A SYSTEMATIC STUDY OF MULTI-LEVEL QUERY UNDERSTANDING

BY

YANEN LI

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, 2014

Urbana, Illinois

Doctoral Committee:

Professor Chengxiang Zhai, Chair
Professor Jiawei Han
Professor Bruce R. Schatz
Professor Dan Roth
Doctor Bo-June (Paul) Hsu, Microsoft Research

Abstract

Search and information retrieval technologies have significantly transformed the way people seek information and acquire knowledge from the internet. To further improve the search accuracy and usability of the current-generation search engines, one of the most important research challenges is for a search engine to accurately understand a user's intent or information need underlying the query.

This thesis presents a systematic study of query understanding. In this thesis I have proposed a conceptual framework where there are different levels of query understanding. And these levels of query understanding have natural logical dependency. After that, I will present my studies on addressing important research questions in this framework.

First, as a major type of query alteration, I addressed the query spelling correction problem by modelling all major types of spelling errors with a generalized Hidden Markov Model. Second, query segmentation is the most important type of query linguistic signals. I proposed a probabilistic model to identify the query segmentations using clickthrough data. Third, synonym finding is an important challenge for semantic annotation of queries. I proposed a compact clustering framework to mine entity attribute synonyms for a set of inputs jointly with multiple information sources. And finally, in the dynamic query understanding, I introduced the horizontal skipping bias which is unique to the query auto-completion process (QAC). I then proposed a novel two-dimensional click model for modeling the QAC process with emphasis on such behavior.

To Chenchen, Mason and Parents

Acknowledgments

First of all I want to express my deepest gratitude to my advisor Professor ChengXiang Zhai, who guided me through my whole Ph.D study. This thesis would not have been possible without his advice. Professor Zhai introduced me to the wonderful world of Information Retrieval and Text Mining; he later guided me to focus on an important topic of multi-level query understanding in web search. In the pursuit of my Ph.D thesis, I have been always inspired by his passion, vision and knowledge in the field. Also, I am heartily thankful to the way Professor Zhai treated his students. He worked with me like peers and colleagues: he always encouraged me to think critically and independently, defended and improved my ideas through discussion with him. From him I have learned the way of being an independent researcher.

I also want to thank my doctoral committee members, Professor Jiawei Han, Professor Dan Roth, Professor Bruce R. Schatz and Doctor Bo-June (Paul) Hsu, 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, Huizhong Duan, Hongning Wang, Hongbo Deng, Anlei Dong, Yi Chang and Kuansan Wang. 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 the University of Illinois at Urbana-Champaign.

I would like to thank my parents, for their unconditioned love, patience and encouragement through my pursuit of my Ph.D thesis. Finally I am grateful to my wife Chenchen Feng and my son Mason Li for their love, faith and confidence in me. Without their support my achievements would have been impossible.

Table of Contents

Chapter 1

Introduction

Search and information retrieval technologies have significantly transformed the way people seek information and acquire knowledge from the internet. The effectiveness of Web search engines such as Google, Bing and Yahoo significantly affects the quality of our life and our productivity. To further improve the search accuracy and usability of the current-generation search engines, one of the most important research challenges is for a search engine to accurately understand the user's intent or information need behind a query. Accurate understanding of the user issued queries also enables new types of applications that help the user make decision and finish tasks directly, resulting in great increase of productivity.

However, accurate query understanding is not an easy task due to the following challenges. First, a query usually contains misspelling or mis-use of words, which leads to a gap between the ideal query in a user's mind and the ill-formed query received by the search engine. Second, The linguistic structure of a query is never explicitly observed. A user query is usually short and ambiguous. It often has no standard grammar or has idiosyncratic grammar. Further, there is usually no capitalization and punctuation in a query. Thus the lack of linguistic and structure makes it hard to infer the semantics of a query by adopting the traditional Natural Language Processing techniques. Third, the intention of a user query is very difficult to infer in some complex situations. One example is partial query. The user would ask the search engine for suggesting the query completion dynamically in real time giving a short prefix.

