INTENT MODELING AND AUTOMATIC QUERY REFORMULATION FOR SEARCH ENGINE SYSTEMS

BY

HUIZHONG DUAN

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, 2013

Urbana, Illinois

ABSTRACT

Understanding and modeling users' intent in search queries is an important topic in studying search engine systems. Good understanding of search intent is required in order to achieve better search accuracy and better user experience. In this thesis work, I identify and study three major problems in the subject: ambiguous search intent, ineffective query formulation and vague relevance criteria. To systematically study these problems, the thesis consists of three parts. In the first part, I study search intent ambiguity in search engine queries and propose a click pattern-based method that captures ambiguous search intent based on behavioral difference rather than semantic difference. Analysis shows that the proposed method is more accurate and robust in measuring query ambiguity. In the second part, I study how to provide query formulation support to facilitate users in expressing search intent. Query completion and correction, and syntactic query reformulation are proposed and studied in this part. Experiments show that the proposed query formulation support methods can help users formulate more effective queries and alleviate search difficulty. In the third part, I study how to model search intent so that we can gain insights about users' behaviors and leverage the knowledge to improve search engines. Two topics are studied in this part: modeling search intent with data level representation and discovering coordinated shopping intent in product search. It is shown that the proposed methods can not only discover meaningful user intent but also improve search and other related applications. The proposed models and algorithms in the thesis are general and can be applied to improve search accuracy in potentially many different search engines. As a systematic study on intent modeling and automatic query reformulation in search engine systems, this thesis work also provides a road map to future exploration on intent understanding and analysis.

ACKNOWLEDGMENTS

I want to express my sincere gratitude to my advisor Professor ChengXiang Zhai, who supported and guided me through my Ph.D study. His knowledge, vision and passion in academic has always inspired me to pursue my research to the full extent. He offered invaluable advices to my work. This thesis would not have been possible without his help. He is also kind, patient and encouraging to me. His way of advising has made it possible for me to work and improve myself more effectively. From him I have learned not only the knowledge, but also the mindset to become an independent researcher.

I would also like to thank my doctoral committee members, Professor Jiawei Han, Professor Dan Roth and Doctor Emre Kiciman, for their valuable guidance on my study and research, as well as constructive suggestions on this dissertation.

I owe gratitude to my colleagues and friends, Yanen Li, Rui Li, Paul Hsu, Yue Lu, Hongning Wang, Yunliang Jiang and Mianwei Zhou. They offer valuable help to my research. I also want to thank all my colleagues in Database and Information System (DAIS) group, and all my friends in UIUC.

I am deeply indebted to my father Yunhai Duan and my mother Yaqin Han. Without their love and support, none of my achievements is possible. Last but not least, I am grateful to my wife Zhen Wu and my daughter Emma Duan for their love and care, and for the happiness they bring into my life.

To the human race.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 RELATED WORK 6
- 2.1 Search Intent Analysis 6
- 2.2 Supporting Query Formulation 8
- 2.3 Intent Modeling in Structured Entity Retrieval 10

PART I Modeling Ambiguous Search Intent 13

CHAPTER 3 MODELING AMBIGUOUS INTENT WITH CLICK PATTERNS 14
- 3.1 Introduction 14
- 3.2 Click Patterns 15
  - 3.2.1 Definition 16
  - 3.2.2 Modeling and Identifying Click Patterns 16
  - 3.2.3 Exploring Click Patterns in Search Logs 17
- 3.3 Click Patterns and Query Ambiguity 20
  - 3.3.1 Pattern Entropy 21
  - 3.3.2 Properties of Ambiguity Metrics 22
- 3.4 Classifying Ambiguous Queries 26
  - 3.4.1 Problem Setup 26
  - 3.4.2 Classification Results 27
  - 3.4.3 Case Study 28
- 3.5 Click Patterns and Query Recommendations 28
  - 3.5.1 Problem Setup 29
  - 3.5.2 Query Recommendation Results 30
  - 3.5.3 Case Study 30
- 3.6 Conclusions 31

PART II Supporting Query Formulation 32

CHAPTER 4 QUERY COMPLETION AND CORRECTION 33
- 4.1 Introduction 33
- 4.2 Problem Formulation 33
- 4.3 Transformation Model 34

4.3.1 Model Estimation 37
4.3.2 Model Pruning 39
4.3.3 Model Smoothing 39
4.3.4 Mixture Models 40
4.3.5 Discussions 41
4.4 Search 41
4.4.1 Trie 42
4.4.2 A* Search 43
4.4.3 Pruning 45
4.4.4 Thresholding 45
4.5 Experiment 46
4.5.1 Datasets 46
4.5.2 Evaluation Metrics 48
4.5.3 Experiment Results 49
4.6 Conclusion 54

CHAPTER 5 QUERY REFORMULATION WITH SYNTACTIC OPERATORS 55
5.1 Introduction 55
5.2 Syntactic Operators for Improving Search Results 58
5.3 Automatic Reformulation with Syntactic Operators 59
5.3.1 Features 60
5.3.2 Combining Operators in Prediction 63
5.4 Experiments 63
5.4.1 Experiment Setup 63
5.4.2 Experiment Results 64
5.4.3 Cases Studies 66
5.5 Conclusion 67

PART III Deep Intent Modeling for Structured Entity Retrieval 68

CHAPTER 6 MODELING QUERY INTENT WITH ENTITY STRUCTURES 69
6.1 Introduction 69
6.2 Probabilistic Query Model for Ranking Product Entity 72
6.2.1 Probabilistic Entity Ranking Based on Query Generation 73
6.2.2 Model Estimation Based on Entity Specifications 77
6.2.3 Improve Estimation by Leveraging Associated Text Data 79
6.2.4 A Mixture Model of Review Text Data 79
6.2.5 Maximum a Posterior (MAP) Estimation of the Mixture Model 81
6.2.6 Smoothing and Interpolation 82
6.2.7 Indexing and Search 83
6.3 Experiments 85
6.3.1 Datasets and Evaluation Metrics 85
6.3.2 Experiment Results 86
6.3.3 Search Efficiency 90
6.4 Other Applications Using the Probabilistic Query Model 90

6.4.1 Facet Generation 90
6.4.2 Query Independent Facet Generation 92
6.4.3 Query Specific Facet Generation 93
6.4.4 Review Annotation 94

CHAPTER 7 DISCOVERING COORDINATED SHOPPING INTENT IN PRODUCT SEARCH 95
7.1 Introduction 95
7.2 Discovering Coordinated Shopping Intent in Product Search 98
7.2.1 Topic Modeling Approach Adapted for Intent Discovery 100
7.2.2 Coordinated Representation of Shopping Intent in Product Search 102
7.2.3 Joint Mixture Model for Discovering Shopping Intent 103
7.2.4 Model Estimation 104
7.2.5 Combining Similar Intent Models 105
7.3 Discovered Shopping Intent 106
7.4 Applications 111
7.4.1 Intent-Oriented Query Ambiguity Analysis 111
7.4.2 Improving Product Search 114
7.4.3 Improving Product Recommendation 116
7.5 Conclusion and Future Work 118

CHAPTER 8 SUMMARY AND FUTURE DIRECTIONS 120
8.1 Summary 120
8.2 Discussions and Future Directions 123

REFERENCES 125

CHAPTER 1

INTRODUCTION

Search engine systems exist everywhere in the world of data. As an important tool for information access, search engine systems are built on top of various data sources, including unstructured text data as well as semi-structured/structured data. In all the search engine systems, the utility of a system is directly affected and largely determined by its ability to understand a user's search intent. Good understanding of search intent usually leads to better search accuracy and better user experience.

However, search intent is a complicated issue. Accurate understanding and modeling search intents is difficult in many ways.

Unclear/Ambiguous Intent. First, it is often that a user does not have a clear search intent. This is especially true for exploratory search, where the user doesn't have a fixed goal. In this case, a user's query is only a rough hint about what exactly is interesting to the user.

Ineffective Query Formulation. Second, even when a user's information need is clear, there are at least two reasons why a user may not be able to describe the information need (query intent) very clearly. First, there is usually a vocabulary gap between the user and the documents, making it difficult for the user to use the right words for the query. Second, the user may not realize that there are many distracting documents in the collection, and additional constraints on top of a keyword query may be needed (e.g., two words should occur next to each other) in order to retrieve some particular content.

Vague Relevance Criteria. Finally, even when a user can describe an information need clearly, the query by itself can only convey a vague criteria for relevance matching, as there is typically difference in intent and data representation. Therefore, it is still challenging to obtain a deep level of representation of intent that allows us to easily match with the relevant data (i.e., obtain a document-level representation of intent).

In this work, I propose to tackle these challenges in intent modeling and exploit search logs to help understanding user intent. I will further leverage the understanding of intent to improve retrieval accuracy through query reformulation. Specifically, I will study the following topics:

- **Modeling ambiguous search intent.** People issuing the same query may target very different information. Ambiguity of search intents exists ubiquitously in search engine systems. Understanding such ambiguity is important for intent modeling and will largely benefit the search experience. Traditional methods in ambiguity analysis assumes that each clicked result represents an unique intent. However, there are many search tasks, such as comparison shopping and research survey, where a user's intent is to explore many documents. In these cases, the simple assumption of a one-to-one correspondence between clicked documents and user intent breaks down. In this work, I will study the modeling of search intents from the perspective of behavior analysis. Specifically, the notion of "click pattern" is proposed to serve as the fundamental unit for intent analysis. Click patterns can be effectively discovered by analyzing user behavior in click-through logs. The model will lead to a better understanding and a more accurate measurement of ambiguity in search intent.

- **Supporting Query Formulation.**

  - **Query completion and correction.** One major difficulty that users encounter in expressing their search intent is the vocabulary gap between the users and the data. Due to the vocabulary gap it is difficult for end users to formulate effective queries based on their search intents. In this work, I will study query completion and correction to help users clarify their search intents. The idea is to provide real time feedback to the user while the query is being entered. This way, by detecting the most likely intent based on active input of the user, we can help the user complete and refine their search intent in real time. Consider a user entered a keyword "schwarzenegger" to the search interface, if we can infer that the most possible targeted query is "shwartzeneger movie 2013", it can provide the query in a drop down list so that the user can directly enter the complete query (by selecting using the arrow keys or the pointing devices) without typing in the rest. To alleviate the problem of vocabulary gap, we further consider the cases where users may not be able to spell correctly. For instance, "schwarzenegger" can be misspelled as “shwartzeneger” and the system should still provide the suggestion of “shwartzeneger movie 2013” as the most possible intent query. To achieve these goals, I will study the modeling of similarity between search intents. A generative joint sequence model will be adapted for the problem. Complex similarity models will be estimated and studied. Efficient search algorithms will also be studied for finding the most possible query completion and correction based on inferred search intent from a (partial) user query.

  – **Query reformulation with syntactic operators.** Although the use of keyword query is prevailing in IR systems, it has not been well understood whether it is sufficient for expressing search intents. Indeed, modern search engines support a range of advanced syntax beyond keyword queries. These syntax are designed to further clarify the users' search intents by imposing certain constraints on one or more keywords. For instance, the query “change “default java” +unix” states that the phrase “default java” must be matched as a phrase rather than as separate keywords, and additionally the keyword “unix” must be matched. Although they are both intended for the same information, the lightly modified query with the syntactic operators (quotation mark for phrase match and plus sign for necessity) turned out to more effective in retrieving the relevant information than the plain keyword query on major search engines. The lack of such semantic constraints in keyword query makes it insufficient to express certain search intents. In this work, I will systematically study the advantage of query syntax in search intent expression. I will also study the automatic reformulation of search queries with syntactic operators based on the inferred search intents. The problem will be casted into a supervised learning to rank problem and a set of effective features will be proposed and studied.

- **Deep intent understanding/modeling in structured entity retrieval.** Even if search intent can be clearly conveyed in query, it is still challenging to obtain a deep level of intent representation which matches with the document representation in the data. Such a representation is usually critical in relevance matching because it determines the relevance criteria. Understanding and modeling of deep intent representation is, however, difficult due to the various types of search intent and document representation.

On the other hand, structured entity retrieval, an emerging and important issue in information retrieval, is of particular importance in e-commerce, medical information system and many other search systems. The difference in data representation (structured data of product entities vs. unstructured text of Web documents) raises challenges as well as new opportunities in intent modeling. In this work, I use structured entity retrieval as a platform for extending my study on deep intent understanding and modeling. Particularly, I study the following two topics.

- **Modeling query intents with entity structures for product search.** In the scenario of product search, the objects to be retrieved are structured entities defined by a set of attributes and values. For example, products are defined by their specifications. Each specification consists of a name and a value, which can be either numeric or categorical. For instance, a laptop can be represented by its brand, cpu speed, screen size, etc. In the scenario of product search, a user's search intent is strongly tied with the specifications of products. However, the query language users use to describe their search intent is usually in an unstructured text format. Because of such difference, it is necessary to obtain a deep level of representation of search intent with the entity structure so that we can understand what users are shopping for given the vague relevance criteria conveyed in the user query. In this work, I study the problem of intent matching between user query and product information in the form of probabilistic models for product search. I show how this can improve the accuracy of product search as well as benefit important related applications such as automatic facet generation.
- **Discovering coordinated shopping intent in product search** In the previous part of work, I study the problem of query intent modeling in product search by matching between user queries and structured product information. Although the proposed method is effective for improving the accuracy of entity retrieval, it does not provide much insight in understanding user intent as it analyzes intent on a per query basis. In reality, user intent is a more complicated issue, especially in product entity search. Different users may use the same query to express different intent, or use different queries to express the same intent. To capture this, we need to model search intent at a higher conceptual level. This calls for an explicit and accurate representation of search intent. In

this part of work, I propose a novel coordinated intent representation, where each intent is collectively represented by query terms and product attributes. A joint mixture model is then proposed to automatically discover such intent by analyzing search engagement logs. I show that the proposed method can effectively discover distinct, coherent and meaningful intent models, and with the interpretable coordinated intent representation, we can gain insights into the user shopping preferences. The discovered intent models can not only be directly incorporated in product search to improve search accuracy, but also be leveraged in search related applications such as query ambiguity analysis, product recommendation and search result diversification and personalization to promote the overall search experience.

Thesis Organization: The rest of the thesis is organized as follows. Chapter 2 surveys the related literature. Chapter 3-7 are divided into three parts. In Part I (Chapter 3), I present one published work on *modeling ambiguous search intent*. In Part II (Chapter 4 and Chapter 5), I present two published work on *supporting query formulation*. In Part III (Chapter 6 and Chapter 7), I present one published work and one work in submission on *understanding and modeling of deep search intent*. Finally, the thesis work is summarized in Chapter 8.

CHAPTER 2

RELATED WORK

2.1 Search Intent Analysis

User intent/query intent analysis has been the subject of much research in recent years, especially for the purposes of search personalization and vertical search engine selection.

Understanding user's intent in search queries helps identify queries that require more personalized search results. Song et al. proposed to summarize queries as in three categories: ambiguous query, broad query and clear query [64]. They found that through topical categorization, the three types of queries are to a certain extent distinguishable according to the topical distribution. They classified the queries into these categories and estimated that 16% of queries are ambiguous in sampled logs. Teevan et al. studied how to automatically identify ambiguous queries [69]. They proposed "potential for personalization curve" for measuring the ambiguity of search queries. They measure the ability of one ranking list of search results satisfying multiple users. They show that the implicit measure (using click-through data) correlates well with the explicit measure. They also show that click entropy correlates well with "potential for personalization". Mei and Church studied the difficulty of search and personalization from an information theory perspective [52]. They used conditional entropy of URLs as an indicator of search difficulty, and compared the general search difficulty to the search difficulty when personalized with user's IP address. Their results show that personalization has huge potential in reducing search difficulty. A backoff model for personalization is also proposed where multiple layers of personalization are combined in optimizing the effectiveness.

Query intent analysis has also been studied in refining search result presentation [25] and vertical search engine selection [51, 39]. Daume and Brill proposed to group web search results based on reformulating the original query to alternative queries the user may have intended [25]. Li et al. proposed to identify queries for different vertical search engines by connecting with the close labeled queries in the click graph [51]. Hu et al. leveraged wikipedia to form query intent space, and used it to improve vertical selection[39]. However, these studies are focused on the semantic level of query intent. In our study, we approach user intent from a fundamentally different aspect by analyzing users' click behaviors.

The problem of query ambiguity has potential impact on the performance of retrieval [42, 60, 2]. The study of query ambiguity has a long history [75]. Early studies are focused on word sense disambiguation[70, 61, 24, 65, 33] with the use of dictionary and thesaurus. Allan and Raghavan studied the use of Part-of-speech Pattern to form clarification questions and reduce query ambiguity. Cronen-Townsend and Croft proposed query clarity as a measure of ambiguity [19]. Query clarity is computed as the KL-divergence of the query language model and the collection language model. The query language model is estimated from the top ranked documents of the query. Therefore, a high query clarity indicates the query is more concentrated on specific topics. Query clarity has been used often for predicting query difficulty.

Wang and Agichtein studied how to distinguish informational and ambiguous queries. They propose the use of user averaged click entropy (*average entropy*)[74]. *Average entropy* computes the average of the click entropy on the click distribution of each individual user. The assumption is that while a query may be ambiguous in general, each individual user has a clear navigational intent (which is different from others) and therefore will click on only few web pages; on the other hand, users with information seeking intent tend to click on more web pages, although the overall intent is clear. As a result, informational queries shall have higher average entropy than the ambiguous queries. However, the assumption that individual intents are clear and navigational for ambiguous queries is ungrounded. An ambiguous query can also be (completely/partially) associated with information seeking intents.

In this thesis work, I propose to study query ambiguity from a different perspective. Rather than analyzing each clicked URL individually, as was done in most previous researches, we tie the concept of query ambiguity to the users' click behaviors. We propose to discover the click patterns and use pattern entropy as a new metric for query ambiguity. To the best of our knowledge, no previous research has studied click pattern for measuring query ambiguity before.

User behavior modeling is an active research area in query log analysis. Craswell et al. studied the problem of position bias in users' click behaviors [18]. Through a large scale experiment of perturbing the search engine rankings, the best explanation for position bias was found to be a model where users view results from top to bottom and leaving as soon as they see a worthwhile document. User modeling has also been studied for search evaluation [27, 77]. Dupret and Piwowarski explored the underlying hypothesis for the mean average precision metric [27]. Yilmaz et al. proposed a new evaluation metric that uses a sophisticated user model tuned by observations over many thousands of real search sessions [77]. This work extends the study of user modeling with an exploration of fine grained user models where each query may correspond to a mixture of different types of behaviors.

2.2 Supporting Query Formulation

Research on spelling correction has a long history [22, 48, 58]. Edit distance, initially proposed by Damerau [22] and Levenshtein [48], has been widely used in generic spelling correction. More recent work on offline spelling correction tends to focus on search engine queries [14, 21, 32, 49, 66]. Cucerzan and Brill [21] studied spelling correction as an iterative process to exploit the information in query logs. Li et al. [49] explored distributional similarity of query terms to estimate the error model. Chen et al. [14] leveraged web search results to improve the performance of spelling correction on rare queries. Sun et al. [66] explored click-through data to identify user correction pairs, and applied them to build a phrase-based error model. Gao et al. [32] proposed the use of a general ranker as a generalization of the traditional noisy channel model in spelling correction, and implemented it with a distributed infrastructure to incorporate large scale data. As offline spelling correction is just a special case of online spelling correction, we consider the performance of both conditions when evaluating our system.

One of the earliest forms of auto-completion is the tab completion feature found in many command prompts. It is later extended by Darragh et al. [23] to support the prediction of general text. Recently, Chaudhuri and Kaushik [13] proposed a technique to further extended auto-completion to tolerate errors. Particularly, they made use of a simple edit distance model and performed a fuzzy search over database records to find completions. To the best of our knowledge, this is the only prior work that addresses the problem of online spelling correction. Unfortunately, with a predetermined cap on edit distance and linear lookup time with increasing data size, the algorithm is not sufficiently robust and scalable for online spelling correction for query completion.

The approach taken in this work for spelling correction is largely inspired by previous work in grapheme to phoneme transformation. Chen [15] studied conditional and joint maximum entropy models for grapheme to phoneme conversion. Taylor [68] used a hidden Markov model, where the graphemes are observations of the hidden phoneme states. Bisani and Ney [9] proposed a joint-sequence model for modeling grapheme to phoneme transformation, where graphemes and phonemes are viewed as a joint sequence generated with a Markov model. In this work, we adapt the joint sequence modeling of Bisani and Ney to model the transformation from the intended query to the observed sequence. However, whereas grapheme to phoneme conversions are strongly constrained by pronunciation rules, typographical errors do not impose any constraint on possible transformations, increasing the difficulty in model training.

Query reformulation/refinement is a broad topic on using modified versions of queries to improve search results. Our work is naturally subsumed by this topic. Under this broad topic, query expansion, query contraction, spelling correction and many other topics have been extensively studied. Query expansion aims to expand the keywords of a query with additional terms, so as to enrich the short keyword query and bridge the vocabulary gap. Recent work studied query expansion based on corpus analysis [76], concept dictionaries [56] and relevance feedback [53]. Query contraction, on the other hand, attempts to abridge long and verbose queries in order to concentrate on the relevant topics [44][5][45][43]. None of these work studied the use of syntactic operators for query reformulation as we did in this work. Guo et al. [34] employed a modified conditional random field model for simultaneously performing multiple reformulation tasks including spelling correction and query segmentation. They used human annotated queries for training, which is not directly targeted at increasing retrieval performance. Moreover, it is unclear how this method would work for alleviating search difficulty and how it could be generalized to different syntactic operators.

Beside explicit reformulation of query, there are also many studies on implicit refinement of queries, such as term proximity and term necessity. This type of refinement is usually transparent to users and more subtle to search results. For instance, term proximity has been widely adopted [57][67][11] to improve retrieval performance. Zhao and Callan proposed to estimate term necessity probability and used it to refine existing retrieval models [81]. These refinements can all be viewed as soft versions of the hard constraints imposed by corresponding syntactic operators. However, there are two major differences that clearly distinguish the explicit use of syntactic operators. First, existing work on implicit refinements are all retrieval model dependent. It requires extra work to integrate refinement scores into different retrieval models. Second, the syntactic operators of query language carry clear and well defined semantics. Therefore, they can be used by search systems to interact with users through query suggestions. Compared with blindly making refinements, it lowers the risk of disappointing users in case the refinements are not appropriate.

Another line of related work is query difficulty prediction. Cronen-Townsend et al. proposed to use query clarity as a predictor to estimate the performance of a query [20]. Hauff et al. did a comprehensive survey on various kinds of query difficulty predicators [35]. Yom-Tov et al. proposed a learning framework to estimate query difficulty. Our work of automatic syntactic reformulation is related to query difficulty prediction in that we want to estimate performance of modified queries and suggest queries with hypothetical performance benefits. Some of the features we use in our method are inspired by the work on query difficulty prediction.

My research is also strongly related to the work on negative feedback. Wang et al. used EM algorithm to estimate language models for negative feedback and use them to re-rank the rest of the results [72]. They later systematically studied the existing methods for improving search accuracy with negative feedback in different retrieval models [73]. Our method also uses the negative feedback information to assist search difficulty, but it distinguishes itself by making use of syntactic operators to introduce additional semantics. Compared with existing methods, this is a novel approach and provides better usage of the negative feedback.

2.3 Intent Modeling in Structured Entity Retrieval

There have been extensive study of supporting keyword queries in structured entity data, but most of the work assume that the returned results are a set of tuples that match all the keywords (see, e.g., [40, 1, 38]) without tackling the important problem of intent modeling, which is critical for a problem such as product entity search. Standard information retrieval (IR) models have been adapted to rank objects in databases (e.g., [37]), but the application is a straightforward use of existing retrieval models to score each single attribute, assuming the relevance definition implied in a traditional IR model, thus this line of work has not addressed the special notion of relevance in product search where we must model preferences of users. Moreover, the retrieval units are joining trees of tuples, which are not appropriate for product entity search.

Another major limitation of these previous work on supporting keyword queries is that they have mainly focused on lexical level of relevance. For instance, relevance is determined by matching keywords with different attributes of the entity. In this case, queries such as "radeon hd graphics laptop" may work reasonably well, but "laptop with dedicated graphics card" will clearly not work. As the method depends on the description of the entity, keywords that are not in the description cannot be matched well. This problem is commonly known as the " vocabulary gap". Some work have attempted to go beyond this simple notion of relevance and applied probabilistic models in IR to database object ranking [12], but the focus was on leveraging workload data to improve estimation of probabilistic models, and the queries considered are restricted to structured queries rather than plain natural language keyword queries as we address in the work. Integration of IR and database search has been considered in some previous work, particularly in developing probabilistic models (see, e.g., [28, 29, 30]). A main goal of this line of work to extend standard relational algebra in such a way that it can handle probabilistic weights required for performing IR-style ranking, thus they have not addressed how to optimize ranking of database objects for unstructured keyword preference queries.

A significant amount of work has been done on XML retrieval, which is also related to our work in that the documents dealt with have weak structures and keyword queries are handled. A comprehensive review of the work in this line can be found in [47]. However, there are many important differences between XML retrieval and product entity retrieval. One major difference is that the queries in XML retrieval tend to refer to structures whereas in product search that we are considering in our work, the queries are pure keyword queries similar to those for Web search. Moreover, the notion of relevance has a unique and different interpretation in product search, i.e., a relevant entity object is one whose attribute values match well the preferred attribute values of a keyword query.

The work in this thesis is also related to recent work on leveraging review data for ranking entities [31], in which the reviews associated with each entity are simply used as text representation of an entity, and keyword queries would be matched with these reviews directly to rank entities.

Such an approach has ignored the attribute value descriptions of product entities completely, thus they do not offer any formal model for ranking product entities in a database with product specifications. Since it *solely* relies on product reviews, it would suffer from the problem of data sparsity as reviews are only available for some product entities, and even fewer of them have enough content for estimating an accurate model. In contrast, we leverage reviews as well as search queries as supplementary data to improve estimation of probabilistic models that would otherwise have to be based on product specifications only. There are also some studies of e-commerce related applications based on product search log mining. Li et al. proposed a semi-supervised method trained with search queries and matched products for tagging query keywords with product meta-data [50]. Sarkas et al. studied a similar problem with a supervised and an unsupervised model [62]. Pound et al. studied facet discovery by mining a search log that contain structured product information [55]. These work are application oriented and utilize task specific mining heuristics. In comparison, our work is focused on deep search intent modeling with entity structure.

Part I

Modeling Ambiguous Search Intent

CHAPTER 3

MODELING AMBIGUOUS INTENT WITH CLICK PATTERNS

3.1 Introduction

Understanding and interpreting a user's search intent is the first step a web search engine must take to fulfill the user's needs. However, accurate understanding and interpretation is usually difficult because of the ambiguity in search intent. Ambiguity is typically caused by a broad query, which is usually entered by user who does not have a clear search intent in the beginning.

To solve this problem, web search engines routinely analyze the click behaviors of past searchers to extract information about query intents and are used to measure query ambiguity, influence ranking decisions and result presentation, as well as generate query recommendations.

Current query analysis techniques assume that each clicked web result provides evidence of a distinct intent. Such a simplifying assumption is problematic, however, as it ignores the many reasons why users with the same semantic intent may click on more than one web result: they may have a high-recall information need, such as when users are comparison shopping or completing a research task; or they may have an exploratory intent with no specific predefined interest. When query analysis techniques ignore such *multi-click intents*, they lose important evidence of relationships among documents and cloud their representation of users' intents.

For example, state-of-the-art measures of query ambiguity are based on measuring the entropy of clicks on web documents aggregated across users issuing a common query [74, 69, 52]. This approach, however, conflates click entropy due to multi-click intents with click entropy due to lexical and task ambiguities. For instance, the query *wedding dresses* represents a high-recall information need and, correspondingly, most users click on all of the top search results (*brides.com*, *elegant-gowns.com*, *onewed.com*). In contrast, the query *auto rent* represents an aggregation of multiple distinct navigational intents, where different users each click on one of the three URLs *rental-*

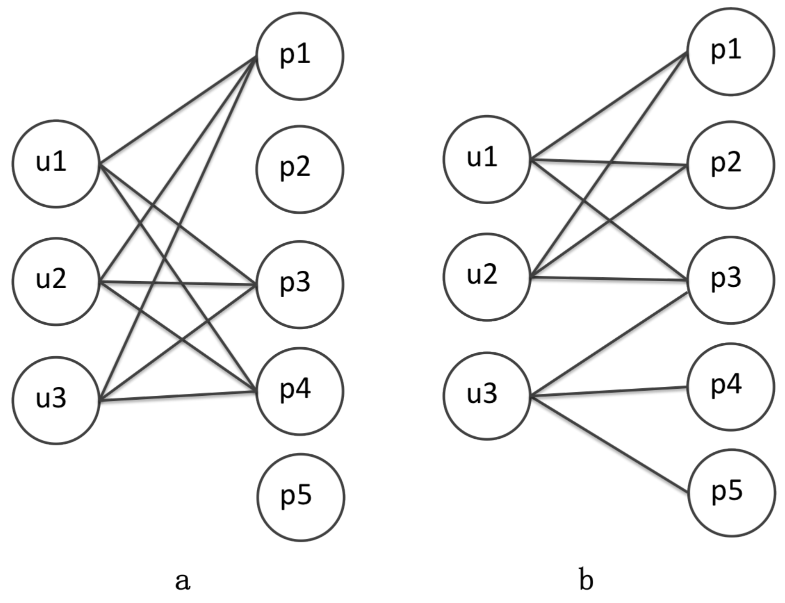


Figure 3.1: Examples of query clicks.

cars.com, *nationalcar.com* and *enterprise.com*. Despite the clear differences in query intents and user behaviors, both have similarly high click entropies.

