., on a 1-5 scale. However, since long-lastingness and exploratoriness lack rigorous definitions, it is hard to give

[1] Although the ranking has changed a lot due to the time difference between logging and crawling, we are still able to find 89449 out of 143932 clicks in the new top 20.