To systematically improve query intent understanding, I propose a conceptual framework where there are different levels of query understanding. In this framework, the highest level of query understanding means knowing precisely the user's interests, the complete linguistic and semantic structure of the query, and the temporal/spatial constraints. In the ideal case, a query can be transformed into an equivalent SQL-like structural query that rigorously defines all aspects of the user's intention. Information retrieval with such a query representation will be similar to searching against the web database with structural query, and can thus be much more accurate than the current retrieval paradigm which depends on the bag-of-word query representation. However, we may not always be able to infer such a deep understanding of a query accurately, thus lower levels of query understanding would also be required to improve robustness. These levels of query intent understanding are listed as follows:

- **Query Alteration**. Queries issued by users usually contain errors and mis-used words/phrases. Although a user might have a clear intent in her mind, inferring the query s intent in this case becomes difficult because of the edit distance or vocabulary gap between the user's ideal query and the query issued to the search engine. Query reformulation is to automatically find alternative forms of a query that eliminate or reduce such gap. Effective query reformulation can help improving information retrieval in two ways. First, it can help inferring the user's intent even if the query is ill-formed. Second, retrieval models can be enhanced by transforming the query to its top reformulations. There are several types of query alterations, including query spelling correction which is to transfer a misspelled query into the correct form, query expansion which to expand the original query by adding related terms so as to make the query intent more evident, and query rewriting which is to transform the original query into a new form that is more representative. Note that on this level of query intent understanding, a user query is represented by bag of words. This query representation has been proved to be very effective for ad-hoc retrieval where relevant documents are returned to a key-word based query.

- **Latent Query Linguistic Signal Discovery**. Different from well-formed Natural Language texts, web search queries are characterized by lack of explicit linguistic structures such as quotation, capitalization, punctuation and standard grammar. This level of query intent understanding aims at discovering the latent linguistic signals like segmentation (phrase boundary), part-of-speech tagging, capitalization *etc.* of a user query. Successful discovery of latent query linguistic signals can help improve the ad-hoc retrieval. For example, query segmentation can reduce the number of candidate terms to score where terms are generalized to phrases instead of individual words. It will also improve the scoring function by leveraging the proximity constraint in a segmentation. Furthermore, This level of query intent understanding builds the foundation for deeper level understanding and representation of query intent – semantic understanding and interpretation of queries.

- **Semantic Annotation of Queries**. The bag-of-words query representation has been a great success in document retrieval where relevant documents are returned to the query. However, as the search has been expanded to many other types of applications, bag-of-words representation is not sufficient to support the requirements of these applications. One such application is entity search. Nowadays the web contains a wealth of structured data, such as various entity databases, web tables, *etc.* There is a growing trend in search engines to match unstructured user queries to these structured data sources. In the entity-centric search, schema annotation of queries is required to match the schema of the structured data sources. Another application is to present direct answer and facts to queries. Examples include the instant answer box of modern web search engines, and the computational knowledge engines like Wolfram Alpha. In order to understand the intention of the user and judge whether a direct answer should be triggered, the query has to be transferred to semantic components. And these components are further precessed and matched against the knowledge bases. This level of query intent understanding goes beyond the bag-of-words representation. It aims at deciphering the semantic structure of queries, that is the meaning of every piece of query segment and their relation. It involves tasks such as target type classification which is to infer the category/domain of the query, name entity and attribute recognition and disambiguation, schema matching to a catalog which is to match a query to predefined catalog schema like product tables, semantic role labelling in queries *etc.*

- **Dynamic Understanding and Representation of Queries**. In the above levels of query understanding, it is assumed that we have the whole query in advance. However in some application scenarios this assumption may not hold. For example, in the application of query auto-completion, the objective is to predict user's preferred query based on partial query prefix dynamically in real time. Because only very limited information is exposed from the partial query, other contextual information such as user's short and long term behavior must be taken into account to predict the real user information need. In this case, to understand and represent user's intent dynamically given partial query may need to take into account contextual information such as the user query history, user's short term interaction with search engine, external knowledge from other knowledge bases *etc.*

In this thesis, I will present several studies that I have conducted on addressing important research questions for advancing different levels of query understanding. First, as a major type of query reformulation, I addressed the query spelling correction problem by modelling all major types of spelling errors with a generalized Hidden Markov Model [56]. Subsequently a Latent Structural SVM model was proposed to model the same problem [30]. Second, query segmentation is the most important type of query linguistic signals. I proposed a probabilistic model to identify the query segmentations using clickthrough data [57]. Third, synonym finding is an important challenge for semantic annotation of queries. I proposed a compact clustering framework to mine entity attribute synonyms for a set of inputs jointly with multiple information sources [59]. Subsequently I applied a similar clustering framework to detect synonymous query templates for attribute intents [58]. Finally, in the direction of dynamic understanding and representation of queries, I introduced a new kind of user behavior called “Horizontal Skipping Bias”, which is unique to the query auto-completion process. I then proposed a two-dimensional click model to model the query auto-completion process. My researches in this thesis are summarized as follows.