We argue for explicitly representing multi-click intents by making *click patterns* a first-class abstraction in query analyses. Specifically, we think each individual user demonstrates a particular type of behavior when confronting a search result page. Therefore, we model the ambiguous search intents by identifying the common patterns of users' behaviors. Such patterns, namely *click patterns*, are a proxy representation of the user intents underlying a query. The underlying assumption is that although we do not know for sure the true intent of two users issuing the same query, we can be fairly sure about their similarity: when two users have similar click patterns, we believe their true intents must also be similar. One of the interesting conceptual implications is that this allows us to represent query intent as a mixture of multiple navigational and multi-click intents.

3.2 Click Patterns

Each individual user behaves in different ways when they are presented with the same search results. However, we hypothesize that underneath the noisy click behaviors each search query corresponds to an underlying behavior model, where users obey a set of common behavioral patterns.

3.2.1 Definition

Formally, we define click pattern and click profile as follows.

Definition 1: Click Pattern Given a query q and its click-through document set C_q, a click pattern τ_q is a probability distribution over the C_q representing how likely each document will be clicked on. Specifically, we use multinomial distribution to model click patterns, i.e. $\tau_q = \{p(c)|c \in C_q, \sum_{c \in C_q} p(c) = 1\}$. In practice, we simplify the representation of click pattern by only considering the top 3 most probable clicks. Therefore, each click pattern is written as a ordered list of 3 elements:

$$\tau := \{(c_1, p(c_1)), (c_2, p(c_2)), (c_3, p(c_3))\}$$

where $p(c_1) > p(c_2) > p(c_3)$.

Definition 2: Click Profile We further define Γ_q as the click profile of query q. $\Gamma_q = \{p(\tau_q)\}$ is a multinomial probability distribution over all the possible click patterns of q. The probability $p(\tau_q)$ indicates how likely a user will obey a certain click pattern τ_q when C_q presented. In later discussions, we also use Γ_q to denote the set of all possible click patterns (i.e. $p(\tau_q) > 0$) of query q where it doesn't cause confusion.

3.2.2 Modeling and Identifying Click Patterns

Ideally, we want to discover a small set of underlying click patterns that capture the common click behavior of most users. Users obeying the same click pattern are expected to show very similar click behaviors, while users obeying different click patterns shall behave in quite different manners.

Because the clicked documents are usually quite different for each query, obtaining direct supervision is difficult. Therefore, we resort to unsupervised methods. Specifically, we employ the divisive clustering algorithm to find the patterns among the noisy sample of observed click behaviors. The detail of the algorithm is shown in Algorithm 1.

The algorithm starts with a click profile of only one click pattern. Then it essentially keeps splitting the click pattern until the average intra-distance of each cluster is below a certain threshold. One merit of the algorithm is that it does not require us to preset the number of click patterns, which varies across queries. Instead, the number of click patterns is determined dynamically by the

Algorithm 1: Divisive Clustering for Discovering Click Patterns

```
input  : A set of click-vectors $V_q$ for query $q$, a similarity threshold $\sigma$, a distance function $\mathcal{F}$
output: The click profile $\Gamma_q$ for $q$
Init Cluster $c_0$ with all click vectors in $V_q$
Init Result list $l$
Enqueue $c_0$ to queue $s$
while $s$ is not empty do
    Dequeue first item $c$ from $s$
    Compute average intra-cluster distance $dist_c$ of $c$ with $\mathcal{F}$
    if $dist_c < \sigma$ then
        Add ($c.center$,$c.prob$) to $l$
    else
        Use KMeans to divide $c$ into two sub clusters $c'$ and $c''$
        Enqueue $c'$ and $c''$ to queue $s$
return $l$
```

threshold σ. This aligns with our principle of modeling click patterns as we control the similarity of click behaviors of users who obey the same click pattern.

We do not specify the distance function $\mathcal{F}$ in this algorithm. Each alternative distance function $\mathcal{F}$ might lead to an interesting exploration of click patterns. In our study, we use cosine distance (one minus cosine similarity) as the distance function. We choose this distance function mainly because it makes the setting of the distance threshold σ more intuitive.

However, it is worth mentioning that here are potentially better ways for devising the distance function, e.g. by taking into account the ranking positions. This could allow us to explore more specialized behavior models. For instance, in analyzing mobile search logs, the ranking position is of particular importance. We plan to continue exploring this idea in our future work.

3.2.3 Exploring Click Patterns in Search Logs

In this section, we explore click patterns and click profiles in real search logs [1], to study how they can help us understand user behaviors and represent complex query intents.

To facilitate our exploration, we empirically categorize click patterns into three categories as follows. These categories are intended to provide the reader with an intuitive understanding of common click patterns.

[1]For our analysis, we use the MSN search log dataset released in 2006.

- **Navigational patterns**. A navigational pattern has a single dominating URL in the click vector. It represents the user's intent of directly navigating to a particular URL.

- **Informational patterns**. An informational pattern corresponds to an information seeking intent where the purpose is to acquire knowledge on a given topic. In this case, each URL has similar chance of getting clicked. Possible scenarios for informational patterns are "comparison shopping" or "research survey" of a topic.

- **Semi-navigational patterns**. Beside navigational patterns and informational patterns, there is a third category of click pattern, where more than one URL dominate the click distribution. In this case, the query intent is still mostly navigational, but the destination of the navigation is not a single URL. We refer to this type of click patterns the semi-navigational pattern. A common scenario for semi-navigational pattern is when there is a complementary page to the major destination of the navigation. The complementary page might not be necessary for completing the user's task, but it contains useful information and improves the user's understanding of the topic.

Table 3.1 shows the click profiles of several queries discovered by the proposed algorithm. The queries were sampled from the test data set released by Wang and Agichtein [74]. For each click profile, the major click patterns and their probabilities are shown. *Query 1*, *Query 2* and *Query 3* have simple click profiles as each of them consists of only one click pattern. For *Query 1*, the intent is to navigate to the website of "*radio shack*". Consequently a single navigational pattern is observed. For *Query 2*, the intent is to survey the information of "*wedding dresses*" available on the Web. Correspondingly an informational pattern is observed. For *Query 3*, while users mostly navigate to the the URL *prom-hair.org*, some find the site *prom.hairresources.net* also helpful and explore both resources. The click pattern falls into the category of semi-navigational pattern. All three queries have clear query intents.

Query 4 is a typical ambiguous query where different users have very different targets. Its corresponding click profile shows three major navigational patterns. *Query 5* is an interesting query whose click profile is a mixture of different types of click patterns. In fact, this type of click profile is not rare in search queries. For this particular query *honda parts*, some of the users directly navigate to the online company store, while others take time to survey all the other web stores beside the

Table 3.1: Examples of click patterns

Query 1: radio shack	
τ_1 (100%): Nav.	http://radioshack.com/:0.97 http://radioshack.com/search/:0.03
Query 2: wedding dresses	
τ_1 (100%): Inf.	http://brides.com/:0.42 http://elegantgowns.com/:0.38 http://onewed.com/dresses/:0.2
Query 3: prom hair	
τ_1 (64%): Sem.	http://prom-hair.org/:0.56 http://prom.hairresources.net/:0.26 http://prom-hairstyles.us/:0.04
Query 4: rental cars	
τ_1 (50%): Nav.	http://rentalcars.com/:0.83 http://priceline.com/:0.08
τ_2 (21%): Nav.	http://nationalcar.com/:0.77 http://alamo.com/:0.2 http://enterprise.com/:0.01
τ_3 (10%): Nav.	http://enterprise.com/:0.83 http://priceline.com/:0.08 http://thrifty.com/:0.04
Query 5: honda parts	
τ_1 (70%): Inf.	http://hondapartspro.com/:0.34 http://partstrain.com/:0.2 http://hondapartstore.com/:0.15
τ_2 (24%): Nav.	http://estore.honda.com/:0.76 http://partstrain.com/:0.11 http://hondapartspro.com/:0.09

official company store. Compared with the former three queries, the query intents of *Query 4* and *Query 5* are much more complex. Such a difference is well captured by the increase of complexity in their click profiles.

Based on our analysis of 5,000 random samples of search queries from MSN search log, the majority of queries (63%) have only a single click pattern, over a third (37%) of queries have multiple click patterns, and 11% have a mixture of different kinds of click patterns, as determined by a categorization of navigational, informational and semi-navigational patterns. The distribution of click patterns shifts when we examine the most popular 5k queries. In this sample, we find significantly fewer single-intent queries (29%) as well as many more mixed-intent queries (30%).

Based on this analysis, it is clear that click pattern's richer representation of user behavior—as compared to representations that assume each unique URL represents a distinct semantic intent—is useful in capturing the observed behavior of a significant fraction of all queries, and is even more important when focusing on the most popular queries or the high-entropy queries that are the hardest to answer.

3.3 Click Patterns and Query Ambiguity

In this section, we show that our modeling of ambiguous search intent has immediate impact in query analysis. We study the measuring of ambiguity level of queries (i.e. query ambiguity). Typically, query ambiguity is measured by "click entropy", which is computed as the information entropy of the distribution of user clicks. Formally, given query q and the set of clicked documents C_q, the click entropy $H_c(q)$ is computed by Equation 3.1:

$$H_c(q) = \sum_{d \in C_q} -p(d) \log p(d) \tag{3.1}$$

where $p(d)$ is the empirical probability of document d being clicked. A higher entropy value indicates the query is more ambiguous. The concept of information entropy was originally proposed by [63] to measure the value of information in a message. Click entropy, however, does not work well in discriminating ambiguous queries, especially from information seeking queries, as both scenarios may result in very similar distribution of clicks, although the search intent of the latter is much clearer.

The fundamental assumption of using click entropy as a measure of query ambiguity is that each document represent a unique "semantic meaning" of the search query. We think this is the reason that click entropy is not able to discriminate ambiguous queries effectively. Even semantically distinct web pages may not always increase query ambiguity if they play a complementary role in fulfilling the user's intent. Recently proposed measures such as domain entropy [74] begin to consider the relationship among multiple clicked URLs. However, they do not fully separate the notion of a user's intent from the lexical and task ambiguity of a query. As a result, they still cannot represent and measure the ambiguity of complex mixtures of intents shown in Section 3.2 that exist in real search logs.

3.3.1 Pattern Entropy

We propose to model query ambiguity as an empirical notion pertaining to user behaviors, in contrast to the previous measures that emphasize on the semantic ambiguity of the query. Particularly, we compute the information entropy of the click profile of a query, i.e. the empirical distribution over the click patterns as a measurement of query ambiguity. We refer to this measurement as *pattern entropy*.

Formally, given the click profile Γ_q for query q, the pattern entropy $H_p(q)$ is computed as:

$$H_p(q) = \sum_{\tau_q} -p(\tau_q) \log p(\tau_q) \tag{3.2}$$

where $p(\tau_q)$ is the empirical probability of τ_q, given by Γ_q.

Pattern entropy is superior to click entropy as an ambiguity measure. It yields low entropy values to both navigational and informational queries where the search intents are clear, while maintaining high entropy values for queries of ambiguous intents. Take two previously discussed queries , *wedding dresses* and *auto rent*, as examples. Although both queries have similar distribution of clicks, they have distinct click profiles. For the query *wedding dresses*, most users tend to obey the same informational pattern to explore all the URLs. For the query *auto rent*, different users form three distinct navigational patterns, leading to a more complex click profile. With the entropy of click profiles, we can recognize that *auto rent* is an ambiguous query and *wedding dresses* is a clear query with an informational intent.

3.3.2 Properties of Ambiguity Metrics

In this section, we identify three important properties of metrics of query ambiguity, i.e. *discriminative power*, *consistency* and *temporal stability*. We then put the previous proposed measurements to test with a synthetic search log and a real search log, according to the three properties.

- **Discriminative power**: The most important property of an ambiguity metric is the ability to discriminate ambiguous queries from queries of clear search intents. That being said, the metric should distinguish ambiguous queries not only from navigational queries, but also from informational queries.
- **Consistency**: It is also important that an ambiguity metric generate consistent output for the same type of queries. The number of log entries for different queries usually vary a lot, even when they are of the same type (navigational, informational or ambiguous). A good query ambiguity metric should not be affected too much by this factor.
- **Temporal Stability**: Temporal stability is a property of particular practical importance. Query log is a continuous data stream and we can only process the log within a certain temporal period at a time. The analysis on the query log needs to be updated from time to time. Therefore, being able to generate temporal stable results is an important criteria for any query log analysis method.

In the following discussions, we first test the discriminative power and consistency of different ambiguity metrics with a synthetic query log. Then we continue to study the temporal stability with a real query log.

Discriminative Power and Consistency. To test the discriminative power and consistency of different ambiguity metrics and to better understand their behaviors under controlled conditions, we created a synthetic query log where different types of user behaviors were observed for different queries. The query ambiguity metrics we study are click entropy, average entropy [74] and pattern entropy. Essentially, average entropy is computed as the average of each user's click entropy.

Table 3.2 shows a set of synthetic click logs representing different level of ambiguity. Table 3.3 shows the comparison for different ambiguity metrics on the synthetic queries, i.e. click entropy, average entropy and pattern entropy.

Table 3.2: Synthetic queries with different levels of ambiguity

Query	Description
a	All users click on one same URL.
b	All users perform 10 random clicks on 10 URLs.
c	Half of the users perform 10 random clicks on 10 URLs.
d	All users perform 5 random clicks on 5 URLs.
e	Two groups of users, each of first group click on the same URL, the second group all click on a different URL.
f	Two groups of users, each of first group performs 5 random clicks on 5 URLs; second group perform 5 random clicks on the 5 different URLs.
g	Two groups of users, each of first group performs 5 random clicks on URL 1-5; second group perform 5 random clicks on the URL 3-8.
h	Three groups of users, each perform 3 random clicks on 3 different set of URLs (1-3,4-6,7-9).
i	Three groups of users, each perform 5 random clicks on 3 overlapping set of URLs (1-5,3-8,6-10).

Table 3.3: Comparison of ambiguity metrics on synthetic queries

Query	Click Entropy	Avg Entropy	Pattern Entropy
a	0.00	0.00	0.00
b	3.31	2.57	0.00
c	3.31	2.57	0.00
d	2.31	1.52	0.00
e	0.99	0.00	1.00
f	3.31	1.59	1.00
g	2.85	1.56	0.97
h	3.31	1.17	1.58
i	3.24	1.57	0.99

Query a is a clear navigational query where every user click on the same document. In this case every metrics yields the lowest value 0. Query b, c and d are clear, informational queries intended to test the consistency of the metrics. In each of the three queries, the users have a clear intention of exploring a set of URLs. Query b has in total 20 users. Query c reduces the number of participants by half to simulate the lack of data (data sparsity). Query d is different from b and c as it targets at a document set of half the size. Yet they all have a clear informational intent. We see neither click entropy nor average entropy is consistent on clear intent queries $a - d$ as they give low value to navigational queries and high value to informational queries. Pattern entropy, however, consistently generates the lowest value for all these the four queries.

Query $e - i$ are all ambiguous queries. Query e, f and g each has two groups of users with different behaviors. Query e has two different navigational patterns, while query f and g both have two informational patterns. We see that either the click entropy metric and the average entropy metric give consistent results to the three queries. They assign relative low entropies to queries with navigational patterns and high entropies to the ones with more informational patterns. In contrast, pattern entropy is consistent for the three queries. Query h and i both have three types of user behaviors. They are even more ambiguous than Query e, f and g. There is, however, no reflection of the increase in ambiguity in click entropy or average entropy. The average entropy of query h even decreases as the number of individual clicks is reduced. Query h shows an increase in pattern entropy which is in accordance with the increase in the level of ambiguity. However, our algorithm does not recognize all the three patterns in query i. This is because the second click pattern on documents 3-8 overlaps much with both the other two patterns on documents 1-5 and 6-10. As a result, instances from the second random click patterns are mistaken as from the other two patterns.

Temporal Stability. The purpose of this study is to test whether a metric generates stable results in different time periods. We first randomly sample a set of 5000 queries, denoted as *rand5k*, from the MSN query log. The MSN query log spans over an entire month from May 1st, 2006 to May 31st, 2006. Therefore, we extract the log entries of the queries in *rand5k* and split them into two buckets (May 1st to May 15th and May 16th to May 31st). We then compute the ambiguity metrics on the two buckets of logs and draw the scatter plots of click entropy, average entropy and pattern entropy in Figure 3.2, Figure 3.3 and 3.4, respectively. In addition, we compute the correlation between the values of each metric calculated with different time period's data:

$$cor(X,Y) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{3.3}$$

We can see that both click entropy and pattern entropy show strong correlation in the two halves of months' data, with *cor* of 0.81 and 0.68, respectively. Therefore, they tend to generate stable results across different time periods. Average entropy, on the other hand, does not show strong visual correlation and has a low *cor* value of 0.54. This indicates that unlike the other two metrics, the computation of average entropy is more volatile in terms of time. This makes it less dependable for query log analysis and application purposes.

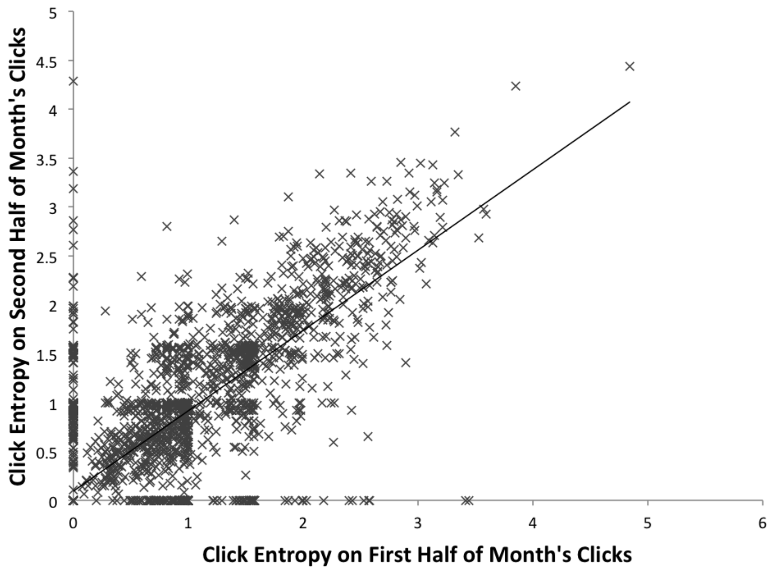


Figure 3.2: Temporal stability of click entropy on 5000 random queries, $cor = 0.81$.

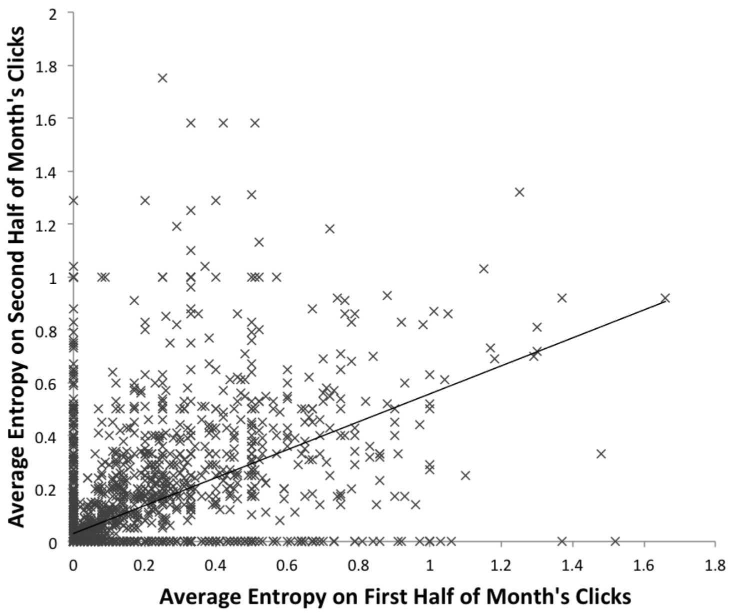


Figure 3.3: Temporal stability of average entropy on 5000 random queries, $cor = 0.54$.

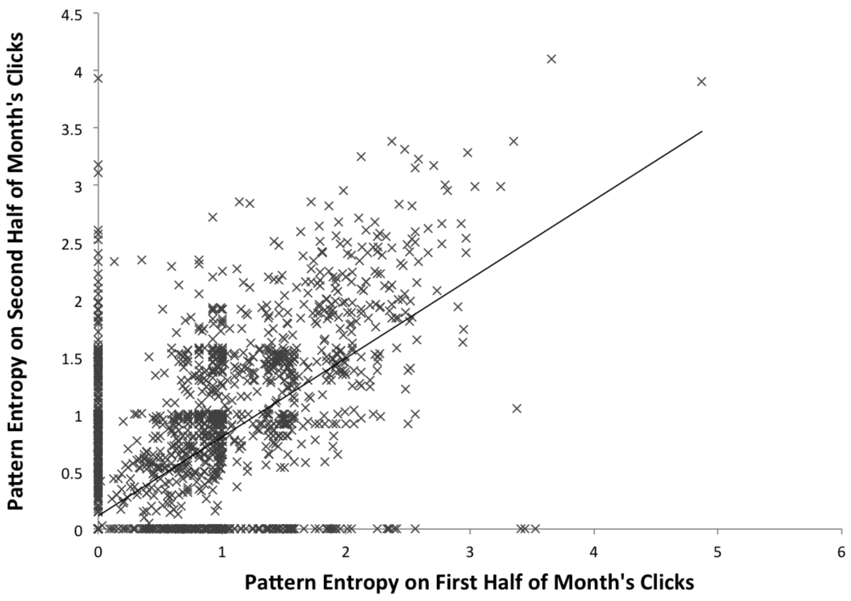


Figure 3.4: Temporal stability of pattern entropy on 5000 random queries, $cor = 0.68$.

It is interesting to see that in all three figures, many data points falls onto the axis. This phenomenon is mostly caused by the volatility of users and queries on search engines. Such queries are usually related to temporal events, such as *what is May day* (on the event "May day" which takes place in May every year) or *Typhoon Chanchu* (on the event "Typhoon Chanchu" in 2006).

3.4 Classifying Ambiguous Queries

To further verify the effectiveness of the proposed ambiguity metric, we use a human labeled query set to experiment with automatic classification of ambiguous queries.

3.4.1 Problem Setup

We use the same dataset as used in Wang and Agichtein's work [74]. There are in total 150 queries in the dataset, labeled as either "navigational", "informational" or "ambiguous" by human annotators. The Kappa value was 0.77. The query log is extracted from the MSN search query log released in 2006. We target at identifying ambiguous queries from the query log. Therefore, instead of performing three class classification, we merge together the "clear" and "informational" queries in the query log as all clear queries. Then we perform binary classification to separate ambiguous and clear intent ("clear" and "informational") queries. We use logistic regression as our classification method. All results are generated by 10-fold cross validation. The feature sets we use are listed in Table 3.4. The domain entropy features are computed by replacing the URLs with corresponding domains, where domains are obtained by truncating the URLs of the clicked web pages to their top level domain (truncating before the first "/" in the URL).

Table 3.4: Features used for classification

Feature set	Description of features
clk-ent	length of query frequency of query click entropy domain click entropy
avg-ent	average entropy domain average entropy
pat-ent	pattern entropy domain pattern entropy

3.4.2 Classification Results

Table 3.5 shows classification results. We can see that both average entropy and pattern entropy features improve the overall accuracy of classification. Both sets of features improve the ambiguous queries more than the clear queries. We also observe that pattern entropy features perform better than average entropy features. This confirms that pattern entropy is a superior ambiguity metric. However, we do not see further improvement in combining the average entropy features and pattern entropy features. This is probably because the additional information delivered by the two set of features overlapped with each other.

Table 3.5: Classification results

	Overall	Clear		Ambiguous	
	Acc	Prec	Rec	Prec	Rec
clk-ent	0.77	0.80	0.95	0.14	0.07
w/ avg-ent	0.78	0.82	0.92	0.37	0.20
w/ pat-ent	**0.81**	**0.83**	**0.96**	**0.42**	**0.21**
w/ both	0.79	**0.83**	0.94	0.37	0.19

3.4.3 Case Study

In Table 3.6 we show several examples of human annotated queries with different entropy values to help understand how pattern entropy helps improving the classification of ambiguous queries.

Most navigational queries tend to have low entropy values for any ambiguity metrics. Therefore they are easy to handle for any ambiguity metric. We see that for navigational queries, their patterns are solely navigational patterns.

Both the two queries *online auction* and *white pages* are labeled as informational queries. The average entropy for the former is relatively high. But the latter does not get the same high value. By identifying click patterns, we find that *online auction* has a single semi-navigational pattern. It therefore has a zero pattern entropy. The query *white pages*, however, has a mixture of both navigational and informational patterns, and the majority of the users are with different navigational patterns. This leads to the relatively low average entropy and pattern entropy.

The queries *song lyrics* and *ares* are labeled as ambiguous queries. The query *song lyrics* has a mixture of navigational, informational and semi-informational patterns. A possible explanation is depending on the “lyric” the user is looking for, he/she may have to try different number of websites to get it. As a result, the query has a relative high average entropy and pattern entropy. For the query *ares*, multiple navigational and semi-navigational click patterns are discovered. Since the individual intents are mostly navigational, it has a low average entropy. However, the pattern entropy is high because different patterns have comparable amount of users.

Overall, we see click pattern is quite consistent in separating the ambiguous queries from the clear queries (both navigational and informational). However, the average entropy does not work well in some cases when mixtures of click patterns exist for one query.

3.5 Click Patterns and Query Recommendations

In this section, we demonstrate the impact of our model for ambiguous search intent in the application of query recommendation. The purpose of query recommendation is to help users explore the query space so that they can quickly find the query that best describe their search intent. To serve this purpose, we want to start from the close neighborhood of the given query and suggest queries that share major intents with the given query. Baeza-Yates et al. proposed to use query log

Table 3.6: Examples of entropies and patterns

Query	Avg-Ent	Pat-Ent	Patterns
Clear Navigational Queries			
chase	0.03	0.00	Nav.
ca lottery	0.01	0.13	Nav.
Clear Informational Queries			
online auction	0.81	0.00	Sem.
white pages	0.13	0.78	Nav+Inf.
Ambiguous Queries			
song lyrics	0.64	3.09	Nav.+Inf.+Sem.
ares	0.28	1.56	Nav.+Sem.

for query recommendation based on the notions of query similarity and support [3]. It was suggested later that the reduction of query ambiguity of the query should also be accounted for in query recommendation [78, 7].

We leverage our model for ambiguous search intent to improve the computation of query similarity and ambiguity reduction. The reduction in ambiguity is naturally measured by pattern entropy. We compute the pattern similarity between two queries q and q' as:

$$S_p(q, q') = \sum_{\tau \in \Gamma_q} \sum_{\gamma \in \Gamma_{q'}} p(\tau) \cdot p(\gamma) \cdot \cos(\tau, \gamma) \tag{3.4}$$

where Γ_q is the set of click patterns and w_τ is the weight of pattern τ.

3.5.1 Problem Setup

We devise query recommendation as a classification task by restricting the search space to queries that add only one term to the original query. We randomly select 200 queries that have frequency above 10 from MSN search query log release in 2006 and generate the candidate recommendations in this way. The queries with no candidate suggestions are removed from the dataset, and we finally have 127 queries. Instead of performing human annotation, we adopt an automatic annotating process. Specifically, we count how many times a candidate suggestion appear after the original query in a session with a different query log. We use the AOL search query log released in 2006 for this purpose. The reason to use a secondary query log is to avoid overfitting a single query log. As the AOL query log is released the same year as the MSN query log, the changes in queries would not be dramatic. We then label the candidate as recommendation or not by checking if the count is above 10 times. In total we have 848 queries labeled as recommendations out of 3416 candidates. We then perform binary classification for each candidate query. Note that this experiment is intended to demonstrate the usage of click pattern and pattern entropy in a real world application, rather than compete with current state-of-the-art techniques for query recommendation.

3.5.2 Query Recommendation Results

Table 3.7 shows the classification result for query recommendation. We see by adding pattern entropy and pattern similarity features, we can incrementally improve the performance of classification. The best performance by using both pattern entropy and pattern similarity with query popularity.

Table 3.7: Results of Query Recommendation as a Classification Task

	Overall	Recommended	
	Acc	Prec	Rec
popularity	0.75	0.67	0.54
w/ pattern entropy	0.76	0.68	0.56
w/ pattern similarity and pattern entropy	**0.78**	**0.70**	**0.59**

3.5.3 Case Study

In Table 3.8 we show the classification result for candidate query suggestions for "baby names". We can see that the recommended queries generally have high frequency, low pattern entropy and high pattern similarity with the original query.

Once again, this application confirms the effectiveness of pattern entropy as an ambiguity metric. Beside query recommendation, click pattern is also potentially useful in many other applications, e.g. personalized search and diversification of search results. In the future, we plan to further study the applications of click pattern and pattern entropy.

Table 3.8: Query recommendation for "baby names" ($H_p(q)$=2.76)

Query	Rec	Freq.	Pat-Ent	Pat-Sim
unique baby names	Y	31	1.44	0.01
popular baby names	Y	30	1.35	0.24
girl baby names	Y	22	1.33	0.01
unusual baby names	Y	18	1.38	0.49
top baby names	Y	12	0.99	0.08
irish baby names	N	19	2.38	0.00
celebrity baby names	N	15	2.61	0.00
spanish baby names	N	14	0.99	0.00
biblical baby names	N	12	1.94	0.00
		......		

3.6 Conclusions

In this work, we propose and study the use of *click patterns* as an empirical representation of user intent for queries with ambiguous search intents. We show how click patterns can be extracted from logs of user behavior and demonstrate that click patterns provide a richer representation of query intents, from multi-click intents such as high-recall research tasks to navigational intents, and mixtures of query intents of various kinds. We examine real query logs and find that the richer representation of query intents afforded by click patterns is critical for capturing the user behavior for a significant fraction of queries, and especially for popular and high entropy queries. We further demonstrate the integration of click patterns into existing query analyses by adapting traditional query ambiguity and query recommendation tasks to use click patterns as the fundamental unit of user behavior.