- **Query Spelling Correction by a Generalized Hidden Markov Model**. Queries issued by web search engine users usually contain errors. Inferring the query’s intent in this case becomes difficult. As an important type of query alteration, query correction aims at transforming the potentially misspelled query into its correct form. Existing methods in the literature have two major drawbacks. First, they are unable to handle important types of spelling errors, such as concatenation and splitting. Second, they usually employ a heuristic filtering step to select a working set of top-K candidates for final scoring, leading to non-optimal predictions. In [56] I addressed both limitations by proposing a novel generalized Hidden Markov Model with discriminative training that can not only handle all the major types of spelling errors in a single unified framework, but also efficiently evaluate all the candidate corrections to ensure the finding of a globally optimal correction. I had also built a query speller system called CloudSpeller [55], which won the second place in the Microsoft Speller Challenge [4].

- **Query Segmentation using Clicktrhough**. One difficulty toward deeper level query intent understanding is that web search query is usually lack of explicit linguistic signals such as quotation, capitalization, punctuation and standard grammar. Successful detection of such latent query linguistic signals can help improve the retrieval performance. I addressed the identification of the most important type of query linguistic signals – query segmentation. Existing segmentation models either use labeled data to predict the segmentation boundaries, for which the training data is expensive to collect, or employ unsupervised strategy only based on a large text corpus, which might be inaccurate because of the lack of relevant information. To address these limitations, in [57] I proposed a probabilistic model to exploit click-through data for query segmentation. I further studied how to properly interpret the segmentation results and utilize them to improve retrieval accuracy. Specifically, I proposed an integrated language model based on the standard bigram language model to utilize the probabilistic structure obtained through query segmentation.

- **Entity Attribute Synonyms Mining**. If the query and structured data sources are all written in well-formed texts, the task of semantic annotation of queries is manageable. However a big challenge of query semantic annotation is to handle the variation of text expression. Users may use many different alternative forms of the same entity when they input a query. For example, people usually issue "LOTR show time" instead of "lord of the rings show time" as typing the short form of the entity is more convenient. Discovering such alternative surface forms of entities and attributes is crucial for improving query semantic annotation thus advancing the query intent understanding and retrieval. However, most previous approaches only focused on utilizing a single feature, such as distributional similarity or query-entity clicks. In addition, previous methods usually look for synonyms one entity at a time, ignoring the information provided by the entire set of inputs. In [59] I proposed a compact clustering framework to identify synonyms for a set of entity attributes jointly. Signals from multiple sources of information are integrated for finding synonyms.

- **Modeling Query Auto-completion by a Two-dimensional Click Model**. Notice that the above levels of query understanding is static, meaning that we have to know the entire query in advance. However in many scenarios it is not possible: the users want to be assisted when they just give a tiny amount of query hint, which is called dynamic query understanding. One example is to predict users intended queries based on partial queries in the task of query auto-completion. For this purpose, in

[54] I have introduced a new kind of user behavior called “Horizontal Skipping Bias”, which is unique to the query auto-completion process. Based on this novel discovery, Ive proposed a novel Two-Dimensional Click Model to model the users behavior in QAC and the resulting relevance model significantly improves the relevance ranking in QAC than most of the existing click models.

The rest of the proposal is organized as follows. In Chapter 2, I review the studies related to this thesis. Chapter 3, 4, 5, and 6 present the approaches for Query Spelling Correction, Query Segmentation, Entity Attribute Synonyms Mining and Query Auto-Completion respectively. Finally I will summarize my thesis works and point out some potential future directions in Chapter 7.

Chapter 2

Related Work

In this chapter, we review related work in existing literature on these topics: (1)Query spelling correction and query reformulation; (2) Query segmentation; (3) Synonym mining; (4) Query Auto-completion; (5) Query log and other resources.

2.1 Query Spelling Correction and Query Reformulation

Query spelling correction has long been an important research topic [50]. Traditional spellers focused on dealing with non-word errors caused by misspelling a known word as an invalid word form. A common strategy at that time was to utilize a trusted lexicon and certain distance measures, such as Levenshtein distance [52]. The size of lexicon in traditional spellers is usually small due to the high cost of manual construction of lexicon. Consequently, many valid word forms such as human names and neologisms are rarely included in the lexicon. Later, statistical generative models were introduced for spelling correction, in which the error model and n-gram language model are identified as two critical components. Brill and Moore demonstrated that a better statistical error model is crucial for improving a speller's accuracy [17]. But building such an error model requires a large set of manually annotated word correction pairs, which is expensive to obtain. Whitelaw *et al.* alleviated this problem by leveraging the Web to automatically discover the misspelled/corrected word pairs [97].

With the advent of the Web, the research on spelling correction has received much more attention, particularly on the correction of search engine queries. Many research challenges are raised, which are non-existent in traditional settings of spelling correction. More specifically, there are many more types of spelling errors in search queries, such as misspelling, concatenation/splitting of query words, and misuse of legitimate yet inappropriate words. Research in this direction includes utilizing large web corpora and query log [23, 27, 6], training phrase-based error model from clickthrough data [82] and developing additional features [32]. However, two important challenges are under addressed in these approaches, i.e., correcting splitting and concatenation errors, and ensuring complete search in the candidate space to evaluate an effective scoring function.

Query spelling correction also shares similarities with many other NLP tasks, such as speech recognition and machine translation. In many of these applications, HMM has been found very useful [46, 89].

Query reformulation is a broader topic which naturally subsumes query spelling correction. Beside correcting the misspelled query, query reformulation also need to modify the ineffective query so that it could be more suitable for the search intent. For this purpose, many research topics have been studied. Query expansion expands the query with additional terms to enrich the query formulation [99, 72, 68]. Other query reformulation methods intend to replace the inappropriate query terms with effective keywords to bridge the vocabulary gaps [95]. Particularly, there is research attempt [35] to use a unified model to do a broad set of query refinements such as correction, segmentation and even stemming. However, it treats query correction and splitting/merging as separate tasks, which is not true for real world queries. Also, it has very limited ability for query correction. For example, it only allows one letter difference in deletion/insertion/substitution errors.

2.2 Query Segmentation

Query segmentation models have been studied in recent literature [43, 47, 15, 101, 84, 36]. Initially, the mutual information (MI) between adjacent words in a query is employed to segment queries with a cutoff [43, 47].The major drawback of MI based methods is that they are unable to detect multi-word or phrase based dependencies. Compared with MI based models, supervised query segmentation approaches can achieve higher accuracies [15, 101]. For example, by considering every boundary between two consecutive query words as a binary decision variable, Bersgma and Wang [15] trains the weights of a linear decision function with a set of syntactic and shallow semantic features extracted from the labeled data. However, its focus on noun phrase features may not be appropriate for the segmentation of web queries. Furthermore, acquiring training labels demands a great deal of manual effort that may not scale to the web. As another supervised learning approach, Yu and Shi [101] applies conditional random fields to obtain good query segmentation performance. However, it relies on field information features specific to databases, not available for general unstructured web queries. Moreover, the evaluation was conducted only on synthetic data, which is less desirable than real query data.

Tan and Peng [84] introduce a generative model in the unsupervised setting by adopting n-gram frequency counts from a large text corpus and computing the segment scores via expectation maximization (EM). It also utilizes Wikipedia as another term in the minimum description length objective function. Similar probabilistic model is also proposed in [102], but this model focuses in parsing noun phrases thus not generally applicable to web queries.

Our work is also related to the retrieval models that capture higher order dependencies of query terms. There are several research attempts to incorporate term dependency in query or document to retrieval models [65]. For example, some attempts have been made to add proximity heuristics to the vector space model or generative query LM model [67, 86]. However these methods rely on heuristics, which is not a principled way of incorporating term dependency. More unified higher-order language models have been studied by Srikanth *et al.* (Biterm LM) [81]. However, their assumption that every position is dependent is too strong. In fact, the word dependency is stronger within a semantic unit than across the unit, which is what we assume in our work. LM with query syntactics [33] assumes a structure on the query, but they are too complex to estimate accurately. More important, the query syntactic models usually take only the top (most likely) query structure in the modeling process. However, it is more appropriate to assess the probability for all possible segmentation if multiple structures have comparable probabilities to represent the query.

2.3 Synonym Mining

There is a rich body of work on the general topic of automatic synonym discovery. This research topic can be divided into sub-areas, including finding word synonyms, entity/attribute synonyms, and related query identification. Identifying word level synonyms from text is a traditional topic in the NLP community. Such synonyms can be discovered using simple dictionary based methods such as WordNet and Wikipedia redirects; distributional similarity based methods [60, 61]; and approximate string matching approaches [69]. In this work, we are interested in finding entity attribute synonyms, which usually have more domain context than plain words.

Researchers have employed several similarity metrics to find synonyms from web data. Such similarities include distributional similarity [60, 61], coclick similarity [24, 20], pointwise mutual information [88], and co-occurrence statistics [11]. Unlike these works, our work introduces a novel similarity metric called categorical pattern similarity for jointly finding synonyms from a set of attributes.

Although several similarity metrics are introduced to find synonyms, most previous approaches use only a single metric in their model. [20] tries to combine multiple metrics, however they manually choose a set of thresholds for individual metrics, which leads to a high precision but potentially low recall approach.