Part II

Supporting Query Formulation

CHAPTER 4

QUERY COMPLETION AND CORRECTION

4.1 Introduction

One major obstacle in accurate search intent understanding is the vocabulary gap between users' queries and the intended information. Lacking knowledge of the relevant information, users are usually unable to accurately describe their search intent in queries, making it challenging to infer the real intent. In this chapter, we propose and study several approaches that alleviate the difficulty of query formulation.

One strategy to alleviate the problem of vocabulary gap is to provide real time guidance in query formulation through query completion and correction. By do so we can effectively guide the users to choose the right terminologies to represent their search intent. In this work we study the problem of real time query completion and correction by modeling search intents from search engine query logs.

Particularly, we propose and study a generative model, where we assume the original search intent is transformed through a noisy channel into a potentially misspelled query. By estimating the intent popularity through click-through logs, we can capture the important search intent and provide guidance to the users so that they can compose search queries that can effectively convey the intent. With an accurate model for the noisy channel (i.e. the transformation between the target query and the user entered query), we can make sure that we will guide the user toward the desired search intent instead of biasing toward the common popular intent.

4.2 Problem Formulation

Most query reformulation models adopt a common generative framework:

$$\hat{c} = \operatorname*{argmax}_{c} p(c|q) = \operatorname*{argmax}_{c} p(q|c)p(c) \tag{4.1}$$

where q is the original query and c is a candidate suggestion. In this noisy channel model formulation, $p(c)$ is a query language model that describes the prior probability of c as the intended user query. $p(q|c) = p(c \rightarrow q)$ is the transformation model that represents the probability of observing the query q when the original user intent is to enter the query c.

In the setting of query completion and correction, we are given only the prefix $\bar{q}$ of the potentially misspelled input query q. Thus, the objective is to find the correctly spelled query $\hat{c}$ that maximizes the probability of yielding any query q that extends the given partial query $\bar{q}$. More formally, we want to find:

$$\hat{c} = \operatorname*{argmax}_{c,q:q=\bar{q}\cdots} p(c|q) = \operatorname*{argmax}_{c,q:q=\bar{q}\cdots} p(q|c)p(c) \tag{4.2}$$

where $q = \bar{q}\cdots$ denotes that $\bar{q}$ is a prefix of q. Modeling the query language model is usually trivial. In this work, we use the popularity model as an empirical distribution of query language model. The challenge is to achieve an accurate transformation model so that automatically complete and correct users' query based on the query intent.

As we can see from Equation 4.2, by relaxing q to be any query that extends the partial query $\bar{q}$, online spelling correction and completion significantly increases the theoretical search space. However, with appropriate data structures and algorithms, this search can be done efficiently.

4.3 Transformation Model

In this work, we propose to study the joint sequence model for query correction and completion. Joint sequence model was originally proposed for grapheme to phoneme conversion in speech recognition [9].

Specifically, we segment the conversion from c to q as a sequence of substring transformation units, or *transfemes*. For example, the transformation $schwarzenegger \rightarrow shwartzeneger$ can be segmented into the transfeme sequence: $\{sch \rightarrow sh$, $war \rightarrow war$, $ze \rightarrow tze$, $negg \rightarrow neg$, $er \rightarrow er\}$. As another example, the transformation $britney \rightarrow britny$ can be segmented into the transfeme sequence $\{br \rightarrow br, i \rightarrow i, t \rightarrow t, ney \rightarrow ny\}$, where only the last transfeme, $ney \rightarrow ny$, involves a correction. By describing the sequence with a transfeme n-gram language model, we can decompose the transformation model into a set of conditional transfeme probabilities. This allows us to not only train the model from segmented correction pairs, but also generalize the model to previously unseen transformations.

Given a sequence of transfemes $s = t_1 t_2, ..., t_{l^s}$, we can expand the probability of the sequence using the chain rule. As there are multiple ways to segment a transformation in general, we further model the transformation probability $p(c \rightarrow q)$ as the sum of all possible segmentations. Formally,

$$p(c \rightarrow q) = \sum_{s \in S(c \rightarrow q)} p(s) = \sum_{s \in S(c \rightarrow q)} \prod_{i \in [1, l^s]} p(t_i | t_1, ..., t_{i-1}) \tag{4.3}$$

where $S(c \rightarrow q)$ is the set of all possible joint segmentations of c and q. Further applying the Markov assumption that a transfeme only depends on the previous $M - 1$ transfemes, similar to an n-gram language model, we obtain:

$$p(c \rightarrow q) = \sum_{s \in S(c \rightarrow q)} \prod_{i \in [1, l^s]} p(t_i | t_{i-M+1}, ..., t_{i-1}) \tag{4.4}$$

We define the length of a transfeme $t = c_t \rightarrow q_t$, as:

$$|t| = max\{|c_t|, |q_t|\} \tag{4.5}$$

In general, a transfeme can be arbitrarily long. To constrain the complexity of the transformation model, we limit the maximum length of a transfeme to L. With both n-gram approximation and transfeme length constraint, we obtain the final model with parameters M and L:

$$p(c \rightarrow q) = \sum_{s \in S(c \rightarrow q): \forall t \in s, |t| \leq L} \prod_{i \in [1, l^s]} p(t_i | t_{i-M+1}, ..., t_{i-1}) \tag{4.6}$$

In the special case of $M = 1$ and $L = 1$, the transformation model degenerates to a model similar to weighted edit distance. With $M = 1$, we assume that the transfemes are generated independently of one another. As each transfeme contains substrings of at most one letter, we can model the standard Levenshtein edit operations [48]: insertions $\epsilon \rightarrow \alpha$, deletions $\alpha \rightarrow \epsilon$, and substitutions

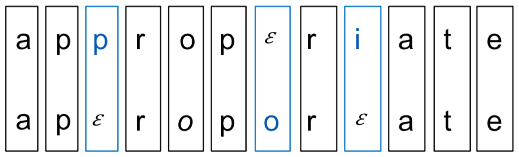


Figure 4.1: Example transformation with $L = 1$

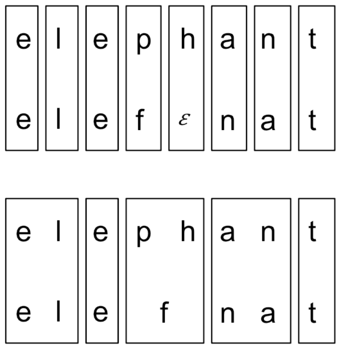


Figure 4.2: Comparing transformations with $L = 1$ and $L = 2$

$\alpha \to \beta$, where ϵ denotes the empty string. However, unlike many edit distance models, the weights in the transformation model represent normalized probabilities estimated from data, not just arbitrary score penalties. Thus, the transformation model not only captures the underlying patterns of spelling errors, but also allows us to compare the probabilities of different completion suggestions in a mathematically principled way. Figure 4.1 contains an example of such a transformation.

With $L = 1$, transpositions are penalized twice, even though it occurs as easily as other edit operations. Similarly, phonetic spelling errors, such as $ph \to f$, often involve multiple characters. Modeling these transfemes as single character edit operations not only over-penalizes the transformation, but also pollutes the model as it increases the probabilities of edit operations, such as $p \to f$, that would otherwise have very low probabilities. By increasing L, we increase the allowable length of the transfemes. Thus, the model is able to capture more meaningful trans-formation units and reduce probability contamination that result from decomposing intuitively atomic substring transformations. Figure 4.2 compares an example transformation with $L = 1$ and $L = 2$.

Instead of increasing L, we can also improve the modeling of errors spanning multiple characters by increasing M, the number of transfemes the model probabilities are conditioned on. Consider the example from Figure 4.2 with $L = 1$. When $M = 1$, no context is considered in the generation of each transfeme. When $M = 2$, the probability of each transfeme is dependent on its previous transfeme. As a result, we are able to capture the fact that $h \rightarrow \epsilon$ has a much higher probability when following the transfeme $p \rightarrow f$. As a more interesting example, ie is often misspelled as ei. A unigram model ($M = 1$) is not able to express such an error. A bigram model ($M = 2$) captures this pattern by assigning higher probability to the transfeme $e \rightarrow i$ when following $i \rightarrow e$. A trigram model ($M = 3$) can further identify exceptions to this pattern when preceded by a c, as cei is more common than cie.

4.3.1 Model Estimation

To learn the patterns of user spelling errors, we use a parallel corpus of input and output query pairs, where the input represents the intended query with correct spelling and the output corresponds to the potentially misspelled transformation of the input. If such data is pre-segmented into transfemes, we can derive the transformation model directly using maximum likelihood estimation (MLE). However, such labeled training data is generally too costly to obtain in large scale. Thus, we devise an expectation-maximization (EM) algorithm to estimate the parameters in the transformation model from partially observed data.

Given a set of observed training pairs $O = \{O^k\}$, where $O^k = c^k \rightarrow q^k$, we can write the log likelihood of the training data as:

$$\log \mathcal{L}(\Theta; O) = \sum_k \log p(c^k \rightarrow q^k | \Theta) = \sum_k \log \sum_{s^k \in S(O^k)} p(s^k | \Theta) \tag{4.7}$$

where $\Theta = \{p(t|t_{-M+1}, ..., t_{-1})\}$ is the set of model parameters. $s^k = t_1^k t_2^k, ..., t_{l^s}^k$, the joint segmentation of each training pair $c^k \rightarrow q^k$ into a sequence of transfemes, is the unobserved variable. By applying the EM algorithm [8], we can iteratively find the parameter set Θ that maximizes the log likelihood.

For $M = 1$ and $L = 1$, where each transfeme of length up to 1 is generated independently, we derive the following update formulas:

$$p(s; \Theta) = \prod_{i \in [1, l^s]} p(t_i; \Theta) \tag{4.8}$$

$$e(t;\Theta) = \sum_k \sum_{s^k \in S(O^k)} \frac{p(s^k;\Theta)}{\sum s' \in S(O^k) p(s';\Theta)} \#(t, s^k) \tag{4.9}$$

$$p(t;\Theta') = \frac{e(t;\Theta)}{\sum_{t'} e(t';\Theta)} \tag{4.10}$$

where $\#(t, s)$ is the count of transfeme t in the segmentation sequence s, $e(t;\Theta)$ is the expected partial count of the transfeme t with respect to the transformation model Θ, and Θ' is the updated model. $e(t;\Theta)$, also known as the evidence for t, can be computed efficiently using a forward-backward algorithm [9].

We can extend the EM training algorithm to higher order transformation models ($M > 1$), where the probability of each transfeme now depends on the previous $M-1$ transfemes. Other than having to take into account the transfeme history context when accumulating the partial counts, the general EM procedure is essentially the same. Specifically, we have:

$$p(s;\Theta) = \prod_{i \in [1, l^s]} p(t_i | t_{i-M+1}^{i-1}; \Theta) \tag{4.11}$$

$$e(t, h;\Theta) = \sum_k \sum_{s^k \in S(O^k)} \frac{p(s^k;\Theta)}{\sum s' \in S(O^k) p(s';\Theta)} \#(t, h, s^k) \tag{4.12}$$

$$p(t|h;\Theta') = \frac{e(t, h;\Theta)}{\sum_{t'} e(t', h;\Theta)} \tag{4.13}$$

where h is a transfeme sequence representing the history context, and $\#(t, h, s)$ is the occurrence count of transfeme t following the context h in the segmentation sequence s. Though more complicated, $e(t, h;\Theta)$, the evidence for t in the context of h, can still be computed efficiently using the forward-backward algorithm.

As the number of model parameters increases with M, we initialize the model parameters using the converged values from the lower order model to achieve faster convergence. Specifically,

$$p(t|h^M;\Theta^M) = p(t|h^{M-1};\Theta^{M-1}) \tag{4.14}$$

where h^M is a sequence of $M-1$ transfemes representing the context, and h^{M-1} is h^M without the oldest context transfeme.

Extending the training procedure to $L > 1$ further complicates the forward-backward computation. But the general form of the EM algorithm remains the same.

4.3.2 Model Pruning

One challenge with a direct implementation of the above algorithms is that as we increase the model parameters M and L, the number of potential parameters in the transformation model increases exponentially. Assuming an alphabet size of 50, a $M = 1$,$L = 1$ model contains $(50+1)^2$ parameters, as each component in $t = c_t \to q_t$ can take on any of the 50 symbols or ϵ. But a $M = 3$,$L = 2$ model may contain up to $(50^2 + 50 + 1)^{2.3} \approx 2.8 \times 10^{20}$ parameters! Although most parameters are never observed in the data, model pruning techniques are still beneficial to reduce the overall search space, during both training and decoding, and to reduce overfitting, as infrequent transfeme n-grams are likely to be noise.

In this work, we employ two pruning strategies in each iteration of the training algorithm. First, we remove transfeme n-grams with expected partial counts below a threshold τ^e. Second, we trim out transfeme n-grams with estimated conditional probabilities below a threshold τ^p. The thresholds τ^e and τ^p are tuned against a held-out development set. By filtering out transfemes with low confidence, we significantly reduce the number of active parameters in the model and speed up the running time of training and decoding.

4.3.3 Model Smoothing

As with any maximum likelihood estimation techniques, the EM algorithm has a tendency to overfit the training data when the number of model parameters is large, for example when $M > 1$. The standard technique in n-gram language modeling to address this problem is to apply smoothing when computing the conditional probabilities. In our work, we study two smoothing techniques: Jelinek-Mercer (JM) and absolute discounting (AD).

In JM smoothing, the probability of a transfeme is given by the linear interpolation of its maximum likelihood estimation at order M (using partial counts) and its smoothed probability from a lower order distribution:

$$p^{JM}(t|h^M) = (1-\alpha)\frac{e(t,h^M)}{\sum_{t'} e(t',h^M)} + \alpha p^{JM}(t|h^{M-1}) \tag{4.15}$$

where $\alpha \in (0,1)$ is the linear interpolation parameter. Note that $p^{JM}(t|h^M)$ and $p^{JM}(t|h^{M-1})$ are probabilities from different distributions within the same model. That is, in computing the M-gram model, we also compute the partial counts and probabilities for all lower-order m-grams, where $m \leq M$.

AD smoothing operates by discounting the partial counts of the transfemes. The removed probability mass is then redistributed to the lower order model:

$$p^{AD}(t|h^M) = \frac{max(e(t,h^M)-d,0)}{\sum_{t'} e(t',h^M)} + \alpha(h^M)p^{AD}(t|h^{M-1}) \tag{4.16}$$

where d is the discount and $\alpha(h^M)$ is computed such that $\sum_t p^{AD}(t|h^M) = 1$. Note that since the partial count $e(t,h^M)$ can be arbitrarily small, it is not possible to choose a value of d such that $e(t,h^M)$ will always be larger than d. Consequently, we will trim the model if $e(t,h^M) \leq d$. For both smoothing techniques, all parameters are tuned on a held-out development set.

4.3.4 Mixture Models

When training from a dataset consisting of only query correction pairs, the resulting model is likely to over-correct. To address this issue, we prepare another dataset of correctly spelled query pairs and propose two ways of using the two datasets for training.

The first approach simply concatenates the two datasets together when estimating the transformation model. We refer to this method as *data mixture*. The second technique trains two transformation models from the two datasets individually. It is easy to see that the model trained from correctly spelled queries will only assign non-zero probabilities to transfemes with identical input and output, as all the transformation pairs are identical. We linearly interpolate the two models as the final model:

$$p(t) = (1 - \lambda)p(t; \Theta^{misspelled}) + \lambda p(t; \Theta^{identical}) \tag{4.17}$$

We label this approach as model mixture, where we can view each transfeme as probabilistically generated from one of the two distributions, according to the interpolation factor λ. As with all other modeling parameters, λ is tuned on a held-out development set.

4.3.5 Discussions

Observant readers may have noticed that the transformation model estimates the joint probability of the input and output substrings in a transfeme. As the transformation probability is later multiplied with the query language model in the generative formulation for online and offline spelling corrections, we are essentially double counting the input query probability. A solution to this problem is to normalize the transformation model for each input substring after training, so as to obtain a conditional model. Although this solution is theoretically sound, initial experiments have failed to improve the performance. As is common in speech recognition, where a "fudge factor" is introduced to balance the language model score against the acoustic model, we reformulate the optimization as:

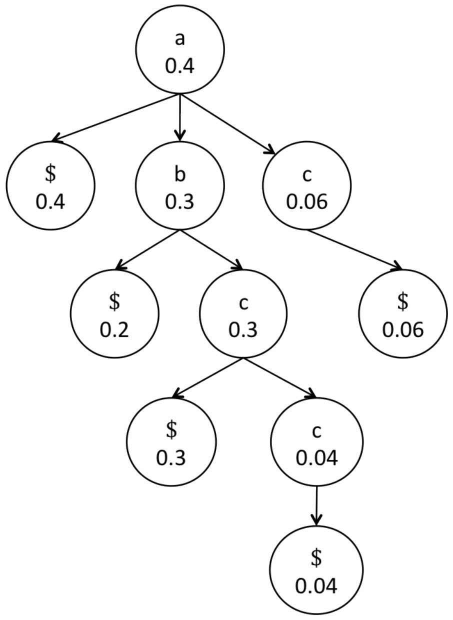
$$\hat{c} = \underset{c}{\operatorname{argmax}}\, p(q|c)p(c) \approx \underset{c}{\operatorname{argmax}}\, p(c \to q)p(c)^{\gamma} \tag{4.18}$$

where $p(c \to q)$ is still the transformation model probability, and γ is the fudge factor controlling the additional probability mass of the query language model. Empirically, this approach turns out to be very effective in our experiments, although it lacks a theoretical foundation. We plan to continue exploring this issue in future work.

4.4 Search

With a query language model and a transformation model, we are able to compute the probability of any query q given an input query c. However, our task is to find the input query $\hat{c}$ with highest probability efficiently, so as to enable offline spelling correction. More generally for online spelling correction, we want to find the top k completions of an observed query prefix $\bar{q}$. To achieve this,

Query	Count
a	20
ab	10
ac	3
abc	15
abcc	2



a b

Figure 4.3: Trie with highest probabilities

we propose to apply the A* search algorithm against a trie representing the query language model. Below we first introduce the modified trie data structure that we use to store the queries and their probabilities. We then present the A* search algorithm, followed by discussions on the pruning and thresholding techniques necessary to improve the efficiency and quality of the suggestions.

4.4.1 Trie

As the search algorithm starts from the beginning of a query and incrementally traverses potential corrections one letter at a time, we use a prefix tree (trie) to represent all queries in the query log. Figure 4.3b shows a trie built over the set of strings in Figure 4.3a. To avoid ambiguity, we end each string with an implicit \$ character. Thus in the trie, all leaf nodes are associated with a complete query. Internal nodes do not represent complete strings. For each node in the trie, we store the largest probability among all queries represented by its descendant leaf nodes. As this represents the largest value among all queries starting with the prefix associated with the node, we can apply it an admissible heuristic function for A* search.

```
Input:  Query trie $T$, transformation model $\Theta$, integer $k$, query prefix $\bar{q}$
Output: Top $k$ completion suggestions of $\bar{q}$
A   List $l$ = new List()
B   PriorityQueue $pq$ = new PriorityQueue()
C   $pq$.Enqueue(new Path(0, $T$.Root, [], 1))
D   while (!$pq$.Empty())
E     Path $\pi$ = $pq$.Dequeue()
F     if ($\pi$.Pos < $|\bar{q}|$)                    // Transform input query
G       foreach (Transfeme $t$ in GetTransformations($\pi$, $\bar{q}$, $T$, $\Theta$))
H         int $i$      = $\pi$.Pos + $t$.Output.Length
I         Node $n$     = $\pi$.Node.FindDescendant($t$.Input)
J         History $h$ = $\pi$.Hist + t
K         Prob $p$     = $\pi$.Prob × ($n$.Prob / $\pi$.Node.Prob) ×
                       $P(t, \pi.\text{Hist}; \Theta)$
L         $pq$.Enqueue(new Path($i$, $n$, $h$, $p$))
M     else                                          // Extend input query
N       if ($\pi$.Node.IsLeaf())
O         $l$.Add($\pi$.Node.Query)
P         if ($l$.Count ≥ $k$)
Q           return $l$
R       else
S         foreach (Transfeme $t$ in GetExtensions($\pi$, $T$, $\Theta$))
T           int $i$      = $\pi$.Pos + $t$.Output.Length
U           Node $n$     = $\pi$.Node.FindDescendant($t$.Input)
V           History $h$ = $\pi$.Hist + t
W           Prob $p$     = $\pi$.Prob × ($n$.Prob / $\pi$.Node.Prob)
X           $pq$.Enqueue(new Path($i$, $n$, $h$, $p$))
Y   return $l$
```

Figure 4.4: A* search algorithm for online spelling correction

4.4.2 A* Search

We use the A* search algorithm to find the top k corrected query completions for the prefix $\bar{q}$, given the query trie T and transformation model Θ. We represent each intermediate search path as a quadruplet $< Pos, Node, Hist, Prob >$, corresponding to the current position in the query prefix $\bar{q}$, the current node in trie T, the transformation history so far, and the probability of this search path, respectively. The full algorithm is presented Figure 4.4.

The algorithm works by maintaining a priority queue of intermediate search paths, ranked by decreasing probabilities. We initialize the queue with the initial path $< 0, T.Root, [], 1 >$ (line C). While there is still a path on the queue, we dequeue it and check if there are still characters unaccounted for in the input prefix $\bar{q}$ (lines F). If so, we iterate over all transfeme expansions that transform substrings starting from the current node in the trie to substrings yet unaccounted for in the query prefix (line G). For each transfeme expansion, we add a corresponding path to the trie (line L). The probability of the path is updated to include adjustments to the heuristic future score and the probability of the transfeme given the previous history (line K).

As we expand the search path, we will eventually reach a point where all the characters in the input query have been consumed. The first path in the search that meets this criterion represents a partial correction to the partial input query $\bar{q}$. At this point, the search transitions from correcting potential errors in the partial input to extending the partial correction to complete queries. In this scenario (line M), if the path is associated with a leaf node in the trie (line N), indicating that we have reached the end of a complete query, we add the corresponding query to the suggestion list (line O) and return if we have sufficient suggestions (line P). Otherwise, we iterate over all transfemes that extend from the current node (line S) and add them to the priority queue (line X). As the transformation score is not affected by extensions to the partial query, we only update the score to reflect the changes in the heuristic future score (line W). When we run out of search paths to expand, we return the current list of correction completions (line Y).

The heuristic future score we use in the A* algorithm, as applied in line K and W, is the probability value stored with each node in the trie. As this value represents the largest probability among all queries reachable from this path, it is an admissible heuristic that guarantees that the algorithm will indeed find the top suggestions.

One problem with this heuristic function is that it does not penalize the untransformed part of the input query. Therefore, we can design a better heuristic by taking into consideration the upper bound of the transformation probability $p(c \to q)$. Formally,

$$heuristic^*(\pi) = \max_{c \in \pi.Node.Queries} p(c) \times \max_{c'} p(c' \to q_{[\pi.Pos,|q|]} | \pi.Hist; \Theta) \tag{4.19}$$

where $q_{[\pi.Pos,|q|]}$ is the substring of q from position $\pi.Pos$ to $|q|$. For each query, we pre-compute the second maximization in the equation for all positions of q using dynamic programming.

The A* search algorithm can also be configured to perform exact match for offline spelling correction by simply substituting the probabilities in line W with line K. In effect, we continue to penalize transformations involving additional unmatched letters even after finding a prefix match.

It is worth noting that a search path can theoretically grow to infinite length, as ϵ is allowed to appear as either the source or target of a transfeme. In practice, this does not happen as the probability of such transformation sequences will be very low and will not be further expanded in the search algorithm.

A translation model with larger L parameter (L bounds the length of transfemes) significantly increases the number of potential search paths. As we need to consider all possible transfemes with length less or equal to L when expanding each path, models with larger L are less efficient.

4.4.3 Pruning

To further improve the efficiency of A* search, we need to limit the search space and prune unpromising paths early. In practice, carefully designed beam pruning methods can usually achieve significant improvement in efficiency without causing much loss in accuracy. In our work, we employ two pruning techniques: absolute pruning and relative pruning.

For absolute pruning, we limit the number of paths to be explored at each position in the target query q. As mentioned earlier, the complexity of our search algorithm is theoretically unbounded due to ϵ transfemes. However, by applying absolute pruning, we can bound the complexity of the algorithm by $O(|q|LK)$, where K is the number of paths allowed at each position in q.

With relative pruning, we only explore the paths that have probabilities higher than a certain percentage of the maximum probability at each position. The threshold values are carefully designed to achieve the best efficiency without causing a significant drop in accuracy. In practice we find relative pruning to be generally more effective for pruning unpromising paths. In our system, we make use of both absolute pruning and relative pruning to improve search efficiency and accuracy.

4.4.4 Thresholding

From the perspective of user interface, it is not always a good idea to show a predefined number of suggestions for every query. Showing more suggestions incurs a cost, as users spend more time looking at them instead of completing their task. Moreover, showing irrelevant suggestions risks annoying users. Therefore, we need to make a binary decision for each suggestion on whether it should be shown to the user. Ideally, we want to measure the distance between the target query q and the suggested correction c. The larger the distance, the more risk we take to include it in the suggestions. One way to approximate the distance is to compute the log of the inverse transformation probability, averaged over the number of characters in the query:

$$risk(c, q) = \frac{1}{|q|} \log \frac{1}{p(c \to q)} \tag{4.20}$$

This risk function is not very effective in practice, as the input query q usually consists of several words, of which only one is misspelled. It is unintuitive to average the risk over all letters in the query. Instead, we can first segment q into words and measure the risk at the word level. Specifically, we measure the risk of each word separately using the above formula and define the final risk function as the fraction of words in q having a risk value above a given threshold.

4.5 Experiment

4.5.1 Datasets

Our primary focus in this work is to build a transformation model that is able to capture all the misspelling behaviors of users. To obtain such behaviors, we make use of the click logs of search engine recourse links. Recourse links are provided when the offline correction mechanisms of search engines detect a potential misspelling. For example, in Google (Figure 4.5a), a recourse link is shown in the sentence "Did you mean: *important*". When the user clicks on this link, it indicates that the user agrees with the correction. Therefore, the search engine will use the suggested query to rerun the search. Similarly recourse links are provided in Bing as well (Figure 4.5b). By recording such clicks, we accumulate a set of high quality corrections that represent real user spelling behaviors.

It is worth noting that although the recourse links are provided by an offline spelling correction system, it does not mean that our ability will be limited to that of the offline system. First, our model captures the underlying patterns of spelling corrections instead of memorizing corrections at the word or query level. For instance, in the example from Figure 4.5, a possible pattern is that *im* tends to be misspelled as *in*. Second, our logs consist of recourse link clicks from multiple sites.

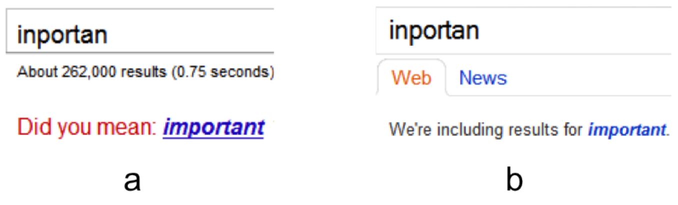

inportan
About 262,000 results (0.75 seconds)
Did you mean: important
a
inportan
Web
News
We're including results for important.
b

Figure 4.5: Examples of recourse links

As the spellers of different search engines behave differently, we can learn from a diverse set of correction pairs.

There are also other ways to obtain records of spelling corrections. For example, by analyzing the webpage metadata for near-miss spellings, such those between title and anchor text, we can extract possible spelling corrections. Similarly, such corrections can also be obtained using click-through data from the query log, where a query-document mismatch would indicate a spelling error. In our work, we view the extraction of correction records as a logical step that precedes transformation modeling. Our model can be easily extended to incorporate all sources of spelling correction pairs.

Our dataset for training the transformation model contains 1.4 million recourse link clicks. The statistics of the training data are shown in Table 4.1. Around 80% of all queries and 70% of all unique queries are correctly spelled. 1/10 of the training data is held out for parameter tuning.

Table 4.1: Statistics of training data

	Correctly Spelled	Misspelled	Total
Unique	101,640 (70%)	44,226 (30%)	145,866
Total	1,126,524 (80%)	283,854 (20%)	1,410,378

The query log we use for estimating the query popularity model consists of 21 million unique queries. Our test set is a human annotated set which contains 9,959 unique queries. Table 3 provides the statistics of the test data. The distribution over correctly spelled and misspelled queries is similar to that of the training data. 1/10 of the test data is also held out for tuning additional parameters, e.g. the coefficient λ for the mixture model. The remainder of the test queries is referred to as “all queries” in our evaluation results. The subset of misspelled queries within all queries is referred to as “misspelled queries”.

Table 4.2: Statistics of test data

	Correctly Spelled	Misspelled	Total
Unique	7585 (76%)	2374 (24%)	9959

4.5.2 Evaluation Metrics

We evaluate our methods with the following metrics:

R@N: Recall@N is the number of correct suggestions in the top ranked N suggestions generated by the system divided by the total number of suggestions in the ground truth. Since in our ground truth, each query has exactly one correction, the total number of suggestions is the same as the number of queries. Intuitively, Recall@N indicates the percentage of queries that the system can correct within the top N suggestions. Therefore, it is a very natural measurement for performance of correction. We take R@1 as our primary evaluation metric in experiment. Recall@N on all queries is also referred to as accuracy in other works [32].

P@N: Precision@N is the number of correct suggestions in the top ranked N suggestions generated by the system divided by the smaller value of N or the total number of suggestions generated by the system. Precision reveals the quality of suggestions generated by the system. Penalty is given to generating more incorrect suggestions. Note that this definition is different from another widely used definition of P@N, where the denominator is fixed to be N. Our definition can be interpreted as the precision of a system that limits its number of outputs to N at most. It is also worth noting that while the micro average and macro average for recall are the same, it is not the case for precision. For precision, we take the micro average because for queries where the system provides no suggestion, precision is not well defined.