2.4 Query Auto-Completion

Query auto-completion is the search process of preferred queries given the issued prefix of a user. Most of the existing works focus on relevance ranking. For this purpose, traditional QAC engines rely on query popularity counts. However it's impossible to return queries matching a user's specific preference such as location and freshness in time *etc.* Recent QAC models employ learning-based strategy that incorporates several global and personal features [10, 79]. But there is no consensus of how to optimally train the relevance model.

The QAC process is very personal in nature, so it's almost impossible to obtain a labeled dataset by third-party annotation. Existing methods use the clicks as a relevance surrogate, and train a model trying to maximize the clicks. The straightforward way is to only utilize the data in the last prefix, and use the skip-above as well as the skip-next hypothesis to obtain a set of labels. Then we could use the learning-based algorithms to train a model that linearly combines a set of features. Most recently [79] introduces a different strategy, which exploits all suggested queries for all simulated prefixes of the clicked query. However, this automatic labeling strategy might be problematic, since it may introduce many false negative examples where the user skips looking down the list. If she had to examine the list, she would have clicked a query. So there is a lot of uncertainty in the labeled examples introduced by this method.

Besides relevance modeling, there are previous works addressing different aspects of QAC. For example, [12, 38] studied the space efficiency of index for QAC. [96, 41] investigated the efficient algorithms for QAC. [29] addressed the problem of suggesting query completions even if the prefix is mis-spelled. And [8] studied the context-sensitive QAC for mobile search.

The QAC is a complex process where a user goes through a series of interactions with the QAC engine before clicking on a query. Deciphering the user behavior in QAC is an interesting and challenging task. Despite of its importance, little research is done on this direction, mainly because of the lack of suitable QAC log. It is in this work we first collect a high-resolution QAC log and attempt to model the user behaviors.

Modeling the query auto-completion is closely related to Click Models. In the field of document retrieval, the main purpose for modeling a user's clicks is to infer the intrinsic relevance between the query and document by explaining the positional bias. The position bias assumption was first introduced by Granka *et al.* [34], stating that a document on higher rank tends to attract more clicks. Richardson et al. [75] attempted to model the true relevance of documents in lower positions by imposing a multiplicative factor. Later examination hypothesis is formalized in [26], with a key assumption (Cascade Assumption) that a user will click on a document if and only if that document has been examined and it is relevant to the query. Later, several extensions were proposed, such as the User Browsing Model (UBM) [31], Bayesian Browsing Model [62], General Click Model [105] and Dynamic Bayesian Network model (DBN) [22]. Despite the abundance of click models, no existing click models can directly apply to QAC without considerable modification. The click model most similar to our work is [104], which models users' clicks on a series of queries in a session. However because of the main difference between QAC and document retrieval, our model structure is very different from [104]. To the best of our knowledge, our work is the first click model for modeling the QAC process.

2.5 Query Log and Other Resources

To better understand the query intents we have to leverage users' short and long term interaction with search engine. Such activities are usually recorded in query log. Query log has been utilized for diverse applications such as query segmentation [15, 84], query reformulation [44, 78], relevance ranking [74, 42, 5], query clustering [100] *etc.* Recently modeled the long-term query logs using language models to improve the personalized search

[85, 83]. Other types of large-scale resources can be also exploited to decipher the challenging problem of query intent understanding. These resources include web ngram language model [3], knowledge bases such as FreeBase and wikipedia, large-scale web page corpus such as clueweb [1, 2].

Chapter 3

Query Spelling Correction by a Hidden Markov Model

3.1 Introduction

Queries issued by web search engine users usually contain errors and mis-used words/phrases. Although a user might have a clear intent in her mind, inferring the querys intent in this case becomes difficult because of the edit distance or vocabulary gap between the user's ideal query and the query issued to the search engine. Query reformulation is to automatically find alternative forms of a query that eliminate or reduce such gap. Effective query reformulation has been proved to be very effective in improving the performance of information retrieval. There are several types of query reformulations, including query spelling correction, query expansion, query rewriting *etc.* In this chapter we focus on an important type of query reformulation – query spelling correction.

The ability to automatically correct potentially misspelled queries has become an indispensable component of modern search engines. People make errors in spelling frequently. Particularly, search engine users are more likely to commit misspellings in their queries as they are in most scenarios exploring unfamiliar contents. Automatic spelling correction for queries helps the search engine to better understand the users' intents and can therefore improve the quality of search experience. However, query spelling is not an easy task, especially under the strict efficiency constraint. In Table 3.1 we summarize major types of misspellings in real search engine queries. Users not only make typos on single words, (insertion, deletion and substitution), but can also easily mess up with word boundaries (concatenation and splitting). Moreover, different types of misspelling could be committed in the same query, making it even harder to correct. Unfortunately, no existing query spelling correction