R@N and P@N are metrics for measuring offline spelling correction. We use these metrics to evaluate our system in the exact match mode. Next, we introduce two metrics for measuring the performance of online spelling correction.

MKS: Minimal Keystrokes measures the minimal number of key presses the user has to make in order to issue the target search query. This metric simulates the scenario of users entering queries to search engines. Suppose the users query is *inportan* and the correct query is *important*. The user types in each letter in *inportan* sequentially. In the case that no suggestion is available, the user types in all the letters in *inportan* and presses the Enter key. Then the user can click on the recourse link provided by the offline speller. Therefore, the total number of keystrokes the user makes is the length of *inportan*, plus 1 for the Enter key, and 1 for the recourse link click. When suggestions are provided while the user is typing, she can use arrow keys to select a query from the suggestion list. For example, after typing in *inpo*, the user sees that *important* appears at the fifth position in the suggestion list. Thus, she can select the query by pressing the Down Arrow key 5 times, followed by the Enter key. In this case, the number of keystrokes is the number of letters the user enters (4) plus the number of arrow keys the user hits (5), plus 1 for the Enter key. If the user continues typing the rest of the query, she may see *important* increase to rank one for the input *inpor*. In this case, the number of keystrokes is 5+1+1=7, which is the minimal number of keystrokes (MKS). A good correction mechanism should have low MKS. In our experiments, we consider superstrings of the target query as positive matches, too. That is, in the case that "*important people*" is suggested instead of "*important*", we still treat it as a match.

PMKS: PMKS refers to penalized MKS, which adds a penalty to MKS for each suggestion generated by the system, as it takes effort for users to examine them for correctness. In this work we heuristically assign 0.1 keystrokes as the penalty for showing each suggestion. Thus, reading each query suggestion costs one tenth the effort of pressing a key. The essential idea of minimizing effort in MKS and PMKS is of independent research interest and could be applied to a wide range of research studies.

4.5.3 Experiment Results

In this section, we study the performance of our proposed system. We conduct all experiments on both the all queries and misspelled queries test sets to demonstrate the overall performance as well as the ability to handle misspelled queries.

We first compare our system with existing baselines in Table 4.3. The first baseline we include is the edit distance model used by Chaudhuri and Kaushik [13]. To the best of our knowledge, this is the only existing research study on online spelling correction. Our system outperforms the edit distance model in terms of all evaluation metrics. Significance test (t-test) shows that the improvement of our system is significant (p-value < 0.05) for all measurements except R@10. This indicates that most misspellings are not very severe; therefore the edit distance model is able to rank the best correction among the top 10 suggestions. However, the edit distance model is not able to further distinguish corrections within the same edit distance. We further observe that although we see a big gap in R@1 for misspelled queries, the overall performance difference for all queries is less than that of the misspelled queries. This is expected as the edit distance model will always rank identical transformation on top (if it exists in the query log).

We also include Googles online query spelling suggestions[1] as a baseline. As it is unclear how Googles online spelling suggestion can be configured to run in exact match mode, we only measure its performance with respect to the online correction metric, i.e. MKS. Not surprisingly, Google outperforms the simple edit-distance model. On average users save 0.38 keystrokes per query using Googles spelling suggestions over that of the edit distance model. For misspelled queries, nearly 1 keystroke is saved. Yet, our system further outperforms Googles suggestion system on MKS with a statistically significant 1.1 and 1.5 keystrokes savings on all queries and misspelled queries, respectively. It is worth noting that a larger search space (query log in our case) may result in worse performance. Since the size of Googles search space is unknown, we cannot jump to the conclusion that our system outperforms Googles spelling suggestion system.

Table 4.3: Comparison of performance with baseline systems

	All Queries			Misspelled Queries		
	R@1	R@10	MKS	R@1	R@10	MKS
EditDist	0.899	0.973	13.39	0.579	0.887	14.53
Google	N/A	N/A	13.01	N/A	N/A	13.49
Proposed	0.918	0.976	11.86	0.677	0.900	11.96

We also see in this experiment that the MKS metric is fairly consistent with Recall. Higher recall values always correspond to lower MKS. This validates the use of MKS as a performance metric.

To further understand how the proposed method works, we study the performance of the transformation model with different configurations of L and M. Figure 4.6 shows the effect of the transfeme Markov order M at $L = 1$ and $L = 2$. As we increase M from 1 to 2, we see a consistent increase in performance; but from 2 to 3, the performance decreased instead. This is contradictory with our intuition that higher order models result in better performance. We believe that this is because higher order models are more likely to suffer from data sparseness. Thus, with more training data, we may find higher order models to further improve the performance over $M = 2$. We also ob-

[1]Based on results collected on August 4, 2010.

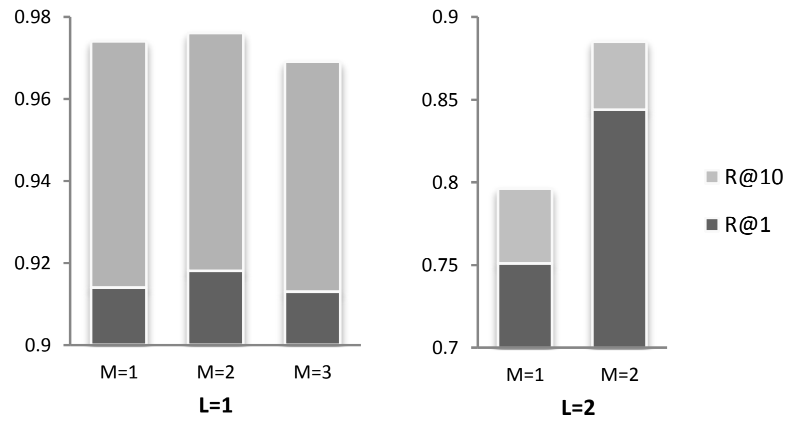


Figure 4.6: Performance with varying L and M

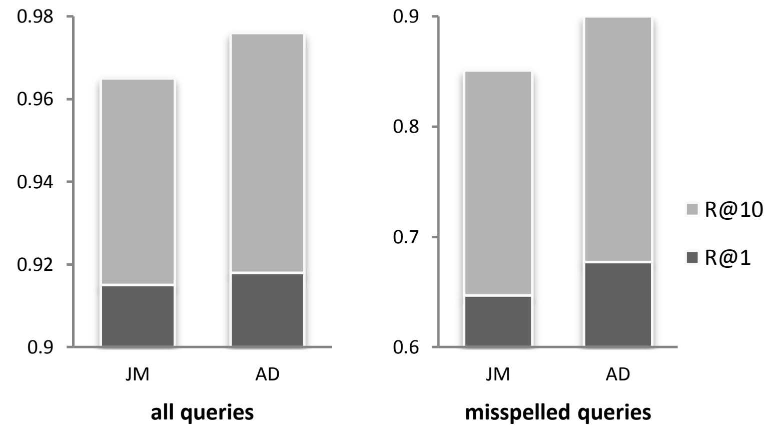


Figure 4.7: Performance of transformation models with different smoothing methods

serve that for a fixed M, increasing L actually decreases the performance. We hypothesize that this may be due to overfitting, as increasing L significantly increases the number of model parameters. As larger L also significantly increases the cost of search, it is impractical for real-time scenarios. Under the current setting, our best result is achieved with $L = 1, M = 2$. Thus for all subsequent experiments, we fix the configuration to $L = 1, M = 2$.

To confirm the effect of smoothing, we experiment with two smoothing methods and compare their performance. In Figure 4.7 we see that absolute discounting (AD) outperforms Jelinek-Mercer (JM) smoothing over every evaluation metric for both the all queries and misspelled queries test sets.

This is in line with previous language modeling research that found discounting based smoothing to outperform simple interpolation techniques. This experiment confirms our hypothesis that employing proper smoothing methods substantially increases the performance of the transformation model.

We present the effectiveness of our proposed methods for avoiding over correction in Table 4.4. As we can see, the non-mixture model, which is trained with misspelled queries only, performs well for misspelled queries. However, the overall performance is not good because it tends to alter queries that are already correctly spelled. Both the data mixture and model mixture approaches improve the overall performance by reducing such overcorrections. For the all queries set, they perform equally well. For misspelled queries, model mixture performs just as well as the non-mixture model. However the performance of the data mixture approach drops significantly. From an application perspective, it is the misspelled queries for which users need suggestions the most. Users are able to enter queries that they can spell no matter what our system suggests. In this sense, the model mixture approach is more preferable than the data mixture approach. Moreover, by estimating the two models separately, the model mixture approach can be updated more easily.

Table 4.4: Comparison of performance with baseline systems

	All Queries			Misspelled Queries		
	R@1	R@10	MKS	R@1	R@10	MKS
Non-Mix	0.893	0.966	11.94	0.678	0.899	11.98
Data Mix	0.918	0.971	11.85	0.669	0.879	11.98
Model Mix	0.918	0.976	11.86	0.677	0.900	11.96

In Table 4.5 and Table 4.6, we study the effect of the proposed thresholding method for pruning irrelevant suggestions. As we can see, with suggestion pruning, the performance of online spelling correction substantially increases for both the all queries and misspelled queries sets in terms of P@1, P@10 and PMKS. This verifies the effectiveness of our proposed thresholding method. But in terms of R@1, R@10 and MKS, the performance actually decreased. The reason behind this pattern is that the first set of metrics (P@1, P@10 and PMKS) assigns penalty for showing irrelevant suggestions, while the second set of metrics does not. In fact, any pruning of suggestions can only decrease the recall, as some correct suggestions may be pruned by mistake. From our perspective, showing too many irrelevant corrections has a strong negative effect on the query completion user

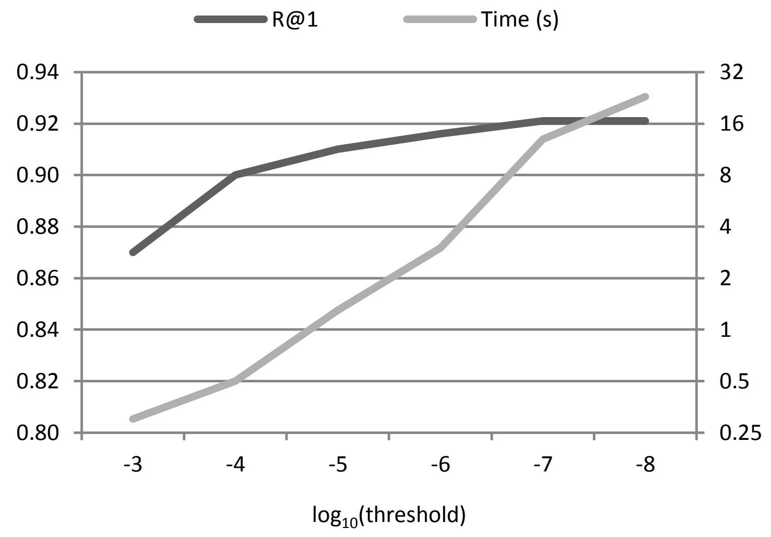


Figure 4.8: R@1/Running time vs. pruning threshold

experience, increasing the risk of losing users. Given that the recall did not significantly decrease, we prune suggestions using risk thresholding in the implementation of our system.

Table 4.5: Effect of pruning for all queries

	R@1	R@10	P@1	P@10	MKS	PMKS
w/ pruning	0.916	0.969	0.927	0.304	11.87	19.42
w/o pruning	0.918	0.976	0.920	0.262	11.86	19.60

Table 4.6: Effect of pruning for misspelled queries

	R@1	R@10	P@1	P@10	MKS	PMKS
w/ pruning	0.669	0.875	0.704	0.241	12.00	19.21
w/o pruning	0.677	0.900	0.685	0.204	11.96	19.56

Finally, we address the efficiency of our approach. From our experiments, we observe that although a better heuristic function can reduce the running time of the search algorithm, beam pruning is still required to achieve practical performance. In Figure 4.8 we plot the performance and running times for different relative beam pruning thresholds. Based on our experiments on an unoptimized implementation, we observe that as we relax the pruning threshold, the running time increases exponentially. However, the increase in R@1 is slow and ceases beyond a relative threshold of 10^{-7}.

Table 4.7: Examples suggestions

Input Query	Top Suggestion
milk shak	milkshake recipes
hwo to tain ur dra	how to train your dragon
alice on wander land	alice in wonderland
mision inpos	mission impossible

In Table 4.7 we list some example correction pairs identified by our system. None of these input queries are in the training corpus. As we can see, our method is capable of capturing various kinds of spelling errors for multiple word phrases. By updating the query language model frequently, we can keep our online spelling correction system up-to-date with the latest query language.

4.6 Conclusion

This work addresses the problem of query completion and correction with a generative model. We first propose a transfeme-based transformation model that is capable of capturing users completion and correction behavior. We then estimate the transformation model with an EM algorithm based on observed parallel queries. Next, we study various techniques to optimize the effectiveness of the transformation model.

To efficiently retrieve the query corrections with the highest probability according to the generative model, we propose the use of an algorithm based on A* search. The A* search algorithm is configured to deal with partial queries, so that online search is possible. We study different pruning and thresholding methods to improve the efficiency of the A* search.

Finally, we propose two novel evaluation metrics for online spelling correction, i.e. minimal keystrokes and penalized minimal keystrokes, based on the idea of minimal effort cost for users. We conduct extensive experiments and conclude that the proposed method is both effective and efficient for the task of online spelling correction.

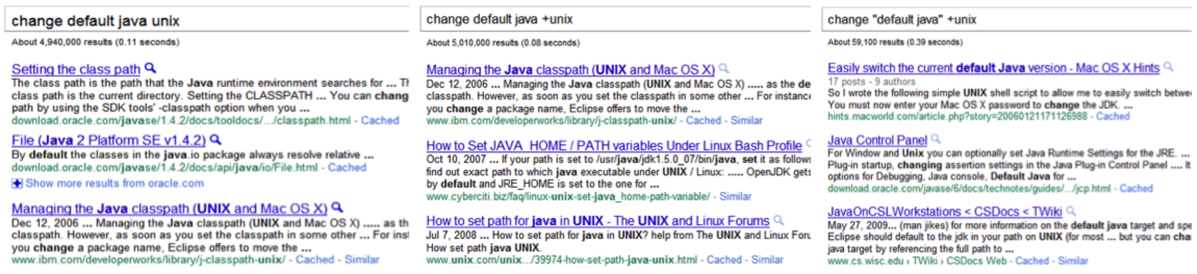
CHAPTER 5

QUERY REFORMULATION WITH SYNTACTIC OPERATORS

5.1 Introduction

Query languages of modern search engines usually include a set of advanced syntactic operators to supplement traditional keyword query. For instance, a necessity operator (plus sign) preceding a query term requires the term to be present in each relevant document; a phrase operator (a pair of quotation marks) imposes that relevant documents must contain the phrase consisting of the quoted terms. To distinguish from keyword query, we refer to a query with syntactic operators as a syntax query. For example, Figure 5.1a shows a keyword query, while Figure 5.1b and Figure 5.1c show two syntax queries which have the same set of terms as the keyword query shown in Figure 5.1a. In the convenience of discussion, we further denote this type of syntactic operator-based reformulation as *syntactic reformulation*.

If used appropriately, syntactic reformulation can strongly enhance the representation of search intent, turning an otherwise ineffective query to an effective one. Figure 5.1 shows an example of using syntactic reformulation to improve the top ranked results with a major US search engine. As we can see from Figure 5.1a, none of the top ranked documents in the search results are relevant to the query. In contrast, in Figure 5.1b, by using the syntactic query with the necessity constraint on term "unix", we are able to find two (2nd and 3rd) relevant documents out of the top three. This is because the search engine overlooked the term "unix" in the original query, which is typically due to the coarse estimation of term importance. By imposing the necessity constraint, the query promotes the importance of the term and re-ranks the results according to whether it is contained in each document. In Figure 5.1c, with an even more complicated syntactic reformulation, we further put a phrase constraint on "default java". This effectively eliminates the possible ambiguities of the query caused by matching terms separately. As a result, all the top three ranked documents are



change default java unix

About 4,940,000 results (0.11 seconds)

Setting the class path
The class path is the path that the **Java** runtime environment searches for ... Tl
class path is the current directory. Setting the CLASSPATH ... You can **chang**
path by using the SDK tools' -classpath option when you ...
download.oracle.com/**java**se/1.4.2/docs/tooldocs/.../classpath.html - Cached

File (**Java** 2 Platform SE v1.4.2)
By **default** the classes in the **java**.io package always resolve relative ...
download.oracle.com/**java**se/1.4.2/docs/api/**java**/io/File.html - Cached
Show more results from oracle.com

Managing the **Java** classpath (**UNIX** and Mac OS X)
Dec 12, 2006 ... Managing the **Java** classpath (**UNIX** and Mac OS X) ..... as th
classpath. However, as soon as you set the classpath in some other ... For inst
you **change** a package name, Eclipse offers to move the ...
www.ibm.com/developerworks/library/j-classpath-**unix**/ - Cached - Similar

a

change default java +unix

About 5,010,000 results (0.08 seconds)

Managing the **Java** classpath (**UNIX** and Mac OS X)
Dec 12, 2006 ... Managing the **Java** classpath (**UNIX** and Mac OS X) ..... as the **de**
classpath. However, as soon as you set the classpath in some other ... For instanc
you **change** a package name, Eclipse offers to move the ...
www.ibm.com/developerworks/library/j-classpath-**unix**/ - Cached - Similar

How to Set JAVA_HOME / PATH variables Under Linux Bash Profile
Oct 10, 2007 ... If your path is set to /usr/**java**/jdk1.5.0_07/bin/**java**, **set** it as follow
find out exact path to which **java** executable under **UNIX** / Linux: ..... OpenJDK gets
by **default** and JRE_HOME is set to the one for ...
www.cyberciti.biz/faq/linux-**unix**-set-**java**_home-path-variable/ - Similar

How to set path for **java** in **UNIX** - The **UNIX** and Linux Forums
Jul 7, 2008 ... How to set path for **java** in **UNIX**? help from The **UNIX** and Linux Foru
How set path **java UNIX**.
www.**unix**.com/**unix**.../39974-how-set-path-**java**-**unix**.html - Cached - Similar

b

change "default java" +unix

About 59,100 results (0.39 seconds)

Easily switch the current **default Java** version - Mac OS X Hints
17 posts - 9 authors
So I wrote the following simple **UNIX** shell script to allow me to easily switch betwe
You must now enter your Mac OS X password to **change** the JDK. ...
hints.macworld.com/article.php?story=20060121171126988 - Cached

Java Control Panel
For Window and **Unix** you can optionally set Java Runtime Settings for the JRE. ...
Plug-in startup, **changing** assertion settings in the Java Plug-in Control Panel .... It
options for Debugging, Java console, **Default Java** for ...
download.oracle.com/javase/6/docs/technotes/guides/.../jcp.html - Cached

JavaOnCSLWorkstations < CSDocs < TWiki
May 27, 2009... (man jikes) for more information on the **default java** target and spe
Eclipse should default to the jdk in your path on **UNIX** (for most ... but you can **cha**
java target by referencing the full path to ...
www.cs.wisc.edu › TWiki › CSDocs Web - Cached - Similar

c

Figure 5.1: Example of Using Syntactic Operators for Improving Search Result

relevant in this case. The proper use of syntactic operators not only clarifies users information need, but also gives clues to the retrieval system to optimize the search results.

However, very few users make use of the syntactic operators in their daily search activities. Statistics from a search engine query log show that only less than 0.5% queries used syntactic operators[1]. This is either because they are unfamiliar with the semantics of the operator, or because they lack the knowledge and statistics to formulate working syntax queries.

In this work, we propose to help users take advantage of the rich query syntax by automatically formulating potentially effective syntax queries for keyword queries. Particularly, we propose to automatically perform the query reformulation with syntactic operators when users encounter difficulty in search. Such difficulty could be detected through monitoring users behaviors. For instance, one possible indicator could be when a user skipped a page of top ranked results to the next page. Our assumption is that if a user can find relevant documents in the top ranked results, the user is likely satisfied with the original ranking and thus there is no much need for syntactic reformulation. We thus focus on studying how to exploit query syntax reformulation to help users when it is evident that the user cannot find any relevant document in the top k results. This scenario is similar to the case of negative relevance feedback [73][72] in the sense that there is a common goal of re-ranking the unseen results based on a small number of negative examples already encountered by the user. However, while existing negative feedback techniques primarily rely on word frequency analysis in the negative examples to improve the query vector or language model [73], we study how to leverage query syntax operators to improve the syntax of a query, which is a different and complementary strategy and has two advantages over the existing methods for negative feedback:

[1]Based on a sample of MSN query log in 2006.

first, it has better interpretability to users. i.e., the suggested reformulated query can be more easily verified by a user if deployed in an interactive feedback system; second, while most negative feedback methods “blindly” treat a whole document as non-relevant, our method would attempt to “diagnose” the cause of poor results and correct any missing semantic constraint via appropriate syntactic operator.

To perform automatic syntactic reformulation, we adopt a supervised learning framework. Particularly, we cast the problem as to predict the potential benefit of each candidate syntax query in improving the performance of a given keyword query. If we are confident on the potential improvement, we can suggest the reformulation to the user or use it directly to refine the search result. One problem in training the prediction model is that syntactic reformulations of different keyword queries may not have comparable performances. To circumvent this problem, we make use of the learning to rank techniques.

We propose and study three types of features to represent the characteristics of a syntax query. Query difficulty measures the intrinsic difficulty of syntax query. Distinguishability computes the amount of changes a syntax query brings to the original result. Negativity measures the similarity between the candidate query and the negative relevance feedback. The three types of features are all generalized features that can be computed for any filtering-based operators. Experiments demonstrate their effectiveness in formulating highly beneficial syntax queries.

To evaluate our proposed method for automatic syntactic reformulation and to test its usefulness on alleviating search difficulty, we conduct experiments with TREC dataset [71]. The experiment results verified the effectiveness of our proposed method for query reformulation with different syntactic operators. It is also demonstrated that our method can be easily applied to different retrieval models to improve search qualities. As a negative feedback method, our approach not only outperforms the state-of-the-art methods, but can also be combined with existing methods to further improve the performance.

Table 5.1: Performance Upper Bounds of Syntactic Operators in Alleviating Search Difficulty

Retrieval Model	**Query Type**	**Operator**	**NDCG@10**
BM25	Description	—	0.065
BM25	Description	Necessity	0.120
BM25	Description	Phrase	0.105
BM25	Title	—	0.078
BM25	Title	Necessity	0.094
BM25	Title	Phrase	0.099
LM	Description	—	0.070
LM	Description	Necessity	0.120
LM	Description	Phrase	0.111
LM	Title	—	0.083
LM	Title	Necessity	0.100
LM	Title	Phrase	0.102

5.2 Syntactic Operators for Improving Search Results

Modern search engines typically include a set of syntactic operators in their query language to complement the plain keyword query. Below we briefly introduce the two operators we study in this work, i.e. the necessity operator and phrase operator.

Necessity Operator (+). A necessity operator preceding a query term imposes the constraint that a relevant document must contain the particular term. For example, given a query *computer +virus*, the search engine shall return all documents containing the term *virus*, but not necessarily *computer*. In this way the query exaggerates the importance of the topical word to guarantee it will not be missed in the returned documents.

Phrase Operator (“”). A sequence of query terms surrounded by a pair of quotation marks indicates a phrase constraint. It requires a relevant document must contain the quoted phrase. For example, given a query *“computer virus” infection*, it intends to find documents containing the exact phrase *computer virus*, and better with the term *infection*. In effect, such a query would discard or degrade (depending on the implementation) a document which mentioned *computer* and *virus* multiple times but without mentioning the phrase.

Table 5.1 shows the performance upper bounds that can be achieved by applying syntactic reformulations. It validates our hypothesis on the effectiveness of syntactic operators in alleviating search difficulty, as we see huge performance improvements in syntactic reformulation compared with the baseline retrieval models. It is also observed that syntactic reformulations are more effective on long queries. As long queries are more verbose and noisy, their topics are more likely to be distracted. Syntactic operators are more helpful in this case as they emphasize on the underrepresented topics.

5.3 Automatic Reformulation with Syntactic Operators

We formulate the automatic syntactic reformulation of keyword queries as a supervised learning problem. Particularly, we cast the problem as to predict the benefit in performance for each candidate syntax query given a keyword query.

Formally, given a keyword query q, a syntactic operator op and a target performance metric M, our goal is to find a list of syntactic reformulations of q with op, denoted as $S_op(q) = q_1, q_2, , q_n$, which is ranked according to M in descending order: $M(q_1) > M(q_2) > ... > M(q_n)$. When it is required to output top m suggestions, the system will respond with a list consisting of q_1, q_2, q_m. When a syntactic reformulation is required to directly optimize the search results, the system will output the top ranked query q_1 if $M(q_1) > M(q)$, or the original query q otherwise. The training process is to learn a function f, which takes a set of features of a syntax query q_i, to predict the performance improvement $M'(q_i) = M(q_i) - M(q)$.

Since there are an exponential number of possible syntax queries for each syntactic operator, we limit the system to consider single appearance of each operator. Although we only explore the limited scope of use of these operators, the proposed methodology is general and can be potentially applicable to other filtering-based operators and multiple uses of each operator.

To solve the learning problem, a natural way is to use a regression model. In this method, a model will be learned to predict the performance for any possible syntax query. One problem with this method is that the performances of syntax queries reformulated based on different keyword queries are usually not directly comparable. For instance, from the example of training data in Table 5.2, we see that query 302 and query 313 have much different performances due to their intrinsic difficulties. To circumvent the problem, we adopt the learning to rank framework. In this setting, syntax queries generated from each keyword query are considered as a group; the loss function is defined on the ranking order of members in each group instead of on the absolute value of the performance. This loss function is more natural in our problem setting and avoids the issue of

Table 5.2: Examples of Training Samples for Syntactic Reformulation with Necessity Operator

QID	Syntax Query	NDCG@10
302	disease poliomyelitis polio under control world	0.163
302#1	+disease poliomyelitis polio under control world	0.105
302#2	disease poliomyelitis +polio under control world	0.203
302#3	disease poliomyelitis polio under +control world	0
313	commercial uses magnetic levitation	0.081
313#1	+commercial uses magnetic levitation	0.077
313#2	commercial uses +magnetic levitation	0.081
313#3	commercial uses magnetic +levitation	0.333

incomparability. We then use the learned model to predict the potential benefit in performance for each candidate in syntactic reformulation. Particularly, we use Ranking SVM [41] as the learning method in our study.

5.3.1 Features

We propose three types of features, namely *difficulty*, *distinguishability* and *negativity*. All three types of features are defined in a general way so that they can be computed for any type of filtering-based operators. As used in previous discussions, we denote q as the keyword query, *op* as the target operator and q_x as a syntactic reformulation of q with operator *op*.

Difficulty. The difficulty feature aims to measure the intrinsic difficulty of a syntax query. Query difficulty prediction has been widely studied in recent years. We modify the clarity feature proposed by Cronen-Townsend et al. [20] slightly to make it applicable to syntax queries:

$$Clarity(q_x) = KL(\theta_m \| \theta_C) = \sum_{w \in V} p(w|\theta_m) \log \frac{p(w|\theta_m)}{p(w|\theta_C)} \tag{5.1}$$

where θ_m is the language model estimated from the set of matched documents S_m of q_x, θ_C is the language model estimated from the entire collection. Queries with high clarity are more likely to work well.

In addition to clarity, we propose another difficulty feature, which is inspired by the inverse document frequency (IDF). We generalize this concept to compute the specificity of a syntax query:

$$GIDF(q_x) = \log \frac{N_c - N_{S_m} + 0.5}{N_{S_m} + 0.5} \tag{5.2}$$

where N_c and N_{S_m} are the number of documents in the entire collection and matched set, respectively. Intuitively, a query matching fewer documents is more specific and meaningful.

To understand how these two features work, let us take a look at an example. Suppose we are given a query *Oscar winner selection.* To reformulate with necessity operator, we evaluate the queries *+Oscar winner selection*, *Oscar +winner selection* and *Oscar winner +selection.* It can be expected that the first query would have much higher clarity and GIDF, as the word "Oscar" is associated with fewer documents, which are more likely to be focused on a particular topic. In reformulation with phrase operator, we consider the candidates *"Oscar winner" selection* and *Oscar "winner selection"*. Similarly, we could imagine the first query will have higher clarity and GIDF as the matched documents on "Oscar winner" is much fewer and more specific than documents matching "winner selection".

Distinguishability. The idea of the distinguishability feature is to quantify the changes a syntax query brings to the original query. For this purpose, we define cross clarity between θ_m and θ_q, the language model estimated from the search results of q:

$$CrossClarity(q_x, q) = KL(\theta_m \| \theta_q) = \sum_{w \in V} p(w|\theta_m) \log \frac{p(w|\theta_m)}{p(w|\theta_q)} \tag{5.3}$$

This feature measures the change in the topical formation of the syntax query q_x and the original query q.

Besides measuring content changes, we use another feature to measure the change in the ranking of documents. Particularly, we measure the correlation between the document rankings of q and q_x:

$$Cor(q_x, q) = \frac{\#concord(q_x, q) - \#discord(q_x, q)}{\frac{1}{2} N_q (N_q - 1)} \tag{5.4}$$

where $concord(q_x, q)$ and $discord(q_x, q)$ are the sets of concordant pairs and discordant pairs between the two ranking lists of search results of q_x and q. The correlation feature quantifies the changes q_x brings to the original ranking of q.

In the example of reformulating *Oscar winner selection* with phrase operator, we have two candidate queries *"Oscar winner" selection* and *Oscar "winner selection"*. The first query will have lower cross clarity and higher correlation with the original keyword query, as it captures the essential topic of the query. It is therefore more suitable for a suggestion.

Negativity. The negativity feature measures the similarity between the syntax query q_x and the negative documents (e.g. skipped documents in browsing). A query less similar to the negative documents naturally has higher chances of working well.

Again, we define the content based negativity feature as the cross clarity between the language model estimated from the matched documents S_m and the language model estimated from the negative feedback documents S_n.

In addition, we define another negative feature as the generalized inverse negative frequency (GINF):

$$GIDF(S_m, S_n) = \log \frac{N_{S_n} + 0.5}{N_{S_m \cap S_n} + 0.5} \tag{5.5}$$

In effect, this feature indicates the necessity of requiring a particular operator in the original keyword query.

By computing negativity features with different operators, we are actually "diagnosing" why the original query does not work well. For instance, if the negative documents of query *Oscar winner selection* discuss a lot about "Oscar winner", but rarely mention "selection", then it is probable that the user intend to focus on the "selection" of "Oscar winner", while the system is somehow biased towards other popular topics on "Oscar winner". In this case, the query *Oscar winner +selection* might be able to amend the mistake. Or if the negative documents mention the three keywords fairly well, but the phrase "Oscar winner" seldom occurs, it probably indicates that the retrieval system overlooked the fact that "Oscar winner" must be matched as a phrase to maintain its meaning. Therefore, the query *"Oscar winner" selection* might work well to serve the users purpose.

5.3.2 Combining Operators in Prediction

Our proposed algorithm not only works for predicting each operator separately, but can also be applied to predict different operators jointly, as filters can be applied additively. We refer to this joint prediction method as Operator-Combination.

An alternative method is Result-Combination, in which we predict each operator separately and select the reformulation with the best predicted performance. In practice we find this method not only more effective but also more efficient than the Operator-Combination, as it considers much fewer candidates in prediction.

5.4 Experiments

In this section, we present the experiment results for evaluating the proposed methods. We begin by introducing the experiment setup, followed by the details of our evaluation results. Finally, case studies are presented for better understanding of our system.

5.4.1 Experiment Setup

We use TREC 2004 Robust track [71] as our experiment dataset. The dataset includes TREC disk 4&5 minus congress reports. There are around 500,000 documents in the dataset. The query set consists of 250 queries. The title field and description field are used to represent different types of queries. The average length of title field and description field is 2.7 and 15.6 terms, respectively. We refer the title field as (relatively) short query, and the description field as (relatively) long query.

Not all the queries are difficult queries. To simulate the scenario of search difficulty, we adopt the minimum deletion method proposed by Wang et al. [73]. We use BM25 as our baseline. In addition, we also implement MultiNeg, a state-of-the-art method for using negative feedback [73]. It is worth noting that our method does not conflict with existing methods for using negative feedback. Instead, we use negative feedback in a different manner, which could potentially bring complementary factor to the existing methods. To test this idea, we further combine the use of syntactic reformulation with the MultiNeg method.

Table 5.3: Automatic Syntactic Reformulation to Directly Refine Search Results on Long Queries (Description) with BM25

	NDCG@10	**P@1**	**MAP**
BM25	0.065	0.157	0.089
Necessity†	0.077	0.205	0.104
Phrase	0.069	0.161	0.093
Operator-Combination	0.065	0.161	0.088
Result-Combination†	0.079	0.209	0.114
MultiNeg†	0.074	0.216	0.093
RC+MultiNeg†‡	0.081	0.221	0.115

MultiNeg Wang et al. reported that the MultiNeg strategy outperforms all the other existing methods for using negative feedback, including Rocchio like SingleQuery method and SingleNeg method. Therefore, we implement MultiNeg methods for both BM25 as a baseline system. The MultiNeg methods work by combining the original score of each document to be re-ranked with a penalty score.

$$S_{combined}(Q, D) = S(Q, D) - \beta S(Q_{neg}, D) \tag{5.6}$$

where the penalty score is computed by looking at each negative document separately:

$$S(Q_{neg}, D) = \max_{Q' \in N} S(Q', D) \tag{5.7}$$

where $S(Q', D)$ is the similarity of the negative document Q' and document D. In this case, it is the BM25 ranking score.

We evaluate automatic syntactic reformulation with necessity and phrase operator by using them to directly refine search result by re-ranking unseen documents. NDCG@10 is used as the primary metric. All the reported results are based on 5-fold cross validation.

5.4.2 Experiment Results

Table 5.3 and Table 5.4 show the performances on long queries (description) and short queries (title) respectively. The base retrieval model we use is BM25. Runs that show statistical significant improvement over the baseline model ($p - value < 0.05$) are marked by †. We also compare our performances with another baseline system for using negative feedback, i.e. MultiNeg. Runs that

Table 5.4: Automatic Syntactic Reformulation to Directly Refine Search Results on Short Queries (Title) with BM25

	NDCG@10	**P@1**	**MAP**
BM25	0.078	0.217	0.111
Necessity	0.081	0.229	0.115
Phrase	0.083	0.221	0.119
Operator-Combination	0.076	0.201	0.108
Result-Combination	0.082	0.225	0.119
MultiNeg	0.078	0.216	0.111
RC+MultiNeg†‡	0.084	0.225	0.119

show statistical significant improvement over the MultiNeg methods ($p-value < 0.05$) are marked by ‡.

We see that for long queries, reformulation with necessity operator achieves higher performances than with phrase operator. However, for short queries, phrase operator tends to work better. Long queries are much more verbose and noisy, and thus more likely have the central topic missed in returned documents. Reformulation with necessity operator is more useful here as it discovers the important but underrepresented topical term and ensures it to be matched. A short query has fewer keywords and therefore less noise. However, the connection between keywords is usually lost as a cost of being succinct. Reformulation with phrase operator alleviates the problem by imposing the phrase constraint strongly connected terms. We also see the Result-Combination strategy always outperforms Operator-Combination. Result-Combination usually brings further improvement to the performance. More importantly, it provides a more robust solution compared with reformulation with single type of operator.

Result-Combination outperforms MultiNeg. MultiNeg method takes the entire content of a negative document as non-relevant. In our method, we try to find the commonly missing semantics among negative examples. Compared with MultiNeg, this is a more precise way of using negative feedback and is therefore more effective. The performance could be further improved by combining syntactic reformulation with MultiNeg (RC-MultiNeg). Since our method does not directly use the negative information to refine scores of documents, it is complementary with the existing methods that work in this way.

We see that it is generally more difficult to improve the search results for short queries. Short queries are typically more succinct. As a result, there is less room for applying syntactic operators.

Table 5.5: Examples of Suggested Queries

ID	Query		NDCG
1	Original	find instances plagiarism literary journalistic worlds	0.023
	Suggested	find instances +plagiarism literary journalistic worlds	0.137
2	Original	fear open public places agoraphobia widespread disorder relatively unknown	0.000
	Suggested	fear open public places +agoraphobia widespread disorder relatively unknown	0.142
3	Original	impact chunnel british economy life style british	0.046
	Suggested	impact +chunnel british economy life style british	0.201
4	Original	commercial uses magnetic levitation	0.081
	Suggested	commercial uses “magnetic levitation”	0.333
5	Original	maternity leave policies various governments	0.208
	Suggested	“maternity leave” policies various governments	0.330

5.4.3 Cases Studies

In order to better understand how syntactic reformulation works for improving retrieval performance, we show some concrete examples of automatically reformulated syntax queries in Table 5.5.

From these examples, we see syntactic operators help convey query intents and clear ambiguities. For instance, the original query of query 5 does not convey clearly that “maternity leave” is a phrase with a specific meaning. It caused ambiguities as the terms are matched separately. Automatic syntactic reformulation eliminates the ambiguities by stressing the phrase match on the two terms.

Syntactic reformulation also discovers the underrepresented concepts in the keyword queries. For instance, the term “chunnel” in query 3 is overlooked in the original query as there are other popular topics in the collection that match the rest of keywords well. Our algorithm is able to detect this problem and solve it by applying necessity operator on the term.

Our algorithm is also able to discover the representative terms in queries. In query 2, “agoraphobia” basically represents the entire query intent, while explanation terms are noisy and distractive. By emphasizing on this term, it maintains relevance to the central topic and dismisses the unnecessary distractions.

5.5 Conclusion

The ability to help users reformulate effective queries when their original queries do not work well is an important enhancement to search intent representation. In this work we propose and study a novel way of automatic query reformulation through the use of query syntax operators, i.e. syntactic query reformulation. With appropriate use of syntactic operators, we are able to detect the missing semantics in the original query and amend it in the reformulated query. We formulate automatic syntactic reformulation as a supervised learning problem under the framework of learning to rank. We propose a set of effective features to represent the characteristics of syntax queries. Extensive experiments are conducted to demonstrate the effectiveness of the proposed methods.

Part III

Deep Intent Modeling for Structured Entity Retrieval

CHAPTER 6

MODELING QUERY INTENT WITH ENTITY STRUCTURES

6.1 Introduction

My previous work has focused on general techniques for search intent modeling and query reformulation. Accurate and comprehensive modeling of search intent in Web search engines is usually difficult as search intent takes various forms on the Web. Recently, large amount of structured data are becoming available in many areas, such as e-commerce and medical information systems, leading to an emerging and important issue in IR, i.e. structured entity search. The problem of structured entity search provides great opportunities for in-depth understanding and modeling of search intents, but also raises many challenges. On one hand, the types of search intent are not as versatile as in unstructured data, which allows us to focus on certain types of search intents, e.g. shopping intent in e-commerce. On the other hand, there is a much more obvious gap between user described intent (in unstructured keywords) and the searchable data (in structured representations) that need to be solved, as compared to typical document retrieval. Due to these differences, existing methods cannot be directly applied to achieve satisfactory results. More importantly, no existing work has fully studied the problem of deep search intent modeling in document retrieval due to the lack of sub-document structures. In this chapter, I study modeling query intent with entity structures.

In e-commerce, being able to let users explore the inventory conveniently has become a necessary and critical service for online retailers. Usually, the product inventory is stored in a structured / semi-structured database, where each entry is an entity representing a particular product and each column describes an aspect of the product. Table 6.1 shows an example database of laptops. In this example, we have seven entities and four attributes. Each attribute may have one or more values. We refer to each attribute-value pair as a *specification* (*abbr. spec*). Each entity is therefore represented

Table 6.1: Example laptop database

Brand	Hard Drive	Graphics	Blu-ray
HP	750G	Radeon HD 7690M XT	No
Dell	782G	NVIDIA N13P-GS	Yes
Acer	500G	Intel HD	No
Asus	128G	UMA	No
Acer	500G	Radeon HD 7640G	Yes
Asus	750G	Intel HD Graphics 3000	No
Sony	640G	Intel HD Graphics 4000	No

by a set of specifications (specs). Traditionally, such data storage can be conveniently accessed by structured queries (e.g. SQL). For example, the query

*select * from laptop where hard-drive > 500G and blu-ray = "Yes"*

searches for laptops that have a large hard disk and blu-ray player. However, end users rarely understand the semantics of such structured queries, and even for a user who is familiar with the query language, it is still a challenge to construct effective queries due to the lack of knowledge of the data. For example, to search for laptops with dedicated graphics cards, we may have to write a long query listing all types of graphics cards except those integrated ones. It is thus necessary to allow users to search for products using keyword queries to express their preferences. Unfortunately, such natural language keyword queries do not clearly specify which products should be returned, thus making it rather challenging to accurately rank product entities so that highly relevant products would be ranked on the top. Traditional methods based on keyword matching are unlikely to work well due to the vocabulary gap between the product specifications and the keywords people use in search queries. Indeed, how to optimize ranking of product entities in response to a keyword preference query has not been well studied in the existing work and is largely an open research question. In order to solve this problem, we need to be able to model query intent with a data level representation.

As repeatedly shown in many other information retrieval (IR) tasks, the accuracy of a search system is largely determined by the soundness of the retrieval model adopted. The lack of a sound retrieval model for product ranking, thus, has hindered the progress in optimizing the ranking accuracy of keyword search in product database. The main goal of our work is thus to develop a sound general probabilistic model for product entity retrieval that can be used in *all* keyword-based product entity search engines to optimize ranking accuracy. Since probabilistic retrieval models have enjoyed much success for ad hoc text retrieval tasks and can be well justified theoretically based on the probability ranking principle [59], and the query likelihood retrieval model, which can be derived based on probability ranking principle using query generation decomposition [46], is quite effective [54, 80], we follow a similar process of probabilistic reasoning, and propose a novel probabilistic model for product entity retrieval based on query generation.

In the proposed model, ranking is primarily based on the probability that a user interested in an entity will pose the query. The model attempts to simulate the query formulation process of a user and score each entity e for a query q based on the conditional probability $p(q|e)$ which captures the likelihood that a user who likes entity e would use query q (to retrieve entity e). Essentially, we associate with each candidate entity a hypothesis "user likes this entity" , and use the observed query q as evidence to infer which hypothesis is true (equivalently which entity is liked by the user). The posterior probability of each hypothesis (equivalently each entity) can then be used for ranking product entities for a given query. Such a model can naturally incorporate prior preferences over product entities into ranking in addition to modeling how well an entity matches a query; it can also naturally separate the two subtasks, i.e., entity type matching and entity preference scoring. As a first step in studying probabilistic models for product entity ranking, in this work we focus on studying the second subtask of preference scoring. This is mainly because many existing product retrieval systems allow a user to choose a product category (which is quite easy for a user to do) in addition to entering keyword preferences, thus the main challenge in improving such a system is to improve the preference modeling. Naturally, an important future work would be to also study the orthogonal problem of entity type matching using the proposed general probabilistic framework.

A key component in our model from the perspective of preference scoring is to model the probability of using a word w in a query by a user who likes entity e, i.e., how we "generate" a query word from an entity. We propose to refine this component based on the attributes of product entities, which roughly models the following generation process of a query. A query is generated by repeatedly sampling a word as follows. A user who likes entity e would first sample an attribute to query according to a specification selection model $p(s|e)$ where s is a spec of e, and then sample a word w from an attribute-specific unigram language model $p(w|s)$. Such a decomposition allows us to model a user's preferences at the attribute level.

Different ways of estimating each component model will lead to different variants of ranking functions. We propose and study several ways to estimate the proposed model, leveraging the product specifications as well as the associated text data such as product reviews and logged search queries. In particular, in contrast to the existing way of using a text description as a whole for ranking entities, we treat text data in a novel way by learning attribute-specific language models from it, which can then be used to improve ranking accuracy for product entity retrieval.

Since product entity retrieval has not been well studied, there is no public available test collection that we can use for evaluation. To address this challenge, we created our own test collections from two major e-commerce systems. We run comprehensive experiments to evaluate the proposed models. Our experimental results show that the attribute-level modeling of relevance, enabled by the proposed model, is more effective than the baseline approaches which straightly model relevance at the entity level. Experiments also show that the proposed model can effectively leverage review and search log data to significantly improve product ranking accuracy, and as in the case of text retrieval, smoothing of the language models is critical and a robust estimate based on interpolation of the proposed model with entity-level language model works the best on our datasets.

Although the probabilistic model introduced in this work is primarily proposed for the purpose of improving product entity retrieval, it also serves as a general approach for modeling structured/semi-structured entity data (e.g. product specifications) coupled with unstructured text data (e.g. user reviews). The model naturally leads to many other interesting applications besides supporting keyword search. We explore the use of the proposed model in two interesting applications: facet generation and review annotation, and demonstrate the effectiveness with promising results.

6.2 Probabilistic Query Model for Ranking Product Entity

From the perspective of information retrieval, the problem of keyword search in product database (i.e. product entity retrieval) is related to several other retrieval tasks, but is unique in many ways. First, it is different from a regular text retrieval problem (e.g., ordinary Web search) in that the "documents" are product entities, which are very well structured and that the relevance of a product entity to a query is primarily based on how well the attribute values (i.e. specifications) of the product match the preferences of the user expressed in the query. This calls for fine-grained modeling of relevance at the attribute level. Second, it is also different from the entity retrieval problem in the form of expert finding [4] or entity search on the Web [17, 16] where the data assumed available are free text data and information extraction techniques are often used to extract relevant entities. Third, it is different from XML retrieval in that the schema for the entity database is generally assumed to be fixed, while the queries are keyword preference queries as opposed to the more structured XML queries. Finally, as a special case of keyword search on databases, our problem formulation has a clearly defined unit for retrieval and emphasizes on modeling relevance at a finer level to satisfy the preferences expressed in fuzzy keyword queries.

While many product search systems exist on the Web, there has been surprisingly little research on developing general product entity retrieval models for optimizing ranking accuracy in this special, yet important retrieval task. As was shown in other search tasks, developing computational retrieval models to model relevance accurately is the key to optimizing ranking accuracy in a search task.

To systematically optimize accuracy for product entity retrieval, we propose a novel general probabilistic model adapted the general idea of query likelihood retrieval model, study various ways to refine the model and estimate the component probabilities, and evaluate multiple specific product entity ranking functions obtained through using different estimation methods. Below we first present the proposed probabilistic entity retrieval model. Although our main motivation for developing this model is to optimize product entity retrieval, the model is actually general and potentially applicable to any entity retrieval problem where keyword queries are used to express preferences on various attributes.

6.2.1 Probabilistic Entity Ranking Based on Query Generation

Formally, we are given a set of entities E. Each entity e in E is described by a list of specifications S_e:

$$S_e = \{s | s \in S\} \tag{6.1}$$

where each specification $s = (a_s, v_s)$ is an attribute-value pair represented by an attribute name a_s (e.g. “brand”) and a value v_s (e.g. “dell”), S is the set of all possible attribute-value pairs. Given a user entered keyword query q, our task is to rank the entities in E according to how likely they will satisfy the user's preferences encoded in q.

Following the derivation of the query likelihood retrieval model in [6], we model the relevance of an entity with conditional probability $p(e|q)$, which can be interpreted as the posterior probability that entity e is liked by the user after we observe the user's query q. With Bayes rule, we have:

$$p(e|q) = \frac{p(q|e) \cdot p(e)}{p(q)} \propto p(q|e) \cdot p(e) \tag{6.2}$$

Since $p(q)$ is only dependent on the query, it can be ignored for the purpose of ranking entities. Therefore, the ranking function only depends on two component probabilities. The first is $p(e)$, which is the prior probability that entity e is liked by a user (the term "prior" can be interpreted as our belief about the relevance of entity e *before* we even see a query). Intuitively, this prior probability may prefer a popular entity to a non-popular one. The second is $p(q|e)$, which is the likelihood that we would observe query q if e is indeed relevant (i.e., liked by the user). This conditional probability can capture how well entity e matches the query q in the sense that if a user likes entity e, the user would likely pose a query matching the attribute values of entity e. The posterior probability of relevance $p(e|q)$ can be regarded as our updated belief about the relevance of entity e *after* observing the query q; ranking based on this posterior probability would naturally prefer an entity that has a high prior probability as well as matches the given query well.

From retrieval perspective, the inclusion of a prior probability $p(e)$ naturally enables us to incorporate any query-independent factors into our ranking function (e.g. popularity statistics of products). In this work, however, we do not assume any knowledge about the entities, thus we simply assume a non-informative (uniform) prior. Our ranking function would then boil down to ranking solely based on $p(q|e)$, which is essentially the query likelihood retrieval function which has proven quite effective for regular text retrieval [54, 80]. However, the challenge is now how to further refine $p(q|e)$ so that we can accurately model the special notion of relevance in product search.

Conceptually, we can refine $p(q|e)$ by modeling the process of query formulation of users. Imagine a user likes an entity e, and we would examine the question how such a user would formulate a query in order to retrieve entity e. Intuitively, the user would have to specify two components in the query: 1) the entity type (e.g., "laptop"), and 2) preferences on attribute values for the target entity (e.g., "small", "cheap"). We can thus assume our query has two parts correspondingly, i.e., $q = (q_t, q_p)$, where q_t is a term denoting the desired entity type and q_p is a keyword query expressing preferences on attribute values. A user's choice of q_t is logically independent of the preferences q_p. Thus we have $p(q|e) = p(q_t, q_p|e) = p(q_t|e)p(q_p|e)$. That is, our task now is to model separately how a user expresses the desired category of entity and how a user expresses preference of values on each attribute.

While in general, the selection of entity category may also be uncertain, in virtually all real applications of product search, the categories of all the products in a database are usually known. Indeed, a user is often asked to select a category of products in addition to entering preference keywords. That is, $p(q_t|e)$ is no longer uncertain in most applications and we have $p(q_t = c|e) = 1$ if c is the category of e, and $p(q_t = c|e) = 0$ for all other categories. The consequence of this assumption is that we would only consider entities of the same category as the category selected by the user (since all other entities would have a zero probability for $p(q_t|e)$, thus also a zero posterior probability $p(e|q)$). In the following, we therefore focus on discussing how we further decompose $p(q_p|e)$ and estimate the model. We want to stress, though, that our proposed model can easily accommodate alternative ways of refining $p(q_t|e)$ (e.g., to accommodate inexact category matching based on ontology).

Again, we refine $p(q_p|e)$ by exploring how a user expresses preferences if the user likes entity e. Intuitively, if a user likes entity e, the user must have liked some of the specifications of e. Thus, it is reasonable to assume that the user would formulate a preference query by first selecting an interesting specification of e and then choosing appropriate words to describe his/her preference on the value of the chosen specification. That is, the probability that a user, who likes entity e, would use word w in the query is given by

$$p(w|e) = \sum_{s \in S} p(w|s)p(s|e) \tag{6.3}$$

where $p(w|s)$ is the unigram language model probability that a user would use word w in the query if the user likes specification s, satisfying the constraint $\sum_{w \in V} p(w|s) = 1$. Naturally, $p(s|e)$ captures the intensity that users are interested in spec s, as opposed to other specs of the entity. It satisfies $\sum_{s \in S} p(s|e) = 1$.

For example, if e is well known as a cheap small laptop, then we could assume that both $p(\text{“}size = small\text{”}|e)$ and $p(\text{“}price = under\ \$250\text{”}|e)$ are reasonably high, likely higher than other specifications on “RAM” or “warranty”. Also, we would expect the language model conditioned on the specification “$(size = small)$” would give higher probabilities to words such as “small”, “portable”, or “lightweight” than other words such as “fast” or “powerful”.

Thus the probability of generating a multiword preference query q_p based on entity e would be

$$p(q|e) = \prod_{w \in V} [\sum_{s \in S} p(w|s)p(s|e)]^{c(w,q)} \tag{6.4}$$

where $c(w, q)$ is the count of word w in q. Note that for convenience, here we have dropped the subscript p in q_p and simply use q to denote preference query q_p since there is no concern of ambiguity.

Clearly, our model allows a user to include preferences on multiple attributes in a single query as it should. Intuitively, the model would favor those entities with attribute values whose preference language model can explain the preference words in the query well, effectively capturing the relevance criterion for product ranking, i.e., ranking product entities whose attribute values match the user's preferences in the query well on the top.

To avoid underflow caused by multiplication of small values, we can score an entity based on the log-likelihood of the query given an entity, which leads to the following general scoring function for product entity ranking:

$$score(q, e) = \log p(q|e) = \sum_{w \in V} c(w, q) \log \sum_{s \in S} p(w|s)p(s|e) \tag{6.5}$$

Since in general, the available data for model estimation would be limited, appropriate smoothing is necessary to avoid zero probabilities. To do this, we assume that there exists a “generic specification” (s_g) whose corresponding specification preference language model is a general background language model θ_B that would give a non-zero probability to every word (token) in our specification database. By allowing such a generic specification to be chosen potentially for every entity, we can ensure that our estimated models would not assign zero probability to any query word that occurs in our database. Specifically, we can assume that with probability λ, the user would choose this generic specification when formulating the query (i.e., $p(s_g|e) = \lambda$), and thus have

$$score(q,e) = \sum_{w \in V} c(w,q) \log \left[\lambda p(w|\theta_B) + (1-\lambda) \sum_{s \in S} p(w|s) p(s|e) \right] \tag{6.6}$$

The background language model θ_B can be estimated based on normalized word counts in the entire database.

It is now clear that in order to make such a general model actually work for product ranking, we must be able to estimate the following two component models:

1. **specification selection model ($p(s|e)$):** this is the probability that a user who likes entity e would include a preference on the specification s in the query.

2. **specification preference language model ($p(w|s)$):** this is the probability that a user would use word w in the query to describe a preference if the user likes specification s.

By exploring different ways of estimating these two probabilities, we can derive many interesting instantiations of this general model, which would lead to distinct ranking functions. In this sense, our model not only provides a theoretical foundation for optimizing product entity search, but also serves as a constructive road map for exploring many interesting ranking functions. In the following, we will first discuss how to estimate these component models based solely on the product specifications, and then we study how to leverage the available text data for product entities to solve the vocabulary gap problem and improve the model estimation.

6.2.2 Model Estimation Based on Entity Specifications

As previously stated, the key question in model estimation is to estimate the preference selection probability $p(s|e)$ and preference language model $p(w|s)$. Without assuming any further knowledge or search log data available, we can only use the product specification data stored in the database to estimate our model.

Let us first look at the specification selection probability. Indeed, without additional knowledge, it is very difficult to guess which attributes are more interesting to a user who likes entity e. Therefore a conservative estimate would be to assume each attribute of entity e is equally likely to be selected. We refer to this estimate the

Uniform Specification Selection (USS):

$$p(s|e) = \begin{cases} 1/|S_e| & s \in S_e \\ o & otherwise \end{cases} \tag{6.7}$$

Clearly, USS is a coarse estimation. As an alternative estimate, rather than assuming a uniform posterior distribution, we assume a uniform prior $p(s) = 1/|S|_u$, where $|S|_u$ is the count of unique specs in S. Then we can derive the new estimate,

Uniform Prior Selection (UPS):

$$p(s|e) = \frac{p(e|s)p(s)}{\sum_{s' \in S} p(e|s')p(s')} = \frac{p(e|s)}{\sum_{s' \in S} p(e|s')} \tag{6.8}$$

where $p(e|s)$ is assumed to be uniform distributed over all the entities containing spec s, and zero otherwise:

$$p(e|s) = \begin{cases} 1/|E_s| & e \in E_s \\ o & otherwise \end{cases} \tag{6.9}$$

where E_s is the set of entities that contain spec s.

In effect, this estimate is similar to the Inverse Document Frequency (IDF) used in document retrieval. A specification unique to entity e would be more likely chosen by a user to express a preference in the query, at least more likely than a very popular feature that is shared by many entities.

For the preference language model $p(w|s)$, a reasonable estimate would be to concatenate the attribute name and value into a short text description and normalize the counts of words in such a text description. That is,

Attribute-Value Text:

$$p(w|s) = \frac{c(w,s)}{\sum_w c(w,s)} \tag{6.10}$$

where $c(w,s)$ is the count of word w in the concatenated text description of specification s.

These estimates can be plugged into our general entity ranking model to obtain different specific ranking functions, which we will evaluate later in the paper.

6.2.3 Improve Estimation by Leveraging Associated Text Data

The estimators discussed above are based solely on the entity database, which make them general. However, as the text data in the entity database is quite limited, these estimates would unlikely be accurate in capturing users' language models. To solve the problem of vocabulary gap, we propose to leverage the user generated text data to improve the estimation of our model. The most useful data is the search log of product search engine, where we can associate user queries with entities by looking at user engagement behavior. However, this requires a product search engine that have already works reasonably well and moreover, the search log data is usually proprietary and thus can only be leveraged inside industry labs. User reviews, on the other hand, are public, easy to obtain and does not have any prerequisite on search engines. Indeed, product reviews have become increasingly available on the Web. They are composed by users and is thus a homogenous datasource of search queries.

In this work, we propose a general method for taking advantage of any kind of text data associated with product entities to improve keyword search in product database, including both logged queries and user reviews, by treating them as data samples generated from the models of the corresponding entities.

6.2.4 A Mixture Model of Review Text Data

Without loss of generality, let us consider a "training set" (for our model) composing of a set of entities E and a set of text descriptions R. Each entity $e \in E$ is associated with a text based description $r_e \in R$. If more than one such descriptions are available for an entity, we can combine them to form a long description if they are of the same type, or use them separately to estimate parallel models if they are different. Our key insight in leveraging text descriptions for estimating both the specification selection model and the preference language model is that we can assume the text associated with entity e, r_e, is generated based on entity e through a similar generation process of an expected query. For previous queries that resulted in clicking the entity, this is obvious; for user reviews, we assume that when a reviewer writes a review, the reviewer would first sample a specification of the entity to discuss in the review, and then sample words discussing the selected corresponding attribute and value. Formally, the log-likelihood of observing text description r_e is thus:

$$\log p(r_e|e) = \sum_{w \in V} c(w, r_e) \log \left[\lambda p(w|\theta_B) + (1-\lambda) \sum_{s \in S} p(w|s) p(s|e)\right] \tag{6.11}$$

We can then use the Maximum Likelihood estimator to estimate both the specification selection model $p(s|e)$ and the specification preference language model $p(w|s)$ by maximizing the following likelihood function of the entire dataset:

$$\mathcal{F} = \sum_{e \in E} \sum_{w \in V} c(w, r_e) \log \left[\lambda p(w|\theta_B) + (1-\lambda) \sum_{s \in S} p(w|s) p(s|e)\right] \tag{6.12}$$

To do so we employ an Expectation-Maximization (EM) algorithm. In the E-step, we compute the contribution of each specification in generating each word in the text data:

$$p(s|w, e) = \frac{p(w|s)p(s|e)}{\sum_{s' \in S} p(w|s')p(s'|e)} \tag{6.13}$$

$$p(\theta_B|w, e) = \frac{\lambda p(w|\theta_B)}{\lambda p(w|\theta_B) + (1-\lambda) \sum_{s \in S} p(w|s)p(s|e)} \tag{6.14}$$

In the M-step, we re-estimate the model parameters:

$$p(s|e) = \frac{\sum_{w \in V} c(w, r_e)(1 - p(\theta_B|w, e))p(s|w, e)}{\sum_{s' \in S} \sum_{w \in V} c(w, r_e)(1 - p(\theta_B|w, e))p(s'|w, e)} \tag{6.15}$$

$$p(w|s) = \frac{\sum_{e \in E} c(w, r_e)(1 - p(\theta_B|w, e))p(s|w, e)}{\sum_{w' \in V} \sum_{e \in E} c(w', r_e)(1 - p(\theta_B|w', e))p(s|w', e)} \tag{6.16}$$

where V is the vocabulary set, $c(w, r_e)$ is the count of word w in r_e.