Table 3.1: Major Types of Query Spelling Errors

Type		Example	Correction
In-Word	Insertion	esspresso	espresso
	Deletion	vollyball	volleyball
	Substitution	comtemplate	contemplate
	Mis-use	capital hill	capitol hill
Cross-Word	Concatenation	intermilan	inter milan
	Splitting	power point	powerpoint

approaches in the literature are able to correct all major types of errors, especially for correcting splitting and concatenation errors. To the best of my knowledge, the only work that can potentially address this problem is [35] in which a Conditional Random Field (CRF) model is proposed to handle a broad set of query refinements. However, this work considers query correction and splitting/merging as different tasks, hence it is unable to correct queries with mixed types of errors, such as substitution and splitting errors in one query. In fact splitting and merging are two important error types in query spelling correction, and a major research challenge of query spelling correction is to accurately correct all major types of errors simultaneously.

Another major difficulty in automatic query spelling correction is the huge search space. Theoretically, any sequence of characters could potentially be the correction of a misspelled query. It is clearly intractable to enumerate and evaluate all possible sequences for the purpose of finding the correct query. Thus a more feasible strategy is to search in a space of all combinations of candidate words that are in a neighborhood of each query word based on editing distance. The assumption is that a user's spelling error of each single word is unlikely too dramatic, thus the correction is most likely in the neighborhood by editing distance. Unfortunately, even in this restricted space, the current approaches still cannot enumerate and evaluate all the candidates because their scoring functions involve complex features that are expensive to compute. As a result, a separate filtering step must first be used to prune the search space so that the final scoring can be done on a small working set of candidates. Take [32] as a two-stage method example, in the first stage, a Viterbi or A* search algorithm is used to generate a small set of most promising candidates, and in the second stage different types of features of the candidates are computed and a ranker is employed to score the candidates. However, this two-stage strategy has a major drawback in computing the complete working set. Since the filtering stage uses a non-optimal objective function to ensure efficiency, it is quite possible that the best candidate is filtered out in the first stage, especially because we cannot afford a large working set since the correction must be done online while a user is entering a query. The inability of searching the complete space of candidates leads to non-optimal correction accuracy.

In this chapter, we propose a generalized Hidden Markov Model (gHMM) for query spelling correction that can address deficiencies of the existing approaches discussed above. The proposed gHMM can model all major types of spelling errors, thus enabling consideration of multiple types of errors in query spelling correction. In the proposed gHMM, the hidden states represent the correct forms of words, and the outcomes are the observed (potentially) misspelled terms. In addition, each state is associated with a type, indicating merging, splitting or in-word transformation operation. The proposed HMM is generalized in the sense that it would allow adjustment of both emission probabilities and transition probabilities to accommodate the non-optimal parameter estimation. Unfortunately, such an extension of HMM makes it impossible to use a standard EM algorithm for parameter estimation. To solve this problem, we propose a perceptron-based discriminative training method to train the parameters in the HMM.

Moreover, a Viterbi-like search algorithm for top-K paths is designed to efficiently obtain a small number of highly confident correction candidates. This algorithm can handle splitting/merging of multiple words. It takes into account major types of local features such as error model, language model, and state type information. The error model is trained on a large set of query correction pairs from the web. And web scale language model is obtained by leveraging the Microsoft Web N-gram service [3].

We conducted extensive evaluation on our proposed gHMM. For this purpose, we have constructed a query correction dataset from real search logs, which has been made publicly available. Experimental results verify that the gHMM can effectively correct all major types of query spelling errors. It also reveal that the gHMM can run as efficient as the common used noisy channel model, while it achieves much better results for obtaining the candidate space of query corrections. Therefore, in addition to being used as standing alone query correction module, the proposed gHMM can also be used as a more effective first-stage filtering module to more effectively support any other complicated scoring functions such as those using complex global features.

3.2 Problem Setup and Challenges

Formally, let Σ be the alphabet of a language and $L \subset \Sigma^+$ be a large lexicon of the language. We define the query spelling correction problem as:

Given a query $q \in \Sigma^+$, generate top-K most effective corrections $Y = (y^1, y^2, ..., y^k)$ where $y^i \in L^+$ is a candidate correction, and Y is sorted according to the probability of y^i being the correct spelling of the target query.

It is worth noting that from a search engine perspective, the ideal output Y' should be sorted according to the probability of y^i retrieving the most satisfying results in search. However, in practice it is very difficult to measure the satisfaction as unlike in *ad hoc* retrieval where the query is given in its correct form, here the real query is unknown. As a result, different corrections could simply lead to queries with different meanings and it would be very subjective to determine which query actually satisfies the user. In this work, we are mostly concerned with the lexical and semantic correctness of queries with the assumption that correction of mis-spelled query terms most likely would lead to improved retrieval accuracy.