It is important to note that $p(s|e)$ should be non-zero only when $s \in S_e$. That is, when selecting attribute specifications to generate words, we can only select from those valid specifications for the particular entity e, and for different entities, this “feasible” set of specifications would generally be different (since products differ from each other on at least one attribute). We can ensure this property by initializing $p(s|w,e)$ in the following way:

$$p(s|w,e) = \begin{cases} 1/|S_e| & s \in S_e \\ o & otherwise \end{cases} \tag{6.17}$$

It is also worth noting that the background language model θ_B is a necessary component in the estimation in order to ensure the learned language models $p(w|s)$ are discriminative.

6.2.5 Maximum a Posterior (MAP) Estimation of the Mixture Model

One problem with the MLE is that the EM algorithm can be easily trapped in local maxima. To alleviate this issue, we need to provide some "guidance" to the estimator. Indeed, in practice, we usually have some prior knowledge of the language people use to describe certain attributes of entities. For instance, we expect to see words such as "cheap" and "expensive" when people talk about "price". If such prior knowledge can be incorporated into the estimator, it can guide the algorithm to find more accurate models.

To do this we employ Maximum a Posterior (MAP) estimator. We assume the knowledge is given in the same format as our model, and consider them as conjugate priors. Specifically, we use $p(w|\tilde{a})$ to denote the prior probability of using word w to describe attribute a, reflecting our belief in the language people use in general. We then maximize the posterior probability of the model. This is done by applying Dirichlet prior to Equation 6.16:

$$p(w|s) = \frac{\mu p(w|\tilde{a}_s) + \sum_{e \in E} c(w, r_e)(1 - p(\theta_B|w,e))p(s|w,e)}{\mu + \sum_{w' \in V} \sum_{e \in E} c(w', r_e)(1 - p(\theta_B|w',e))p(s|w',e)} \tag{6.18}$$

where a_s is the attribute (name) of spec s.

Generating Prior Knowledge. Clearly, it is infeasible to manually create all priors. To automatically discover such knowledge, we employ a co-occurrence analysis algorithm. More specifically, we analyze the co-occurrences of attribute names and all keywords in the text data. We use normalized pointwise mutual information (NPMI) [10] to compute the pseudo counts. NPMI for word w

and attribute a is defined as:

$$i_n(w,a) = \frac{\log \frac{p(w,a)}{p(w)p(a)}}{-\log p(w,a)} = \frac{\log \frac{N \cdot c(w,a)}{c(w)c(a)}}{\log \frac{N}{c(w,a)}} \tag{6.19}$$

where $c(w)$ and $c(a)$ are counts of sentences that word w and (the name of) attribute a appear, respectively; $c(w,a)$ is the count of sentences that w and a co-occur; N is the total number of sentences in the text data. It is worth noting that an attribute name could be a multi-word phrase. Therefore, we allow partial counts when dealing with occurrences. For instance, if a sentence contains word "screen" but not "size", we count it as 1/2 times of occurrence for attribute "screen size".

One nice property of NPMI is that its range of values is $[-1,+1]$. The upper bound and lower bound indicate perfect positive and negative correlation, respectively; value 0 indicates complete independence. We only keep words that have positive correlation to the attributes:

$$i'_n(w,a) = \begin{cases} i_n(w,a) & i_n(w,a) > 0 \\ o & otherwise \end{cases} \tag{6.20}$$

Then we compute $p(w|\tilde{a})$ by normalizing the above concurrence measure over all the words:

$$p(w|\tilde{a}) = \frac{i'_n(a,w)}{\sum_{w' \in V} i'_n(a,w')} \tag{6.21}$$

6.2.6 Smoothing and Interpolation

While the estimated $p(s|e)$ and $p(w|s)$ can be directly plugged into the general ranking model, the specification selection model cannot be used for "unseen" entities that are not associated with any text data, because the above estimation methods do not estimate $p(s|e)$ for them. One way to circumvent this issue is to "back off" to $p(s)$ for these entities, where $p(s)$ is computed as:

$$p(s) = \sum_{e \in E} p(s|e)p(e) \tag{6.22}$$

With the assumption of uniform distribution of $p(e)$, this back-off model can be easily computed based on the learned parameters.

An even better solution is to combine these estimates with the estimates discussed in Section 6.2.2 through interpolation. In general, such an interpolation would allow us to utilize all the available evidence and has been shown to be an effective strategy in our experiments.

Note that given the text data, we could also build a language model directly for each entity. Indeed, such a "blackbox" strategy could give us a query generation model (i.e., $p(w|e)$) directly, which we can use to compute the likelihood of a query q conditioned on entity e, thus allowing us to rank these entities that have text data. Unfortunately, the model estimated using this strategy can only be used to rank entities that have associated text data. In other words, this model is not *generalizable*. In contrast, our proposed mixture model above enables us to learn *generalizable* component models since $p(w|s)$ can be used for ranking any product entities with specification s. (Clearly such products do not have to have their own reviews/queries). However, for an entity with sufficient text data, the blackbox strategy may help alleviate the potential errors introduced by attempting to infer the latent specification selection process. Thus, a combination of this strategy with the mixture model estimation can potentially improve the robustness of the retrieval model, which is confirmed in our experiments.

6.2.7 Indexing and Search

With the estimated model parameters, we can use Equation 6.6 to rank product entities given any query. However, due to efficiency concern, we usually cannot afford to score every product entity for each query. Therefore, we need to build an indexing structure to allow the most promising candidate to be retrieved efficiently.

To do so, we first aggregate the specification selection model and the preference language model offline and compute a multinomial word distribution for each product entity:

$$p(w|e) = \sum_{s \in S} p(w|s)p(s|e) \tag{6.23}$$

Table 6.2: Entity retrieval models based on estimation with entity specs on Bestbuy dataset

	All Queries		Hard Queries	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
LM	0.542	0.574	0.579	0.636
QAM	0.546	0.577	0.588	0.642
AM-USS	0.476	0.520	0.528	0.592
AM-UPS§	0.525	0.564	0.567	0.614
AM-USS-LM§	0.533	0.577	0.570	0.640
AM-UPS-LM†‡§	**0.579**	**0.611**	**0.630**	**0.672**

Then we threshold $p(w|e)$ to get a set of index words for each entity:

$$W_e = \{w|p(w|e) > \sigma\}$$

That is, we drop the binding between a term and an entity if we are not confident enough, in order to avoid unnecessary computation wasted on non-promising candidates.

We then build inverted index for words and product entities, based on W_e and $p(w|e)$. In the meanwhile, we rewrite Equation 6.6 in the same way as in classic language modeling approach, leading to the following scoring function:

$$\begin{aligned} score(q,e) &= \sum_{w \in q \cap W_e} c(w,q) \log \big[1 + \frac{(1-\lambda)p(w|e)}{\lambda p(w|\theta_B)}\big] + \alpha_q \\ &\propto \sum_{w \in q \cap W_e} c(w,q) \log \big[1 + \frac{(1-\lambda)p(w|e)}{\lambda p(w|\theta_B)}\big] \end{aligned} \tag{6.24}$$

where

$$\alpha_q = \sum_{w \in V} c(w,q) \log \lambda p(w|\theta_B)$$

is a factor only dependent on query q. Therefore it does not affect ranking of product entities and can be omitted in scoring. With this setup, we can efficiently retrieve the candidates through the word-entity index and score them for ranking.

6.3 Experiments

6.3.1 Datasets and Evaluation Metrics

Evaluation of the proposed models is challenging since no previous work has studied our problem setup and as a result, there is no existing test collection that we can use for evaluation. We thus had to construct our own datasets.

For this purpose we developed two different evaluation datasets. The first dataset consists of a full crawl of the "Laptop & Netbook Computers" category of *Bestbuy.com*[1], a popular e-commerce website for electronics. Our crawl includes all the specs and reviews of each laptop. There are in total 864 laptops in the database; on average, each entity has 44 specifications. Among these laptops, only 260 of them have user reviews. For evaluation we construct a query set with the following procedure. We first extract a set of simple queries by sampling "laptop" queries from a commercial query log. We filter these queries so that each query contains a well written descriptor on a single attribute. Examples in this query set are "thin laptop", "quad core laptop". We then use the strategy introduced by Ganesan and Zhai [31] to develop a relatively harder query set, simulating long and comprehensive preference queries that we expect users to enter. This is done by randomly combining multiple descriptors from the simple queries. As discussed in [31], because current product search engines cannot support such queries very well, it is difficult to find them in the search log. Thus it is necessary to simulate the query set in such a manner. An example of "difficult" query is "large screen quad core backlit keyboard laptop". To quantitively evaluate the performance of product entity retrieval, we pool together the retrieval results from several baseline models and our proposed models. A well trained industrial annotator is then asked to label the pool of candidates with 5 levels of relevance. On average, 60 entities are judged for each query. In total we obtained 40 queries with annotations for this dataset. On average there are 2.8 keywords per query, and 3.8 keywords per query for the hard queries.

The second dataset consists of several major categories in electronics from another popular e-commerce website, *Walmart.com*[2]. This includes the categories of *laptop*, *camera*, *camcorder*, *TV* and *ebook reader*. In total, the database consists of 1066 entities; on average, each entity has 14.0

[1]http://www.bestbuy.com

[2]http://www.walmart.com

Table 6.3: Entity retrieval models based on estimation with entity specs and reviews on Bestbuy dataset

	All Queries		Hard Queries	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
LM	0.680	0.706	0.592	0.621
QAM	0.697	0.726	0.626	0.663
AM-Base	0.591	0.638	0.491	0.558
AM-MLE§	0.686	0.721	0.625	0.653
AM-MAP§	0.697	0.721	0.625	0.676
AM-Base-UPS §	0.662	0.718	0.587	0.651
AM-MLE-UPS†§	0.698	0.736	0.637	0.682
AM-MAP-UPS†§	0.711	0.755	0.666	0.710
AM-Base-LM§	0.685	0.710	0.589	0.623
AM-MLE-LM†§	0.702	0.730	0.633	0.655
AM-MAP-LM†§	0.705	0.734	0.626	0.668
AM-Base-UPS-LM†‡§	0.722	0.761	0.661	0.705
AM-MLE-UPS-LM†§	0.708	0.752	0.651	0.706
AM-MAP-UPS-LM†‡§	**0.729**	**0.774**	**0.684**	**0.734**

specifications. Each product entity is associated with a set of queries by thresholding the number of clicks observed in a search log accumulated for over a year. Queries are randomly sampled from commercial queries in Walmart search engine, and are annotated in the same way as the first dataset. In total we obtain 425 queries with annotations. On average, each query has 2.4 keywords.

The two datasets are referred to as the *Bestbuy* dataset and the *Walmart* dataset, and they are used to evaluate the effectiveness of our model estimated with review data and search log data, respectively. The evaluation is based on the metric of Normalized Discounted Cumulative Gain (NDCG). We use cutoff at 5, 10 as our primary metrics. Since the first impression is usually delivered by the top results, NDCG@5 measures the immediate satisfaction to the users' search. On the other hand, NDCG@10 is more commonly interpreted as the first page satisfaction.

6.3.2 Experiment Results

Table 6.2 shows the comparison between the baseline models and our proposed probabilistic models that are estimated solely based on entity specs data. LM is the baseline language model estimated by treating all the specs of each entity as a single document. Language model is a well established model for document retrieval [54, 80]. In this work, we use query likelihood language model with

Jelinek-mercer smoothing [80]. Another baseline method we compare with is the Query Aspect Modeling (QAM) method proposed by Ganesan and Zhai [31]. In this method, the search query is assumed to be pre-segmented into multiple preference queries, each covering one aspect of the product. Then the method evaluates each of these preference queries separately and combine the results to obtain the ranking for the original query. The best combining method, i.e. average score method, is implemented in our evaluation (denoted QAM). All the other models tested are attribute-value-based models (AM) under the same general framework proposed in this work. The spec preference language models $p(w|s)$ of all AM models are estimated with the Attribute-Value Text method (see Section 6.2.2). AM-USS uses Uniform Specification Selection as the estimate of specification selection model (see Section 6.2.2). We consider AM-USS as another baseline system (where the estimations are not optimized). These baseline models largely represent the state-of-the-art methods for supporting keyword queries in probabilistic databases based on keyword matching.

AM-UPS refines the estimation of specification selection model by adopting the Uniform Prior Selection estimate (Section 6.2.2). AM-Base-LM and AM-UPS-LM are the weighted interpolation models with AM-Base and LM, AM-UPS and LM, respectively. The best performance on each metric is shown in bold font. We use †, ‡ and § to mark the models that show statistically significant improvement on all evaluation metrics over LM, QAM and AM-Base, respectively.

From the table we can see that when used alone, LM is a relatively strong baseline compared to AM-USS. QAM slightly improves over LM, especially on hard queries. This is in accordance with the findings in [31]. AM-UPS significantly improves over AM-USS with the more accurate estimation for specification selection. Although AM-UPS used alone does not show improvement over LM and QAM, the interpolated model AM-UPS-LM significantly outperforms all three baseline methods. This verifies that the proposed models and the traditional language modeling approach are complementary to each other in effect. While LM method estimates accurate language models for entities with sufficient data, our model provides the generalizability with language models estimated on the attribute level.

Table 6.3 shows the comparison of different methods based on product review data. Here the LM method is estimated with both specs and review data. AM-Base is another baseline method, where each spec preference language model is estimated using MLE on a long review document constructed by pooling together all the reviews of entities with this spec. In fact, this is just the result of the first iteration of our EM estimation algorithm. AM-MLE and AM-MAP are the MLE estimate (Section 6.2.4) and MAP estimate (Section 6.2.5) of the query generation model, respectively. The $*$-UPS models use UPS estimate (Section 6.2.2) to interpolate with the corresponding specification selection model. The $*$-LM models are models interpolated with general LM. Again, we use bold font to show the best performance for each metric. †, ‡ and $\S$ are used to mark the models that have statistically significant improvement on all evaluation metrics over LM, QAM and AM-Base, respectively.

From the table, we can see that LM is a stronger baseline than AM-Base, and QAM outperforms LM on the hard queries, which are in accordance with previous findings in Table 6.2. Similarly, we also see AM-Base-LM slightly outperforms LM. This shows that even though sometimes the estimation is not good enough to improve the baseline language model by itself, it still captures a different level of relevance signals that could be used to alleviate the impact of data sparsity.

The proposed estimation methods, i.e. AM-MLE and AM-MAP, improve over the simple estimation method (AM-Base) significantly. This verifies the effectiveness of the MLE and MAP estimation. We can also see that both MLE and MAP outperforms ML slightly, especial for the hard queries. Between the two estimation methods, MAP estimation performs slightly better than MLE. This validates the positive effect of incorporating prior knowledge in model estimation.

The UPS interpolated models ($*$-UPS) all show significant performance boost from the original models. This is also in accordance with our previous finding in Table 6.2. These findings confirm that the "IDF" effect in specification selection is indeed positively correlated with real users' preferences.

We see improvements in almost all LM interpolated models ($*$-LM) (over their base models). The best performance is achieved by AM-MAP-UPS-LM. The results show that with the use of advanced estimates and product review data, we can train effective models to capture the attribute level of relevance. Compared with the entity language model which models a coarse level of relevance, our models are superior. By interpolating the two types of models, we can achieve even more robust and accurate estimates.

It is worth noting that in the evaluation, the "hard" queries do not necessarily have lower NDCG values compared to the "easy" queries, as NDCG is a metric normalized by the ideal DCG value on each query.

Table 6.4: Entity retrieval models based on estimation with entity specs on Walmart dataset

	NDCG@5	NDCG@10
LM	0.384	0.303
AM-USS	0.381	0.299
AM-UPS	0.392	0.308
AM-USS-LM	0.386	0.306
AM-UPS-LM	**0.393**	**0.310**

Table 6.5: Entity retrieval models based on estimation with entity specs and logged queries on Walmart dataset

	NDCG@5	NDCG@10
LM	0.501	0.392
AM-Base	0.507	0.394
AM-Base-UPS	0.509	0.401
AM-MAP	0.509	0.401
AM-MAP-UPS	0.515	0.406
AM-MAP-UPS-LM†§	**0.518**	**0.411**

In Table 6.4 and Table 6.5 we run similar experiments with the Walmart dataset. Again, the experiments confirm that UPS is a better estimate for specification selection model $p(s|e)$ as compared with USS. In general, the use of text data clearly improves the search performance. The experiments also confirm that the mixture model (AM-MAP-$*$) is a superior method for utilizing text data, as it outperforms all baseline systems including LM, AM-Base and AM-Base-UPS. We also observe that the combination of our model and language model would always lead to a performance boost. As in consistent with the evaluation on the Bestbuy dataset, the best performance is achieved by AM-MAP-UPS-LM.

It is worth mentioning that although in this work we separated the product type detection from product preference matching and focused on modeling the latter component, the model we use can actually provide a natural solution to product type detection, by treating product type as a special attribute in product specification. Indeed, this method is used in the experiments with Walmart dataset and achieved good performance.

6.3.3 Search Efficiency

As discussed in Section 6.2.7, we can build index to ensure the efficiency of the proposed models by computing an indexing word set for each entity using threshold σ. In this section, we study the impact of the threshold on search efficiency and search accuracy. In Figure 6.1 we plot the average running time (milliseconds) of AM-MAP-UPS model for different threshold σ. For comparison we also plot the average running time of language model (LM). Both models are ran on a single machine with Intel core i7 2.7GHZ processor and 8GB Ram.

We can see that our model is much more efficient than language modeling approach in general. This is because the learned model is able to capture the most discriminative topical words for each entity and demote/discard the meaningless general words. We also observe that both the running time and the retrieval performance (in Figure 6.2) stabilized after $\sigma < 1e-4$ This indicates we have included most of the word-entity associations produced by our model.

6.4 Other Applications Using the Probabilistic Query Model

Although the query generation model is primarily proposed for entity retrieval, it provides a general probabilistic framework which could also lead to many useful applications. In this section, we demonstrate the usability of our model by exploring two novel applications: facet generation and review annotation. We show the parameters in our model can be easily adapted for these tasks and achieve very encouraging results.

6.4.1 Facet Generation

Facet Generation is an important application for e-comme- rce websites. Its purpose is to engage users and help them clarify their preferences for the product search engine. To do this, the result page of a product search system usually provides a list of facets, i.e. attributes of products, on a sidebar by the search results. Traditionally, the facet list is generated by hiring experienced analysts to manually select a subset of attributes. The task is very laborious as we need to generate facets for each category of products. Moreover, the manually generated facets do not necessarily match users' interests and cannot reflect the change of such interests over time.

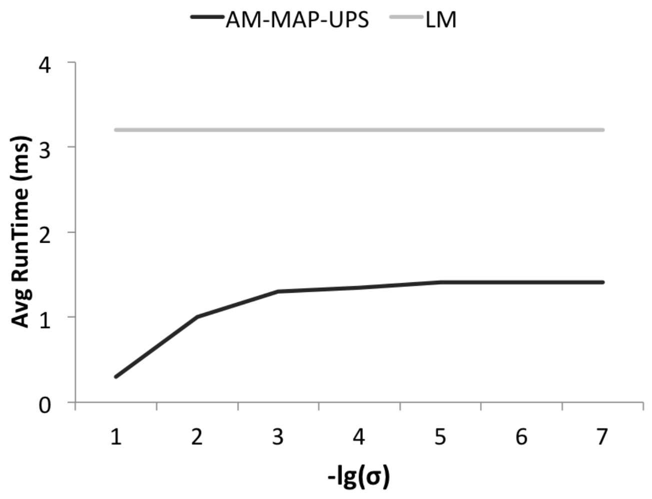


Figure 6.1: Average search time for different threshold σ

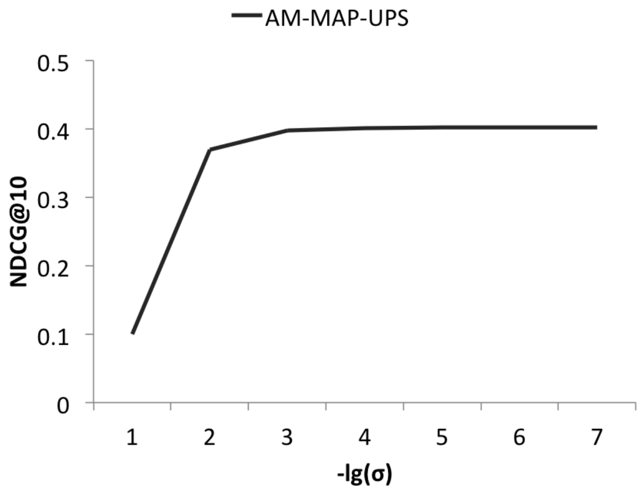


Figure 6.2: NDCG@10 for different threshold σ

Table 6.6: Query independent facet generation for laptop category

Most Popular Facets	Least Popular Facets
Graphics Card	ENERGY STAR Qualified
Pointing Device	Multi-Carrier
Audio	Built-in Webcam
Hard Drive Type	Product Height
Brand	Wi-Fi Built In
Computer Hard Drive Size	System Bus
Operating System	BBY Software Installer
Processor	Green Compliance
Video Memory	Color Category
Battery Life	Touchscreen

In this section we show how we can use the parameters of our proposed model as a building block to automatically generate the facets based on popular interests. Besides the traditional (query independent) facet generation, we further show that our model can also be used for generating facets tailored to the search intent of each query.

6.4.2 Query Independent Facet Generation

Let us use $p(a)$ to denote the probability that a user is interested in attribute a when searching for products. $p(a)$ can be computed by summing over the marginal probability $p(s)$ of all the specifications defined on attribute a:

$$p(a) = \sum_{s \in \{s|a_s=a\}} p(s) = \sum_{s \in \{s|a_s=a\}} \sum_{e \in E} p(s|e)p(e) \tag{6.25}$$

With the assumption of uniform $p(e)$, we can easily compute $p(a)$ from our learned models. In Table 6.6 we show the most and least popular attributes for the laptop category. The results are intuitively very meaningful, as the attributes ranked at the top of the facet list (e.g. *Graphics Card*, *Audio*, *Hard Drive*, *Brand*) are mostly the commonly concerned aspects when users shop for laptop computers, while the facets at the bottom are product features that do not affect buying decisions very much.

Table 6.7: Query specific facet generation

q_1:	*surround sound laptop*	
a_1:	Audio	$\log p = -6.5$
q_2:	*gaming laptop*	
a_1:	Graphics Card	$\log p = -7.3$
a_2:	Gaming Series	$\log p = -11.2$
q_3:	*ssd large screen laptop*	
a_1:	Computer Hard Drive Size	$\log p = -12.5$
a_2:	Screen Size	$\log p = -18.1$
q_4:	*quad core blu ray laptop*	
a_1:	Graphics Card	$\log p = -11.7$
a_2:	Processor	$\log p = -14.2$
a_3:	Blu-ray Player	$\log p = -14.4$
a_4:	Optical Drive	$\log p = -17.3$

6.4.3 Query Specific Facet Generation

The facets are useful for engaging users and helping users refine their shopping preferences, but the static facets are not very effective as they cannot capture the user intentions in the search query. To solve this problem, we further study query specific facet generation.

Let $p(a|q)$ denote the probability that users are concerned about attribute a when issuing query q. $p(a|q)$ can be computed as:

$$\begin{aligned} p(a|q) &= \sum_{s\in\{s|a_s=a\}} p(s|q) \\ &\propto \sum_{s\in\{s|a_s=a\}} p(s)p(q|s) \\ &\propto \sum_{s\in\{s|a_s=a\}} p(s)\prod_{w\in q}[\lambda p(w|\theta_B)+(1-\lambda)p(w|s)] \end{aligned} \tag{6.26}$$

Using $p(a|q)$ as a scoring function, we can dynamically discover the important facets for each query. Table 6.7 shows the top most suggestions with their probabilities ($\log p(a|q)$) for several example queries. In general we find our model performs very well, especially for short queries. For all the queries in the examples, the most related facet (i.e. attribute) is ranked at the top or the second position; for queries with multiple focuses (query q_3, q_4), all the related facets are successfully discovered.

Table 6.8: Sample review annotation for HP - ENVY Spectre Ultrabook

t_1:	*Excellent display*
s:	Graphics Card: Intel HD Graphics 3000
t_2 :	*the best sound on the planet for a notebook*
s:	Audio: Beats Audio
t_3:	*very fast SSD hard drive*
s:	Hard Drive Type: SSD (Solid State Drive)
t_4:	*WiFi is super fast*
s:	Networking: Built-in 10/100/1000 Gigabit
t_5:	*and the weight is perfect for long use on a trip*
s:	Pointing Device: Multitouch Imagepad
t_6:	*Touchpad is a bit skiddish*
s:	Pointing Device: Multitouch Imagepad

6.4.4 Review Annotation

Another interesting application we explore is review annotation. In this task, we want to automatically detect what feature(s) of the product each review sentence is commenting on. This is done by matching the review sentences with the specs of the corresponding product. With our learned model, we use the conditional probability of $p(s|t,e)$ to rank each sentence t in the review of entity e:

$$\begin{aligned} p(s|t,e) &\propto p(s|e)p(t|s,e) \\ &\propto p(s|e)\prod_{w\in t}[\lambda p(w|\theta_B)+(1-\lambda)p(w|s)] \end{aligned} \tag{6.27}$$

Table 6.8 shows an example of annotated review. Such annotations can be used to better organize the reviews and generate useful opinion summarizations. We can see that despite some mistakes, the model works reasonably well for the task. Within six randomly selected examples, we found that the method performs well on five of them (sentence t_1, t_2, t_3, t_4 and t_6). A mistake was made on sentence t_5, where the intent of the description is mostly about “Weight”, but our algorithm tagged it with “Pointing Device”. The reasons are that the sentence is relatively long containing lots of distractive terms and “Pointing Device” is a generally popular attribute which has a high probability of getting selected. In general, we observe the model works better annotating short review sentences than long sentences. One potential solution to this problem may be to further segment a long sentence into shorter clauses and apply the annotation algorithm to each clause.

CHAPTER 7

DISCOVERING COORDINATED SHOPPING INTENT IN PRODUCT SEARCH

7.1 Introduction

In Chapter 6, I proposed and studied query intent modeling with data level representations. Although the proposed method is effective in matching query intent into entity structure information and improving search accuracy, it does not provide much insight in understanding users' shopping preferences, as the intent analysis is performed on a per query basis. In product search, different users may use the same query to express different intent, or use different queries to express the same intent. It is important to explicitly model such ambiguities and represent users' shopping intent in product search at a higher conceptual level. In this chapter, I study the modeling and automatically discovering of users' shopping intent in product search.

Clearly, understanding such preferences is very useful for improving the product search engine. Not only will it improve search accuracy, but it can also largely benefit the search experience as a whole. For instance, when the user query is too specific, matching none or few products (e.g. directly copied from product titles elsewhere), we can provide recommendations of similar products based on the inferred preferences from the query. On the other hand, when a user query is too broad or ambiguous, based on the potential intent of the query, we can better organize search results so that representative products of different intents are displayed on top.

To achieve these benefits, I study the modeling and automatic discovery of user's shopping intent in product search. Because intent is not directly observable, we can only infer it from user's behaviors such as search engagement. However, such inference is no easy task. First, we cannot determine user's intent solely based on queries. Users with the same query could eventually buy very different products, and different queries could be used for the same/similar intent. Second, it is also difficult to infer the intent based on the target products. Users who purchased the same product

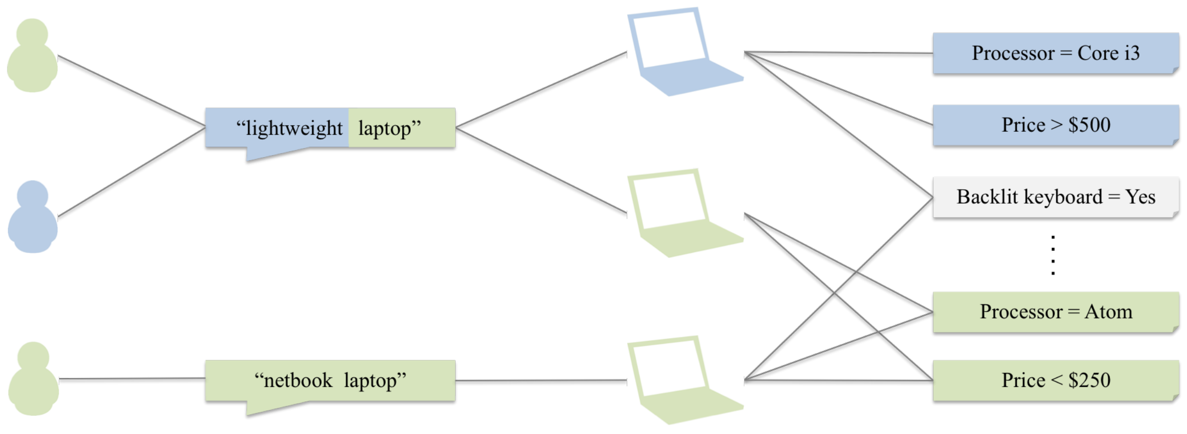


Figure 7.1: Example of Search Intent

could be looking for different aspects of the product, and those who look for same properties may end up buying different items. As an illustrating example, in Figure 7.1, the two users with the query "lightweight laptop" are looking for very different products: the first user is looking for an inexpensive lightweight laptop used for web browsing (*Processor=Atom & Price¡$250*); the second user is looking for a portable but high performance laptop with more capabilities (*Processor=Core i3*), and price is not the primary concern (*Price¿$500*). The third user, although having a similar preference as the first user, ended up buying a different laptop.

Due to these difficulties, it is not easy to automatically discover user intent from their search behaviors; and although the subject has been studied in the literature of information retrieval, no existing work has explored search intent modeling in product search, where the fundamental units are very different from those of document retrieval or Web search. In this work, we study the problem of modeling and discovery of shopping intent in product search. First, we solve the practical challenge of intent representation. As we see in the previous examples (e.g. Figure 7.1), it is difficult to capture user's shopping intent based on information on the query and product level. To achieve an effective representation, we need to make use of more detailed information. Therefore, a coordinated intent representation is proposed, where each intent is characterized together by the targeted product features and the corresponding query terms. More specifically, three distributions are used to represent a intent: an attribute distribution capturing the attributes that are important to the intent (e.g. *Processor*), a feature value distribution for each attribute capturing the preference on the attribute (e.g. *Processor=Atom*), and a word distribution capturing the choice of search terms (e.g. "netbook").

We then propose a joint mixture model for automatically discovering the coordinated intent representation in search logs. In this model, a basic generative process for the user's search engagements is assumed as follows. First, the user chooses an intent from a mixture of search intent. The user then composes a query according to the word distribution of the chosen intent model. Upon receiving the search results, the user decides which product to further engage with by generating the most desired products by sampling the features on each of its attributes. Particularly, the user first selects an attribute based on the attribute preference distribution, and then samples a value based on the feature value distribution of the attribute. To estimate the joint mixture model, an expectation-maximization (EM) algorithm is employed. In general, we can use all kinds of engagement signals in product search (e.g. clicks and ordering events), and user engagement detection is naturally an interesting preceding step of our work. Since in this work we mainly focus on modeling shopping intent, we use a simple detector which consider every clicking behavior as positive engagement. Nonetheless, we want to stress that our model is general and can be naturally applied on top of more complex engagement detectors.

We empirically examine the fitted models on several large domains in online shopping. We show that, the proposed model can be used as a fundamental building block for important analytical tasks such as query ambiguity analysis. Compared to traditional methods which mainly capture the document/category level ambiguity, the new analysis based on our model is more effective as it captures the intent level ambiguity that are more intrinsic to users' behaviors. We then incorporate the intent models in product search. We show that the models can be easily adapted in the language modeling framework. With the intent model, we can achieve significant improvement in search accuracy. To further demonstrate the usefulness, we use the intent model to compute an additional similarity signal for product recommendation. Therefore, different products could be related although they do not share exact same features (e.g. *Processor=Atom vs. Processor=Opteron X*). Experiment results confirm the effectiveness of our model in both product search and product recommendation. Although not studied in this work, the intent model has potential in improving many other related applications such as query suggestion and search result diversification.

Our contribution of this work can be summarized as follows:

- We recognize the importance of modeling user intent from product search. We proposed a coordinated representation to capture such intent.
- We propose a joint mixture model for automatically discovering the coordinated user intent from search engagement. We develop an EM algorithm for the estimation of the model.
- We apply the proposed intent representation and modeling technique on a large domain of products and examine the intent models we discover.
- Beside improving the accuracy of product search, we demonstrate the usefulness of the proposed intent modeling technique in a variety of applications, including query ambiguity analysis and product recommendation.

Although in this study we primarily experiment with click engagements for discovering user's shopping intent, the technique we use for modeling intent is general and can naturally incorporate other types of user engagements. In the future, we plan to enhance the intent discovery by taking into account of all types of engagements. We also plan to further improve our model by expanding the span of search engagements we use, e.g. utilizing the session information.

7.2 Discovering Coordinated Shopping Intent in Product Search

In product search, a search intent is essentially a stereotype of users' shopping preference embodied in search behavior. Typically, such behavior includes query formulation and engagement with search results, such as clicks and other further actions. For example, users who search for "lightweight laptop" might be mainly categorized as those looking for convenient netbooks for personal use and those looking for business class portable laptops. Obviously, being able to understand such stereotypes would greatly benefit the search experience. It can not only improve search accuracy, but also enables a variety of applications such as recommendation, diversification and personalization. However, the problem is challenging as such stereotypes (i.e. shopping intent) are usually not directly observable. Although some techniques in document understanding could be adapted for the task, the results are usually suboptimal. In particular, the topic modeling approach, when applied to engagement data, could discover meaningful feature clusters. While these clusters can be regarded as shopping intent as they group together the common product features in users' engagements, their data structure is not specifically designed for intent modeling and they do not contain sufficient information to comprehensively represent shopping intent.

In this work, we propose to study modeling and automatically discovering shopping intent in product search from search engagement logs. To the best of our knowledge, no existing work has studied this problem before. One major obstacle in modeling shopping intent in product search is intent representation. Since intent is not observable, it is important to develop an appropriate representation. Ideally, a good representation should not only contain sufficient information to uniquely represent a shopping intent, but also be easy to interpret. Following this line of thought and also taking into consideration the structures of product information, we propose a coordinated intent representation, where each intent is characterized collectively by the target product features and corresponding search terms.

With the coordinated representation, we propose a joint mixture model, which models the generative process of the observations (i.e. user engagements) based on the intent models. We then employ an EM algorithm for the estimation of the joint mixture model, so that the intent models can be discovered automatically by fitting to a real log of user engagements in product search.

Notations. In the convenience of discussion, we introduce a set of notations and will use them throughout the rest of the section. We use E to denote the set of product entities in the database. Each product entity is represented by a feature vector on a predefined attribute space A. For a product $e \in E$, the value on attribute $a \in A$ is denoted by $v_{e,a}$. We also use S to denote a set of sessions acquired from product search logs, where each session $s \in S$ consists of a query q_s and a set of relevant product entities E_s. Each query q is a set of keywords, i.e. $q = \{w | w \in W\}$, where W is the word vocabulary.

Note that our definition of session is different from the typical definition of session, as each session contains only one query. In case multiple queries are present during a session, we split them into separate ones. The reason for this is that we can safely assume that a user's shopping intent remains invariant during the search of a single query. The same assumption may not apply for multiple query sessions because users may change minds on what they are shopping abruptly and alternately during a time period. Despite this simplification in our modeling framework, the method we propose in this work can be easily adapted to work with multi-query sessions when reliable session detecting method is available. The discussion of such methods falls out of the scope of this work.

In the rest of the section, we first review the topic modeling approach. We show how it can be adapted for intent discovery and study the limitations of the method. We then introduce the novel coordinated intent representation, followed by the joint mixture model for discovering shopping intent and the corresponding estimation method.

7.2.1 Topic Modeling Approach Adapted for Intent Discovery

With loss of generality, we use probabilistic latent semantic analysis (PLSA) [36] to show how topic modeling approach can be adapted to discover shopping intent in product search. PLSA was initially proposed as a method to automatically discover the salient topics from a set of text documents. The method essentially assumes a document generation process where a user iteratively selects a topic and sample a word from the topic. PLSA has been widely used in document understanding and document similarity computation. To adapt PLSA for the purpose of discovering shopping intents, we can conceptually transform the product engagement data into text document set in the following way. First, we consider each instance of product feature as a word. That is, we discard the attribute value structure of entities and simply regard each attribute-value pair as a word. For example, *Processor=Atom* and *Processor=i3* are different words, and they are as different as *Processor=Atom* and *Backlit Keyboard=Yes*. Second, we treat all the products that a user engages with in a search session as a single document. By performing such combination, product features capturing the same intent will “co-occur” with each other in the same “document”. Since topic modeling approach rely on co-occurrence information, we expect the product features that can coherently represent unique shopping intent will emerge.

Formally, let T be a set of feature clusters and each $t \in T$ represents a unique shopping intent as a multinomial distribution over product features. This method estimates the parameters of T by maximizing the following likelihood function:

$$\mathcal{L} = \sum_{s \in S} \sum_{e \in E_s} \sum_{a \in A} \log \left((1 - \lambda) \sum_{t \in T} p(v_{e,a}|t_i) p(t|s) + \lambda p(v_{e,a}|t_G) \right)$$

where t_G is a generic intent model and λ is an interpolation parameter controlling the amount of content generated from the generic intent, s.t. $0 \leq \lambda \leq 1$ Typically, t_G can be estimated by normalizing the counts of words (i.e. product features) under the entire collection. By setting λ to a relative large value, we can ensure the discovered intent models are discriminative.

However, this method suffers from several important issues, making it less ideal for modeling and discovering shopping intent. First, although the uses an intuitive representation of shopping intent using the product features, it fails to take into consideration the structure of products. In this model, product features are simply regarded as "words" and the relations between features are overlooked. As a result, each product feature competes not only with the features from the same attribute (e.g. *Processor=Atom* and *Processor=i3*), but also with the features from other attributes (e.g. *Processor=Atom* and *Backlit Keyboard=Yes*). As a result, the model cannot achieve accurate estimations of feature probabilities, hence the intent models are not accurate. Also, because of the disregard of product structures, the discovered intent models are difficult to interpret. For instance, we cannot easily find answer to the question "which product attribute is the most concerned for users who enter a particular query?" with the current intent representation.

Another problem with the model is that it does not take into consideration the search queries in estimating the shopping intent models. Consequently, the intent models estimated in this way may not comply with the queries users use in product search. As pointed out in previous discussions, due to the ambiguity in user behavior, neither product or search query can be used to solely represent shopping intent. Failing to incorporate the search queries makes the model less accurate in discovering shopping intent. Another drawback due to this inconsideration is that the discovered shopping intent does not contain sufficient information to interpret, i.e. it misses the terminologies users use to describe and search for particular intent.

Finally, the generative process in this model allows each product feature in a session to be generated using different intent models by choosing separately from the intent mixture. Because of this modeling assumption, the consistency of intent within single sessions cannot be guaranteed, hence affecting the quality of the discovered intent models.

In the following discussions, we will introduce a novel coordinated intent representation and a joint mixture model for discovering shopping intent from product search. We will show that all the problems suffered by the topic modeling approach will be addressed in the proposed model.

7.2.2 Coordinated Representation of Shopping Intent in Product Search

In product search, users mainly interact with search queries and products. Both of them contain important information in characterizing shopping intent. The queries capture the users' language in describing the intent and the products capture the target of the intent. In order to make the discovered shopping intent useful in real applications, the intent models need to be able to explain both queries and products in search sessions. Because the queries are in free text and the products are represented by well structured feature information, they need to be modeled separately.

Based on these considerations, we propose a coordinated intent representation to compressively capture user's shopping intent. Essentially, we use three distributions to characterize a shopping intent. Formally, let I be a set of intent models, let k be the size of I and I_i be the ith intent in I, where $1 \leq i \leq k$. We then have

$$I_i = \{\psi_i, \Gamma_i, \Theta_i\}$$

where ψ_i is a multinomial distribution over search terms W, capturing the importance of the terms in describing the intent. For example, given a shopping intent looking for *portable TV set*, words such as "potable" and "compact" will likely have high probabilities, while words such as "signal" and "wifi" may have relatively low probabilities. $\Gamma_i = \{\gamma_{i,a} | a \in A\}$ is a set of Bernoulli distributions over all the attributes of products A. Each $\gamma_{i,a}$ captures how likely the corresponding attribute is concerned in this intent (vs. the generic intent). A small $\gamma_{i,a}$ value corresponds to a high probability that the user want to impose a preference on the attribute, rather than complying with the popular preference. For example, in the *portable TV set* intent, the *Size* attribute will have a very high probability, possibly close to 1; whereas other attributes like *Refresh Rate* and *Warranty* will have low probabilities, possibly close to 0. The third parameter Θ_i is a set of multinomial distributions, defined on the product attributes, too. Each $\theta_{i,a} \in \Theta_i$ is multinomial distribution over product features, capturing the preference on the given attribute. Taking the previous example again, the *Size* attribute will have high probabilities on the small sizes such as *Size=10"* and *Size=18"*.

With this intent representation, we can effectively capture all the most useful information in a shopping intent. Compared to the topic modeling approach, this representation not only makes the intern models more interpretable, but also makes it easier to apply the models to real applications, hence making it more useful. In the next section, we will introduce how such intent models can be automatically discovered from user engagements in product search logs.

7.2.3 Joint Mixture Model for Discovering Shopping Intent

With the coordinated intent representation, we propose a novel joint mixture model to automatically discover shopping intent in product search logs. In this model, the observations of user engagements in search sessions are generated using the following generative process. At the beginning of a session s, the user chooses an intent I_i by sampling from a mixture of intent. After the intent is decided, the user will use the corresponding intent model $\{\psi_i, \Gamma_i, \Theta_i\}$ to generate all the observations in the session, i.e. the query q_s and the engaged product set E_s. To do so, the user first selects a query q_s by repeatedly sampling a word from the word distribution ψ_i. Although a specific intent model has been selected, the user still has the freedom to choose to either generate a topic specific word or a more general word from the generic word distribution ψ_G. Second, each product $e \in E_s$ is generated in the following way. For each attribute, the user first decides whether he/she has strong preferences on the features of the attribute, versus just wants to go with popular choices. Essentially, the user chooses between the selected intent model I_i or the generic intent model I_G by sampling a binary variable from the Bernoulli distribution $\gamma_{i,a}$. Based on the decision, the user generates the value of the feature $v_{e,a}$ by sampling from the corresponding value distribution ($\theta_{i,a}$ or $\theta_{G,a}$).

The joint mixture model has many advantages. First, it is consistent with our intuition that user's shopping intent remains unchanged throughout a single search session. Second, it takes into consideration both the query and the products in a user engaged session. Therefore, the intent models discovered will be coherent and less likely to be affected by users' behavioral ambiguity in choosing queries and engaging with products. Finally, we incorporate the structure of product in modeling shopping intent. The feature generation is modeled separately on different attributes. This ensures that the estimation of one product feature will not affect other features on different attributes. The intent models is therefore more meaningful and interpretable. As an illustration, consider a session with a query "kitchen TV" and a clicked product featuring *Size=7"* and *Color=Black*. To generate this observation, first, an intent is sampled from the mixture of intent based given this session, e.g. the *portable TV set* intent. Given this intent, query terms "kitchen" and "TV" are generated according to the word distribution of the *portable TV set* intent. To generate the observed product, we generate the feature value on each individual attribute. We first generate a indicator variable z_G to select which model to use in generating the value (i.e. the selected intent model vs. the generic intent model). For *Size* attribute, the indicator variable is likely 1 based on the selected intent model; for the *Color* attribute, the indicator variable could be zero. Based on these two indicator variables, we finally generate the value *Size=7"* and *Color=black* according to the selected intent model and the generic intent model, respectively.

7.2.4 Model Estimation

Given a set of observed sessions S, we can write down the data likelihood based on the joint mixture model:

$$\begin{aligned}\mathcal{L} = \sum_{s\in S} \log \Bigg[\sum_{i=1}^{k} p(I_i|s) \prod_{w\in q_s} \Big((1-\lambda)p(w|\psi_i) + \lambda p(w|\psi_G) \Big) \\ \cdot \prod_{e\in E_s} \prod_{a\in A} \Big((1-\gamma_{i,a})p(v_{e,a}|\theta_{i,a}) + \gamma_{i,a} p(v_{e,a}|\theta_{G,a}) \Big) \Bigg]\end{aligned} \tag{7.1}$$

To estimate the model parameters, we want to maximize this likelihood of data. To do so we employ an expectation-maximization (EM) algorithm. The algorithm maximizes the likelihood by iteratively applying an expectation step (E-step) and a maximization step (M-step).

In the E-step, the algorithms computes the expectation of the likelihood by inferring the identity variables:

$$p(z_i = 1|s) = \frac{\tilde{p}(z_i = 1|s)}{\sum_{i'=1}^{k} \tilde{p}(z_{i'} = 1|s)} \tag{7.2}$$

$$p(z_G = 1|s, e, a, I_i) = \frac{\gamma_{i,a} p(v_{e,a}|\theta_{G,a})}{(1-\gamma_{i,a})p(v_{e,a}|\theta_i) + \gamma_{i,a} p(v_{e,a}|\theta_{G,a})} \tag{7.3}$$

$$p(\xi_G = 1|s, w, I_i) = \frac{\lambda p(w|\psi_G)}{(1-\lambda)p(w|\psi_i) + \lambda p(w|\psi_G)} \tag{7.4}$$

where

$$\begin{aligned}\tilde{p}(z_i = 1|s) =& p(I_i|s) \prod_{w\in q_s} \Big((1-\lambda)p(w|\psi_i) + \lambda p(w|\psi_G) \Big) \\ &\cdot \prod_{e\in E_s} \prod_{a\in A} \Big((1-\gamma_{i,a})p(v_{e,a}|\theta_i) + \gamma_{i,a} p(v_{e,a}|\theta_{G,a}) \Big)\end{aligned} \tag{7.5}$$

Here ψ_G is a generic word preference distribution and $\theta_{G,a}$ is a generic value preference distributions on attribute a. Both can be estimated using the entire collection data. z_i, z_G and ξ_G are indicator variables. If $z_i = 1$, the ith intent, i.e. I_i, will be selected. If $z_G = 1$, the generic intent model will be used to generate the feature value; if $z_G = 0$, the selected intent model I_i will be used instead. The same semantic is applied to ξ_G for generating words.

In the M-Step, the algorithm reestimate the model parameters based on the values of the latent variables inferred in the previous iteration:

$$p(v|\theta_{i,a}) = \frac{\sum_{s \in S} \sum_{e \in E_s} \delta(v_{e,a}, v)(1 - p(z_G = 1|s, e, a, I_i))p(z_i = 1|s)}{\sum_{s \in S} \sum_{e \in E_s} (1 - p(z_G = 1|s, e, a, I_i))p(z_i = 1|s)} \tag{7.6}$$

$$\gamma_{i,a} = \frac{\sum_{s \in S} p(z_i = 1|s) \sum_{e \in E_s} p(z_G = 1|s, e, a, I_i)}{\sum_{s \in S} p(z_i = 1|s)|E_s|} \tag{7.7}$$

$$p(w|\psi_i) = \frac{\sum_{s \in S} c(w, s)(1 - p(\xi_G = 1|s, w, I_i))p(z_i = 1|s)}{\sum_{w' \in V} \sum_{s \in S} c(w', s)(1 - p(\xi_G = 1|s, w', I_i))p(z_i = 1|s)} \tag{7.8}$$

$$p(I_i|s) = p(z_i = 1|s) \tag{7.9}$$

where $\delta(v_1, v_2)$ is an indicator function assigning value 1 if $v_1 = v_2$, and 0 otherwise.

7.2.5 Combining Similar Intent Models

In the previous discussions, we fix the number of intent models in I to k. In practice, we can determine the number of intent models dynamically by using a relatively large k and combine similar intent models int the end. To do so we need a measure of similarity between intent models. In our work, we use the KL-Divergence of the word distributions as the similarity function for intent models. Essentially,

$$sim(I_i, I_j) = -KL(\psi_i||\psi_j) \tag{7.10}$$

Table 7.1: Corpora statistics

	Laptop	TV	Camera
#Product	328	272	370
#Session	9633	25489	10819
#Engagement	35808	97995	42269
#Engagement per session	3.7	3.8	3.9

Appropriate smoothing for both ψ_i and ψ_j using the generic word distribution ψ_G is necessary as we need to avoid zero probabilities in computing the KL-Divergence. By thresholding the similarity, we can combine the similar intent models. The combination can be done by modifying the indicator variables in the E-step and reestimate the model parameters. Using this method, the size of intent models can be set dynamically.

7.3 Discovered Shopping Intent

By fitting the joint mixture model to a product search log, we can automatically discover a set of shopping intent in the form of coordinated representation. In this section, we empirically examine the discovered intent models using search engagement logs from a popular commercial product search engine. Particularly, we consider three major categories in online shopping, "Computer Laptops", "TVs" and "Cameras". For each category, we have around 300 products. We obtain a month's search logs and extract engagements for the three categories separately by matching the engaged product entities to the corresponding category's database. Queries that have at least one engagement in the given category will be selected for the category. In total, we have more than 9000 sessions for the "Computer Laptops" category with user engagements (each session containing one query), more than 25000 sessions for the "TVs" category and more than 10000 sessions for the "Cameras" category. On average, there are 3.8 observations of user engagements in each session. The detailed statistics of the corpora are summarized in Table 7.1.

Additional information on intent models. Although the intent models we discovered are easily interpretable, some additional information will have us to better understand the model. Particularly, we need to know the importance of each intent and the representative queries of each intent. Both can be obtained using the estimated parameters from the joint mixture model.

Intent popularity. Intent popularity is an important piece of information that we need to know in order to draw insights from the discovered intent models. Essentially, we want to know how likely users will have a particular intent. Given the ith intent I_i, its popularity is captured by $p(I_i)$, which can be computed as the marginal probability using the joint mixture model parameter $p(I_i|s)$:

$$p(I_i) = \sum_{s \in S} p(I_i|s)p(s) \tag{7.11}$$

Since we don't have any prior knowledge on each session, it is reasonable to assume $p(s)$ is uniformly distributed. With $p(I_i)$, we can determine which intent is more popular in search.

Representative queries Although our intent models capture the preference over query terms, they do not directly relate to queries. It is much more intuitive and easier to understand shopping intent in search if each intent is associated with a set of representative queries. To select representative queries, first we need to rank queries according to the intensity of their relationship with the given intent. According to the generative procedure, we can compute the probability of generating a query q using the word distribution ψ_i given ith intent I_i:

$$p(q|I_i) = \prod_{w \in q} \left((1-\gamma)p(w|\psi_i) + \gamma p(w|\psi_B)\right) \tag{7.12}$$

We may be tempted to use $p(q|I_i)$ to rank and select queries. However, $p(q|I_i)$ is biased as shorter queries will tend to have a higher probability. To conquer this issue, we instead use the posterior probability of intent I_i, $p(I_i|q)$ as a scoring function for ranking queries. $p(I_i|q)$ can be computed as

$$p(I_i|q) = \frac{p(q|I_i)p(I_i)}{\sum_j p(q|I_j)p(I_j)} \tag{7.13}$$

where $p(I_i)$ is the intent popularity computed by Equation 7.11.

Given the ranked list of queries, we still need to select a sub set of them so that they can well represent the intent. To do so, we use the coverage of word probability mass as the selection criteria. More specifically, we keep selecting queries according to the ranking order of the queries until the aggregated probability mass of the words in the selected queries reach a pre-specified threshold. For instance, if the words "portable", "kitchen" and "TV" have probabilities of 0.1, 0.1 and 0.2, respectively, then by selecting two queries "portable TV" and "kitchen TV" we have a coverage of

0.4. If the threshold is smaller than 0.3, we only select the first query; otherwise both queries are selected.

With the additional information, we summarize the discovered most popular intent models in the three categories, “Laptop”, “TV” and “Camera”, in Table 7.2, Table 7.3 and Table 7.4, respectively. For each intent model I_i, we first show the probability $p(I_i)$ as an indication of intent popularity. We then show the representative queries, the top 5 words in ψ_i, the top 5 attributes ranked by $1 - \gamma_{i,a}$, and up to 3 top feature values on each attribute a according to θ_i (values with probabilities smaller than 0.01 are ignored).

Judging from the representative queries in Table 7.2, we can see that the top 3 intent models in “Laptop” category can be interpreted as “laptop bundles”, “netbook laptop” and “sony laptop”, with “laptop bundles” being the most dominating intent. Looking into the words and features, we can see the three intent models are clearly different from each other. The first one is mostly concerned with the “bundle” feature; the second has emphases on a particular type of processor and the size of the laptop; the third intent model preferences the high end CPUs and specific brands and colors. From the discovered intent models for the “TV” category, we can see that the user difference is mostly captured by the screen sizes. Different intent models have clear distinctions on the “Price” attribute and the “Screen Size” attribute. For the “Camera” category, the top 3 intent models are clearly different in the “Type” attribute, emphasizing on “Point Shoot”, “Ultra-zoom” and “DSLR”, respectively. Each intent model also has a unique selection of price ranges.

Overall, we can see that the proposed intent modeling method can successfully capture the user behavioral difference and generate meaningful intent models as well as interpretable representations. The queries selected based on the proposed query selection procedure can effectively represent the intent models.

In the following discussions, we will introduce how we perform quantitive evaluations of the discovered intent models through applications including product search and product recommendation. It is worth noting that it is generally difficult to perform direct quantitative evaluation on the discovered intent models because users have very little prior knowledge of them and as a result their judgement could be very subjective. However, it is possible to evaluate certain aspects of the intent models through carefully designed user studies. For example, we could evaluate the coherence of intent models by showing part of the coordinated intent representation (e.g. queries) and ask a user

Table 7.2: Top discovered intent models in “Laptop” category

Intent 1. $p(I) = 0.52$		
Queries	laptop bundle	
Words	laptop, computer, bundle, gateway, refurbished	
Features	Type	Bundle
	Processor	Celeron, Core 2 Duo, Core i3
	RAM	2GB, 4GB
	Screen size	11”-14”, 17” & larger, 15”-16”
	Brand	Dell, Acer, HP
Intent 2. $p(I) = 0.11$		
Queries	netbook, hp mini, acer netbook, hp netbook, mini laptop	
Words	laptop, netbook, acer, computer, mini	
Features	Processor	Atom
	RAM	1GB, 2GB
	Screen size	10” & under, 11”-14
	Type	Netbook, Bundle
	Hard Drive	250GB-640GB, 250GB&under
Intent 3. $p(I) = 0.10$		
Queries	pink laptop, sony vaio, sony, sony laptop	
Words	laptop, sony, pink, computer, vaio	
Features	Processor	Core i5, Core i3, Core i7
	Brand	Sony, HP
	Color	White, Silver, Pink
	RAM	8GB, 4GB, 6GB
	Screen size	11”-14”, 15”-16”, 17”&larger

Table 7.3: Top discovered intent models in "TV" category

Intent 1. $p(I) = 0.21$		
Queries	vizio 47, vizio 47 class, vizio 42 class, vizio 47 class theater, vizio 3d	
Words	tv, vizio, 3d, led, smart	
Features	Price	\$500-\$750, \$250-\$500, \$750-\$1k
	Screen Size	40"-49", 50"-59"
	Resolution	1080p
	Brand	Vizio, LG, RCA
	Refresh	120Hz, 60Hz
Intent 2. $p(I) = 0.18$		
Queries	32 inch tv, 32 tv, pink tv, 32 inch, 32 in tv, 32 class, sceptre	
Words	tv, 32, vizio, inch, lcd	
Features	Price	\$200-\$250, \$250-\$500, \$150-\$200
	Screen Size	30"-39", 21"-29"
	Brand	Sceptre, Vizio, RCA
	Refresh	60Hz
	Technology	LCD, LED
Intent 3. $p(I) = 0.11$		
Queries	19 inch tv, tv with dvd, 19 inch flat screen tv, 22 tv, 22 inch tv, vizio 22	
Words	tv, dvd, vizio,with, combo	
Features	Price	\$100-\$150, \$150-\$200, \$200-\$250
	Screen Size	21"-29", 20"&Smaller, 30"-39"
	Brand	Vizio, RCA, Emerson
	Technology	LED, LCD
	Refresh	60Hz, 120Hz

to choose from a list of candidates for the remaining part of the intent representation (e.g. product features). We leave this for future work because it requires a lot of human efforts.

Table 7.4: Top discovered intent models in "Camera" category

Intent 1. $p(I) = 0.19$		
Queries	canon powershot, nikon coolpix, point shoot camera	
Words	camera, digital, canon, shoot, point	
Features	Type	Point Shoot, Ultra Zoom
	Price	\$50-\$100, \$100-\$150, \$150-\$200
	Features	Image Stabilization, HD Video, Face Detection
	Optical	2x - 8x, 9x - 18x
	Brand	Canon, Nikon, Samsung
Intent 2. $p(I) = 0.15$		
Queries	ultra-zoom camera, olympus, ge camera	
Words	camera, ultra-zoom, dslr, digital, olympus	
Features	Price	\$150-\$200, \$100-\$150, \$200-\$250
	Type	Ultra Zoom, DSLR, Point Shoot
	Optical	20x & Up, 9x - 18x
	Features	HD Video, Image Stabilization
	Brand	GE, Olympus, Samsung
Intent 3. $p(I) = 0.14$		
Queries	nikon d5100, nikon d3100, nikon d3200, canon t3i, nikon dslr	
Words	nikon, camera, dslr, d5100, d3100	
Features	Price	\$500-\$750, \$250-\$500, \$750-\$1000
	Type	DSLR, Ultra Zoom
	Features	HD Video, Image Stabilization
	Color	Black
	Brand	Nikon, Canon

7.4 Applications

The proposed query intent model enables a wide arrange of applications. Below we discuss some of them.

7.4.1 Intent-Oriented Query Ambiguity Analysis

Query ambiguity analysis is an important task in query understanding. It has impact in many practical applications such as search result personalization and diversification. Traditionally, query ambiguity is measured by dispersion of user engagements. In terms of Web search, this mainly corresponds to the click behavior. In particular, a widely used metric is a straight forward computation of information entropy over the distribution of clicks, i.e. *click entropy*:

$$Clk.Ent(q) = \sum_{e \in E_q} p(e|q) \log \frac{1}{p(e|q)} \tag{7.14}$$

where q is a query, E_q is the set of user clicked entities and $p(e|q)$ is an empirical distribution of clicks given the query q.

One issue with the click entropy metric is that it captures the document level ambiguity, not the behavioral level ambiguity. Previously, we proposed *click pattern* to capture the behavioral differences in analyzing query ambiguity [26], but the method relies on observation level analysis and could not gain insight into the behavioral differences due to the lack of structure in text documents.