The problem of query spelling correction is significantly harder than the traditional spelling correction. Previous researches show that approximately 10-15% of search queries contain spelling errors [27]. First, it is difficult to cover all the different types of errors. The spelling errors generally fall into one of the following four categories: (1) in-word transformation, e.g. insertion, deletion, misspelling of characters. This type of error is most frequent in web queries, and it is not uncommon that up to 3 or 4 letters are misspelled; (2) mis-use of valid word, e.g. "persian golf" $\rightarrow$ "persian gulf". It is also a type of in-word tranformation errors; (3) concatenation of multiple words, e.g. "unitedstatesofamerica" $\rightarrow$ "united states of america"; (4) splitting a word into parts, e.g. "power point slides" $\rightarrow$ "powerpoint slides". Among all these types, the splitting and concatenation errors are especially challenging to correct. Indeed, no existing approaches in the academic literature can correct these two types of errors. Yet, it's important to correct all types of errors because users might commit different types of errors or even commit these errors at the same time. A main goal of this work is to develop a new HMM framework that can model and correct all major types of errors including splitting and concatenation.

Second, it is difficult to ensure complete search of all the candidate space because the candidate space is very large. The existing work addresses this challenge by using a two-stage method, which searches for a small set of candidates with simple scoring functions and do re-ranking on top of these candidates. Unfortunately, the simple scoring function used in the first stage cannot ensure that the nominated candidate corrections in the first stage always contain the best correction, thus no matter how effective the final scoring function is, we may miss the best correction simply because of the use of two separate stages. In this chapter, we address this challenge by developing a generalized HMM that can both be efficiently scored to ensure complete search in the candidate space and accurately correct all types of errors in a unified way.

3.3 A Generalized HMM for Query Spelling Correction

Our algorithm accepts a query as input, and then generates a small list of ranked corrections as output by a generalized Hidden Markov Model (gHMM). It is trained by a discriminative method with labeled spelling examples. Given a query, it scores candidate spelling corrections in a one-stage fashion and outputs the top-K corrections, without using a re-ranking strategy. Other components of our algorithm include a large clean lexicon, the error model and the language model. In this section we will focus on the gHMM model structure, the discriminative training of it, as well as the efficient computation of spelling corrections.

3.3.1 The gHMM Model Structure

We propose a generalized HMM Model to model the spelling correction problem. We call it a generalized HMM because there are several important differences between it and the standard HMM model which will be explained later. Without loss of generality, let an input query be $q = q_{[1:n]}$ and a corresponding correction be $y = y_{[1:m]}$ where n, m are the length of the query and correction, which might or might not be equal. Here we introduce hidden state sequence $z = z_{[1:n]} = (s_1, s_2, ..., s_n)$ in which z and q have the same length. An individual state s_i is represented by a phrase corresponding to one or more terms in correction $y_{[1:m]}$. Together the phrase representing z is equal to y. Therefore, finding best-K corrections $Y = (y^1, y^2, ..., y^k)$ is equivalent to finding best-K state sequences $Z = (z^1, z^2, ..., z^k)$. In addition, there is a type t associated with each state, indicating the operation such as substitution, splitting, merging *etc*. Also, in order to facilitate the merging state we introduce a NULL state. The NULL state is represented by an empty string, and it doesn't emit any phrase. There can be multiple consecutive NULL states followed by a merging state. Table 3.2 summarizes the state types and the spelling errors they correspond to. Having the hidden states defined, the hypothesized process of observing a mis-spelled query is as follows:

1. sample a state s_1 and state type t_1 from the state space Ω and the type set T;

2. emit a word in q_1, or empty string if the s_1 is a NULL state according to the type specific error model;

3. transit to s_2 with type t_2 according to the state transition distribution, and emit another word, or multiple words in $q_{[1:n]}$ if s_2 is a merging state;

4. continue until the whole (potentially) mis-spelled query q is observed.

Table 3.2: State Types in gHMM

State Type	Operation	Spelling Errors
In-word Transformation	Deletion Insertion Substitution	Insertion Deletion Substitution
Mis-use	Transformation	Word Mis-use
Merging	Merge Multiple Words	Splitting
Splitting	Split one Word to Multiple Words	Concatenation

Figure 3.1 illustrates our gHMM model with a concrete example. In this example, there are three potential errors with different error types, e.g. "goverment" $\rightarrow$ "government" (substitution), "home page" $\rightarrow$ "homepage" (splitting), "illinoisstate" $\rightarrow$ "illinois state" (concatenation). The state path shown in Figure 3.1 is one of the state sequences that can generate the query. Take state s_3 for example, s_3 is represented by phrase *homepage*. Since s_3 is a merging state, it emits a phrase *home page* with probability $P(\textit{home page}|\textit{homepage})$. And s_3 is transited from state s_2 with probability $P(s_3|s_2)$. With this model, we are able to come up with arbitrary corrections instead of limiting ourselves to an incomprehensive set of queries from query log. By simultaneously modeling the misspellings on word boundaries, we are able to correct the query in a more integrated manner.