In this study, we propose to study intent-oriented query ambiguity analysis for product search. Rather than analyzing based on the observations, we directly measure query ambiguity based on shopping intent. This is done by computing the Intent distribution for each query using the intent we discovered in the joint mixture model. Particularly, the intent distribution given a user query is computed exactly the same as in Equation 7.13. With the intent distribution $p(I|q)$, we can compute the *intent entropy* for a query as follows:

$$Int.Ent(q) = \sum_{i=1}^{k} p(I_i|q) \log \frac{1}{p(I_i|q)} \tag{7.15}$$

In Table 7.5 and Table 7.6 we show examples of queries and the computation of ambiguity measures. The left side of the table shows the queries with the highest intent entropy and the right side shows the queries with lowest intent entropy. For each query we show the intent entropy and the click entropy. We can see queries with high intent entropy are mostly the ones that only specify preferences on few attributes, e.g. “hp”, “refurbished laptops”. For these queries, the users have not decided what product they want to purchase when they issue the query, therefore displaying differences in future search engagement. With the intent entropy measure, we can effectively identify these queries, then we can perform better diversification or personalization of search results. Although the click entropy of these queries also tend to be larger, they are not as consistent as intent

Table 7.5: Queries with High Intent Entropy in Laptop Category

Query	Int.Ent	Clk.Ent
hp	0.83	1.34
hp laptops	0.78	1.17
refurbished laptops	0.73	0.96
dell laptops	0.73	0.7
asus	0.72	1.16
hp laptop	0.72	1.22
refurbished laptop	0.71	0.93
laptop	0.69	1.2
refurbished	0.69	0.91
laptop pc with intel	0.64	1.26

Table 7.6: Queries with Low Intent Entropy in Laptop Category

Query	Int.Ent	Clk.Ent
hp - pavilion g6-1c58dx / amd quad-core a6-3400m ...	0	0.59
acer - gateway 15.6’ laptop - 4gb memory - 500gb ...	0	0.48
samsung 13.3’ amd dual-core a6-4455m laptop ...	0	0.28
acer aspire one ’ao722 0473’ ’11 6 inch’ hd netbook ...	0	0.41
tyhe dark purple laptop bundle optional matching ...	0	0.45
the blue laptop bundle with optional matching ...	0	0.8
lenovo thinkpad x130e 062223u 11.6-inch led ...	0	0.28
hp 14” g4-2149se butterfly blossom design laptop ...	0	0
17.3’ hp laptop amd quad core accelerated processor	0	0.48
hp envy 15-3040nr 15.6 inch laptop (black/silver)	0	0.3

entropy, as the values ranges from 0.7 to 1.34. For the queries with small intent entropy, they are mostly long queries where detailed preferences on all aspects are specified. Some of the queries could be directly copied from the Web. In these cases, the users have clear targets in mind. However, the search results may not contain the specified product, or contain more interesting products that can substitute the product the user has in mind. As a result, the user may still engage with multiple products. Consequently, the click entropy metric is problematic in measuring the ambiguity of the query, as it will regard each engaged product as individual semantic unit. In computing intent entropy, we are able to detect that that although the user engaged with multiple products, these products obey a coherent intent, as their features can be well explained by a single intent model.

7.4.2 Improving Product Search

The most intuitive and important usage of shopping intent models in practical applications is to improve the accuracy of product search. In this section, we study how to incorporate the intent models with existing IR model. Particularly, we integrate the discovered intent models with language model for IR. With the KL-Divergence language model, the scoring function is essentially the negative KL-Divergence of the query language model and the document/product language model:

$$score(q, e) = \sum_{w \in W} p(w|q) \log p(w|e) \tag{7.16}$$

Without any assumption of feedback data, the query language model is simply estimated using the query itself. Typically, the document language model is estimated by the content of the document, and smoothed by the collection language model. In the case of product search, we can only use the limited product information (i.e. words describing each feature) for estimating the product language model. Our intent model provides a natural bridge of products and query words. Therefore, we can augment the product language model by computing a new intent based model:

$$p(w|e) = \sum_{i=1}^{k} p(w|I_i)p(I_i|e) \tag{7.17}$$

where

$$p(I_i|e) = \frac{p(e|I_i)p(I_i)}{\sum_{j=0}^{k} p(e|I_j)p(I_j)} \tag{7.18}$$

where

$$p(e|I_i) = \prod_{a \in A} (1 - \gamma_{i,a})p(v_{e,a}|\theta_i) + \gamma_{i,a}p(v_{e,a}|\theta_{G,a}) \tag{7.19}$$

There are two ways to combine the language models. One is to merge the two models to get one model. The other way is to keep two separate models and combine the scores. Interestingly, we find the second method tend to perform better than the first one. Therefore, we use linear interpolation of scores to integrate the intent based product language model with the base language model. The interpolation weight is set by cross validation.

To evaluate the performance of product search, we created an evaluation set by randomly selecting queries and asking experienced industrial annotators to label the returned products from the search engine into 5 relevance degrees. In total we obtain 96 queries for the “Laptop” category, 146 for “TV” and 98 for the “Camera” category. We evaluate the search systems based on NDCG@3 and NDCG@10.

Table 7.7 shows the experiment of product search. In this experiment, we compare several methods. The first is a base language model approach, Base.LM, where the product language models are estimated based on product features. Int.LM is the method which estimates product language models based on the intent models we discovered. A combination of the two different types of language models is also evaluated. We then modify our intent discovery method to discard the use of product structure. This can be easily done by fixing the γ_{i_a} to a constant value in the joint mixture model in Section 7.2.3 (same as λ). We compute another type of language model based on this method, denoted as Int.LM$^-$. The reason we include Int.LM$^-$ in our study is that we want to see if we can gain advantage by incorporating the product structure in modeling intent. Because the intent modeling methods all use product search logs for estimation, they are indirectly using relevance feedback information. Therefore, we also include a language model with relevance feedback (FB.LM) for comparison. We follow the method proposed by Zhai and Lafferty [79] to implement FB.LM.

From Table 7.7 we can see that both Int.LM and Int.LM$^-$ outperforms Base.LM. This is expected as we make use of feedback information in estimating the intent models. However, we also observe that Int.LM almost always outperform the typical feedback method FB.LM, and by combining Int.LM and Base.LM we can significantly improve over both Base.LM and FB.LM in all three categories. This shows that the intent modeling approach is a superior way of using relevance feedback. One main reason is that through intent modeling, we can generalize the knowledge from the feedback information, whereas the existing feedback method only use the feedback information for the corresponding query (i.e. queries with no feedback will not be treated). In comparison with Int.LM$^-$, Int.LM consistently performs better. This shows that our method is effective in modeling the structure of products. Finally, we find that for both Int.LM and Int.LM$^-$, the search accuracy could be further improved by combining with the Base.LM model. The best performance is achieved by Base+Int.LM in all three categories. In combining the two types of language models, we also find that the score-interpolation tend to generate better results than model-interpolation.

Table 7.7: Evaluation of product search

Category	Method	NDCG@3	NDCG@10
Laptop	Base.LM	0.275	0.161
	FB.LM	0.387	0.222
	Int.LM$^-$	0.383	0.218
	Int.LM	0.390	0.223
	Base+Int.LM$^-$	0.399	0.234
	Base+Int.LM	**0.409**	**0.240**
TV	Base.LM	0.527	0.366
	FB.LM	0.640	0.439
	Int.LM$^-$	0.641	0.442
	Int.LM	0.647	0.445
	Base+Int.LM$^-$	0.656	0.459
	Base+Int.LM	**0.661**	**0.461**
Camera	Base.LM	0.512	0.395
	FB.LM	0.593	0.463
	Int.LM$^-$	0.588	0.459
	Int.LM	0.597	0.461
	Base+Int.LM$^-$	0.605	0.475
	Base+Int.LM	**0.607**	**0.477**

7.4.3 Improving Product Recommendation

Product recommendation is an important task in e-commerce. To make product recommendations, we need to be able to measure the similarity of two products. Typically, the similarity can be measured by the product features. Essentially, we can compute the similarity of two products as the promotion of the same features. One problem with this method is that it cannot distinguish the importance of different attributes. If two products differ on only a few important attributes, they could still have high similarity since their majority features are the same, although they are likely designed for very different purposes.

To solve this problem, we study use the discovered intent models. Conceptually, our intent models not only bridge the products and queries, but also connect products with other products. If two products have similar intent distributions, we could infer with a high confidence that they are good recommendations for each other. But sometimes products don't have similar intent distributions, but they both have high probabilities on same intent. In this case, they could still be good recommendation candidates for each other. To capture this intuition, we use cosine similarity to measure the similarity of intent distributions. In particularly, given two products e_1 and e_2, their similarity is computed as:

$$cos(e_1, e_2) = \frac{\sum_{i=1}^{k} p(I_i|e_1)p(I_j|e_2)}{\sqrt{\sum_{i=1}^{k} p(I_i|e_1)^2}\sqrt{\sum_{i=1}^{k} p(I_i|e_2)^2} \tag{7.20}$$

Similarly to product search, we add this new similarity as an additional signal for product recommendation. The weight is set by cross validation.

To evaluate product recommendation, we create an evaluation set by extracting highly related products based on session co-occurrence in product search. To avoid bias, we use a different time stamp from the data we use to estimate intent models. The two logs used are half a year apart. We use a high threshold so that only we only extract those recommendation candidates that we are confident with. Using this method, we generated recommendations for 133 products in "Laptop" category (out of 328 products), 95 products in "TV" category (out of 272) and 190 products in "Camera" category (out of 370). On average, each of these products has 2.8 recommendations. To evaluate different methods, we use each to generate a ranking of products based on an input product. We then select top 5 products and measure the precision, recall and F1 score.

Table 7.8 shows the evaluation of several methods for product recommendation. In this table, co-occurrence counting (Co.Cnt) is a baseline method where the same process of generating the evaluation ground truth is used to generate product recommendations, using the training search log (which has a different time stamp than the one used for generating ground truth). A low threshold was used in this method to guarantee the coverage. Feat.Sim method uses the proportion of same product features to define product similarity. Tpc.Sim and Int.Sim are the cosine similarities based on the discovered topics and intent models, respectively. From the table, we can see that by incorporating the Intent similarity, we can always improve the accuracy of product recommendation. The topic similarity could also improve the performance in some of the categories, but overall it is suboptimal compared to intent similarity. This is mainly because that the intent similarity benefits from simultaneously modeling queries and products in search engagement. As shown in our previous discussions, since both queries and product engagements are ambiguous, by modeling them in a single mixture model, we can effectively eliminate the ambiguity in discovering the representative intent models. We also find that the co-occurrence counting method is not very consistent across different categories. This is because the method mainly does not perform any generalization of the knowledge. Depending on the fluctuation of user behavior, such a method may not work

Table 7.8: Evaluation of product recommendation

Laptop			
Method	Precision	Recall	F1
Co.Cnt	0.103	0.195	0.136
Feat.Sim	0.131	0.203	0.159
Feat+Tpc.Sim	0.126	0.204	0.156
Feat+Int.Sim	0.135	0.202	0.162
Feat+Int.Sim+Co.Cnt	**0.166**	**0.272**	**0.207**
TV			
Method	Precision	Recall	F1
Co.Cnt	0.103	0.207	0.138
Feat.Sim	0.119	0.247	0.162
Feat+Tpc.Sim	0.132	0.285	0.181
Feat+Int.Sim	0.147	0.310	0.200
Feat+Int.Sim+Co.Cnt	**0.152**	**0.329**	**0.207**
Camera			
Method	Precision	Recall	F1
Co.Cnt	0.174	0.325	0.227
Feat.Sim	0.138	0.282	0.136
Feat+Tpc.Sim	0.141	0.277	0.187
Feat+Int.Sim	0.144	0.299	0.194
Feat+Int.Sim+Co.Cnt	**0.197**	**0.402**	**0.264**

well when used alone due to the lack of coverage. However, by combining the Co.Cnt signal with Feat+Int.Sim, we can always improve the accuracy of recommendation. The direct co-occurrence counting and the Intent similarity complements each other as co-occurrence counting captures the most confident recommendation signal and intent similarity provides more coverage by successfully generalizing the co-occurrence knowledge in user engagement data.

7.5 Conclusion and Future Work

In this work, we introduced and studied a novel problem of search intent understanding in product search. To comprehensive characterize users' shopping intent in product search, we proposed a coordinated representation of intent, where each intent is collectively represented the query term preference, product attribute importance and product feature preference. To discover such intent representations, we proposed a novel joint mixture model to simultaneously model the unstructured queries and the structured products in search engagements. By employing an EM algorithm, we can automatically discover the representative shopping intents in a product search log. Evaluation results show that the model is effective for discovering distinct, coherent and interpretable shopping intents. The discovered intents are quite meaningful and representative, and can serve as a ”summary” of user interests and preferences. We further proposed several applications of the proposed intent model, including query ambiguity analysis, product search and product recommendation. The results of both qualitative and quantitative evaluation demonstrated clear benefits of leveraging the proposed intent model in all the applications.

Our work opens up many interesting future directions. First, it opens up new ideas in search intent modeling, where detailed intent representation is the key for improvement. Second, it opens new direction in search log mining where both unstructured data and structured data need to be modeled simultaneously. Finally, the novel intent model we proposed in product search will enable/improve many practical applications. In the future, we plan to continue improving the intent model by taking into account more types of user engagements and investigate into its values in more critical applications such as business intelligence.

CHAPTER 8

SUMMARY AND FUTURE DIRECTIONS

8.1 Summary

This thesis is focused on the general problem of search intent modeling and query reformulation in search engine systems. It explored three main topics: *Modeling ambiguous search intent*, *Supporting query formulation* and *Deep intent modeling in structured entity retrieval*.

In *Modeling ambiguous search intent*, I proposed to capture ambiguous search intent based on behavioral difference instead of content difference as most typical ambiguity metrics do. I proposed to use click pattern as a building block for search log analysis. Each query is characterized by a set of click patterns, i.e. a click profile. A measurement based on this behavioral analysis, i.e. pattern entropy, was then proposed to quantify the intent ambiguity in search queries. I proposed and analyzed three important aspects of ambiguity measurements. Experiments showed that pattern entropy is a consistent and robust metric that can accurately quantify search intent ambiguity.

As part of the findings in *Modeling ambiguous search intent*, we know that a large portion of search queries are illy composed. Users' lack of domain knowledge has largely affected the query formation process. In order to guide users to form the most effective queries based on their search intent, I studied the topic of *Supporting query formulation*. Particularly, I studied two problems. First, I studied the problem of automatically completing and correcting users' queries based on their search intent. To solve this problem, I employed joint sequence model to model the query transformation. With an EM algorithm, the model can be estimated from past observations of users' query correction behaviors. Experiments confirmed that the proposed model can help users form more effective search queries based on their search intent with less effort. Second, I studied the effect of query representation in expressing search intent. Particularly, I studied whether the typical keyword query representation is sufficient for expressing search intent. Through the comparison with an augmented query representation space with syntactic query operators, I found that the augmented query space can be significantly more effective in capturing search intent, as it constantly results in better search performance. Since users are usually not familiar with syntactic query operators, I then studied the feasibility of automatic reformulation from keyword queries to syntactic queries. Experiments showed that the reformulation can be effectively performed in scenarios when users encounter search difficulty and certain negative feedback information becomes available.

The previous two parts of work have focused on general techniques for search intent modeling and query reformulation. Indeed, accurate and comprehensive modeling of search intent in regular document retrieval is difficult due to the variety of intent types and the lack of document structures. On the other hand, structured entity retrieval, an increasingly important retrieval task, uses mainly structured data and has less variety of intent types, raising opportunities as well as challenges in search intent modeling. Therefore, in the third part of the thesis work, I studied *Deep intent modeling in structured entity retrieval*. First, I studied query intent modeling with data level representation (i.e. entity structures). I proposed and studied a probabilistic retrieval model that can effectively match query intent into entity structure representation. The model can be estimated using any text data source associated with the entities, such as user reviews and observed search queries. Experiments confirm that with the proposed model, product search in keyword queries can be significantly improved. This study bridged the gap between intent representations in queries and entities in product search, but it does not provide much insight in understanding users' shopping intent. In product search, different users may use the same query to express different intent, or use different queries to express the same intent. To model such ambiguities and accurately capture users' shopping intent, I studied automatic discovery of users' shopping intent in product search. A coordinated intent representation was proposed, which can simultaneously capture the topical words and the characteristic product features of each intent. I then proposed a novel joint mixture model to automatically discover the coordinated intent representations from search engagement logs. Experiments show that the discovered shopping intent models can not only improve search accuracy, but also benefit many search related applications such as product recommendation. Also, as a building block for query analysis, this also extended our work on *Modeling ambiguous intent with click pattern*, which performs query analysis mainly on the observation level. With the automatically discovered, comprehensive intent models in the coordinated representation, we can perform accurate ambiguity analysis on all search quires, even if they have not been observed before.

In conclusion, this thesis work is a systematic study of search intent understanding, and how to leverage it to improve the performance of search engine systems. The contributions of the thesis work are:

- **Behavior analysis in query intent understanding.** As shown in my work on modeling ambiguous search intent with click pattern, behavior analysis is of critical importance in understanding search intent in queries. Compared traditional query analysis methods based on search result contents, behavior analysis can lead to more accurate understanding of search intent.

- **Supporting search intent expression.** The work on query completion and correction is among the pioneering works in the area. The importance of the task is established and it is shown that real time support is feasible and can be achieved with high accuracy.

- **Understanding query space in intent representation.** The impact of query representation on intent expression is analyzed. It is shown that with syntactic query operators, we can achieve an augmented query space that can greatly improve the expressing of search intent from the keyword query space. It is also shown automatic reformulation from keyword query to the augmented query representation is possible when users encounter difficulty in search, so that their search intent can be "diagnosed" and refined.

- **Modeling query intent with data level representation.** The study shows that in special IR domains, we can deep intent models with a date level representation, which has critical impact on search accuracy. It is confirmed through experiments that the proposed probabilistic model can effectively match search queries with the product feature representation of search intent, and successfully incorporate the information in product search to improve search accuracy.

- **Discovering coordinated shopping intent.** The study shows the importance of modeling search intent at a higher conceptual level, where queries and products serving similar shopping intent can be identified and grouped together. The proposed coordinated representation is shown to be effective in characterizing users' shopping intent, and the proposed joint mixture model can automatically identify and capture such intent in search engagement logs.

8.2 Discussions and Future Directions

Although the thesis work is logically separated into three parts, it follows a clear road map of studying intent modeling and query reformulation, and none of the parts is isolated work. As aforementioned, the work on deep intent modeling extends the work on modeling ambiguous search intent, because it enables a meaningful, data level representation for each intent group whereas the previous modeling method is based on behavior observations. The data level representation not only differentiates each intent group, but also provides an explanation of their differences. In terms of click patterns, we can discover that a certain type of click behavior is formed because the related items obey a data specific distribution determined by the intent group. This allows us to more accurately model the ambiguity in search queries, and also to gain insights into users' behavior. Meanwhile, the work on deep intent modeling also has potential impact on supporting query formulation. For instance, the coordinated intent representation enables many potentially useful features for syntactic query reformulation. Because query terms are clustered into different intent groups, we can leverage this information to determine the importance of terms and improve the prediction for necessity operator. It would be interesting to further explore how to use the intent models we discovered to improve query analysis and query reformulation.

In this work, we have mainly focused on structured data retrieval for studying deep intent modeling. Another important future direction is to explore the data level intent modeling in text retrieval. Deep intent understanding is generally difficult for text data due to the lack of structures. In recent years, there has been great effort in the semantic web community to transfer text data into structured/semi-structured data. Since it has been shown that our proposed techniques can effectively model structured entity data with unstructured text data, we can leverage the information extraction techniques to obtain the necessary structural information from text data, and adapt our methods to model the underlying user intent.

We have also mostly focused on studying users' intent in a static time period in this work. In reality, users' intent could change or shift over time. Understanding such changes of intent is not only useful for improving search accuracy but also important to business intelligence. Consider product search for example, being able to predict the trend of shopping intent can help merchants make smart purchasing plans and better organize their stock. Furthermore, certain types of user intent also show periodical behaviors. Detecting such periodicity is also important for making shelf arrangement. To capture the changes of user intent, we need to incorporate the temporal aspect into our intent models. Particularly, we can impose proper regularizations based on time specific models.

Yet another interesting future direction is to capture term dependency in intent mining. For example, given a query "cheap laptop case", we should understand the primary intent is "laptop case" instead of "laptop". Such dependency has been studied in our work on syntactic query reformulation (Chapter 5). But in the search intent modeling work we have mostly assumed term independency. In product search, we have made similar assumption for product attributes. Typically, such assumptions can simplify the modeling work and achieve robust results. But as shown in our work on syntactic query reformulation, with proper use of dependency we can potentially further improve the accuracy of intent modeling. Thus an interesting future direction would be to refine the proposed intent models by taking into account term and attribute dependencies.

REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of ICDE 2002*, 2002.

[2] D. N. Aurelio and R. R. Mourant. The effects of web search engine query ambiguity and results sorting method on user performance and preference. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 46(14):1271–1275, 2002.

[3] R. Baeza-yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *In International Workshop on Clustering Information over the Web (Clust-Web, in conjunction with EDBT), Creete*, pages 588–596. Springer, 2004.

[4] K. Balog, Y. Fang, M. de Rijke, P. Serdyukiv, and L. Si. Expertise finding. *Foundations and Trends in Information Retrieval*, 6(3), 2012.

[5] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 491–498, New York, NY, USA, 2008. ACM.

[6] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proceedings of the 1999 ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229, 1999.

[7] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 795–804, New York, NY, USA, 2011. ACM.

[8] J. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, 1998.

[9] M. Bisani and H. Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Commun.*, 50(5):434–451, May 2008.

[10] G. Bouma. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, pages 31–40, 2009.

[11] S. Büttcher, C. L. A. Clarke, and B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 621–622, New York, NY, USA, 2006. ACM.

[12] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.*, 31(3):1134–1168, 2006.

[13] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 707–718, New York, NY, USA, 2009. ACM.

[14] Q. Chen, M. Li, and M. Zhou. Improving query spelling correction using web search results. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 181–189, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[15] S. F. Chen. Conditional and joint models for grapheme-to-phoneme conversion.

[16] T. Cheng, H. W. Lauw, and S. Paparizos. Entity synonyms for structured web search. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1862–1875, 2012.

[17] T. Cheng, X. Yan, and K. C.-C. Chang. Entityrank: Searching entities directly and holistically. In *VLDB*, pages 387–398, 2007.

[18] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 87–94, New York, NY, USA, 2008. ACM.

[19] S. Cronen-Townsend and W. B. Croft. Quantifying query ambiguity. In *Proceedings of the second international conference on Human Language Technology Research*, HLT '02, pages 104–109, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[20] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 299–306, New York, NY, USA, 2002. ACM.

[21] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In D. Lin and D. Wu, editors, *Proceedings of EMNLP 2004*, pages 293–300, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[22] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, Mar. 1964.

[23] J. J. Darragh, I. H. Witten, and M. L. James. The reactive keyboard: A predictive typing aid. *Computer*, 23(11):41–49, Nov. 1990.

[24] K. Darwish and D. W. Oard. Probabilistic structured query methods. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 338–344, New York, NY, USA, 2003. ACM.

[25] H. Daumé, III and E. Brill. Web search intent induction via automatic query reformulation. In *Proceedings of HLT-NAACL 2004: Short Papers*, HLT-NAACL-Short '04, pages 49–52, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[26] H. Duan, E. Kiciman, and C. Zhai. Click patterns: an empirical representation of complex query intents. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 1035–1044, New York, NY, USA, 2012. ACM.

[27] G. Dupret and B. Piwowarski. A user behavior model for average precision and its generalization to graded judgments. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 531–538, New York, NY, USA, 2010. ACM.

[28] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *VLDB*, pages 696–707, 1990.

[29] N. Fuhr. A probabilistic relational model for the integration of ir and databases. In *SIGIR*, pages 309–317, 1993.

[30] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[31] K. Ganesan and C. Zhai. Opinion-based entity ranking. *Inf. Retr.*, 15(2):116–150, 2012.

[32] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A large scale ranker-based system for search query spelling correction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 358–366, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[33] J. Gonzalo, F. Verdejo, I. Chugur, and J. Cigarrin. Indexing with wordnet synsets can improve text retrieval. In *Proceedings of CORR*, pages 38–44, 1998.

[34] J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 379–386, New York, NY, USA, 2008. ACM.

[35] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 1419–1420, New York, NY, USA, 2008. ACM.

[36] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[37] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 850–861. VLDB Endowment, 2003.

[38] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proceedings of VLDB 2002*, 2002.

[39] J. Hu, G. Wang, F. Lochovsky, J.-t. Sun, and Z. Chen. Understanding user's query intent with wikipedia. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 471–480, New York, NY, USA, 2009. ACM.

[40] A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 431–, Washington, DC, USA, 2002. IEEE Computer Society.

[41] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[42] R. Krovetz and W. B. Croft. Lexical ambiguity and information retrieval. *ACM Trans. Inf. Syst.*, 10:115–141, April 1992.

[43] G. Kumaran and J. Allan. A case for shorter queries, and helping users create them. In *In NAACL-HLT*, pages 220–227. HLT, 2007.

[44] G. Kumaran and J. Allan. Effective and efficient user interaction for long queries. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 11–18, New York, NY, USA, 2008. ACM.

[45] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 564–571, New York, NY, USA, 2009. ACM.

[46] J. Lafferty and C. Zhai. Probabilistic relevance models based on document and query generation. 2003.

[47] M. Lalmas. *XML Retrieval (Synthesis Lectures on Information Concepts, Retrieval, and Services)*. Morgan and Claypool, 2009.

[48] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, Feb. 1966.

[49] M. Li, Y. Zhang, M. Zhu, and M. Zhou. Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 1025–1032, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[50] X. Li, Y.-Y. Wang, and A. Acero. Extracting structured information from user queries with semi-supervised conditional random fields. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 572–579, New York, NY, USA, 2009. ACM.

[51] X. Li, Y. yi Wang, and A. Acero. Learning query intent from regularized click graphs. In *SIGIR 2008*, pages 339–346. ACM, 2008.

[52] Q. Mei and K. Church. Entropy of search logs: how hard is search? with personalization? with backoff? In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 45–54, New York, NY, USA, 2008. ACM.

[53] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 206–214, New York, NY, USA, 1998. ACM.

[54] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the ACM SIGIR*, pages 275–281, 1998.

[55] J. Pound, S. Paparizos, and P. Tsaparas. Facet discovery for structured web search: a query-log mining approach. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 169–180, New York, NY, USA, 2011. ACM.

[56] Y. Qiu and H. P. Frei. Concept based query expansion, 1993.

[57] Y. Rasolofo and J. Savoy. Term proximity scoring for keyword-based retrieval systems. In *Proceedings of the 25th European conference on IR research*, ECIR'03, pages 207–218, Berlin, Heidelberg, 2003. Springer-Verlag.

[58] E. M. Riseman and A. R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *IEEE Trans. Comput.*, 23(5):480–493, May 1974.

[59] S. E. Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33(4):294–304, Dec. 1977.

[60] M. Sanderson. Ambiguous queries: test collections need more sense. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 499–506, New York, NY, USA, 2008. ACM.

[61] M. Sanderson and C. J. Van Rijsbergen. The impact on retrieval effectiveness of skewed frequency distributions. *ACM Trans. Inf. Syst.*, 17:440–465, October 1999.

[62] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 771–782, New York, NY, USA, 2010. ACM.

[63] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[64] R. Song, Z. Luo, J.-R. Wen, Y. Yu, and H.-W. Hon. Identifying ambiguous queries in web search. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1169–1170, New York, NY, USA, 2007. ACM.

[65] C. Stokoe, M. P. Oakes, and J. Tait. Word sense disambiguation in information retrieval revisited. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 159–166, New York, NY, USA, 2003. ACM.

[66] X. Sun, J. Gao, D. Micol, and C. Quirk. Learning phrase-based spelling error models from clickthrough data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 266–274, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[67] T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 295–302, New York, NY, USA, 2007. ACM.

[68] P. Taylor. Hidden markov models for grapheme to phoneme conversion, 2005.

[69] J. Teevan, S. T. Dumais, and D. J. Liebling. To personalize or not to personalize: modeling queries with variation in user intent. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 163–170, New York, NY, USA, 2008. ACM.

[70] O. Uzuner, B. Katz, and D. Yuret. Word sense disambiguation for information retrieval. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 985–, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[71] E. M. Voorhees. Overview of the trec 2004 robust retrieval track. In *In Proceedings of the Thirteenth Text REtrieval Conference (TREC2004*, page 13, 2004.

[72] X. Wang, H. Fang, and C. Zhai. Improve retrieval accuracy for difficult queries using negative feedback. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 991–994, New York, NY, USA, 2007. ACM.

[73] X. Wang, H. Fang, and C. Zhai. A study of methods for negative relevance feedback. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 219–226, New York, NY, USA, 2008. ACM.

[74] Y. Wang and E. Agichtein. Query ambiguity revisited: clickthrough measures for distinguishing informational and ambiguous queries. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 361–364, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[75] S. F. Weiss. Learning to disambiguate. *Information Storage and Retrieval*, 9(1):33–41, 1973.

[76] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '96, pages 4–11, New York, NY, USA, 1996. ACM.

[77] E. Yilmaz, M. Shokouhi, N. Craswell, and S. Robertson. Expected browsing utility for web search evaluation. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 1561–1564, New York, NY, USA, 2010. ACM.

[78] Z.-J. Zha, L. Yang, T. Mei, M. Wang, and Z. Wang. Visual query suggestion. In *Proceedings of the 17th ACM international conference on Multimedia*, MM '09, pages 15–24, New York, NY, USA, 2009. ACM.

[79] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 403–410, New York, NY, USA, 2001. ACM.

[80] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of SIGIR'2001*, pages 334–342, Sept 2001.

[81] L. Zhao and J. Callan. Term necessity prediction. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 259–268, New York, NY, USA, 2010. ACM.