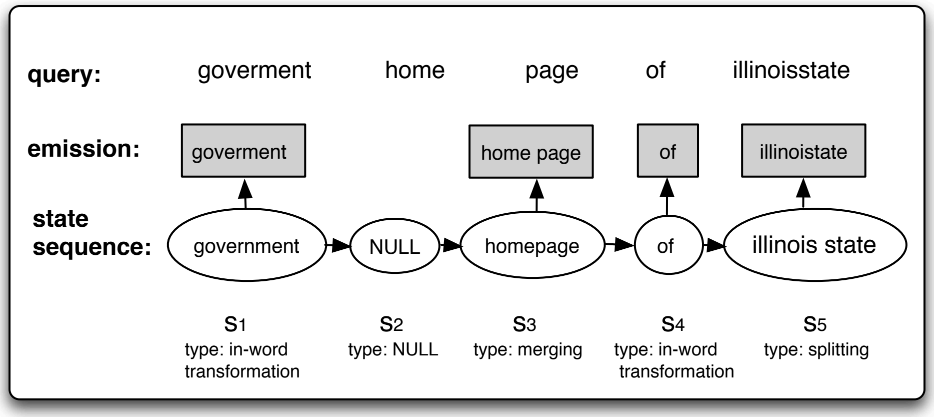


Figure 3.1: Illustration of the gHMM Model

3.3.2 Generalization of HMM Scoring Function

For a standard HMM [73], let $\theta = \{A, B, \pi\}$ be the model parameters of the HMM, representing the transition probability, emission probabilities and initial state probabilities respectively. Given a list of query words $q_{[1:n]}$ (obtained by splitting empty spaces), the state sequence $z^* = (s_1^*, s_2^*, ..., s_n^*)$ that best explains $q_{[1:n]}$ can be calculated by:

$$z^* = \arg\max_z P(z|q_{[1:n]}, A, B, \pi) \tag{3.1}$$

However, theoretically the phrase in a state can be chosen arbitrarily, so estimating $\{A, B, \pi\}$ is such a large space is almost impossible in the standard HMM framework. In order to overcome this difficulty, the generalized Hidden Markov Model proposed in this work generalizes the standard HMM as follows: (1) gHMM introduces state type for each state, which indicates the correction operations and can reduce the search space effectively; (2) it adopts feature functions to parameterize the measurement of probability of a state sequence given a query. Such treatment can not only map the transition and emission probabilities to feature functions with a small set of parameters, but can also add additional feature functions such as the ones incorporating state type information. Another important benefit of the feature function representation is that we can use discriminative training on the model with labeled spelling corrections, which will lead to a more accurate estimation of the parameters.

Formally, in our gHMM model, there is an one-to-one relationship between states in a state sequence and words in the original query. For a given query $q = q_{[1:n]}$ and the sequence of states $z = (s_1, s_2, ..., s_n)$, we define a context h_i for every state in which an individual correction decision is made. The context is defined as $h_i =< s_{i-1}, t_{i-1}, s_i, t_i, q_{[1:n]} >$ where $s_{i-1}, t_{i-1}, s_i, t_i$ are the previous and current state and type decisions and $q_{[1:n]}$ are all query words.

The generalized HMM model measures the probability of a state sequence by defining feature vectors on the context-state pairs. A feature vector is a function that maps a context-state pair to a d-dimensional vector. Each component of the feature vector is an arbitrary function operated on (h, z). Particularly, in this study we define 2 kinds of feature vectors, one is $\phi_j(s_{i-1}, t_{i-1}, s_i, t_i), j = 1...d$, which measures the interdependency of adjacent states. We can map this function to a kind of transition probability measurement. The other kind of feature function, $f_k(s_i, t_i, q_{[1:n]}), k = 1...d'$ measures the dependency of the state and its observation. We can consider it as a kind of emission probability in the standard HMM point of view. Such feature vector representation of HMM is introduced by Collins [25] and successfully applied to the POS tagging problem.

Specifically, we have designed several feature functions as follows: we define a function of $\phi(s_{i-1}, t_{i-1}, s_i, t_i)$ as

$$\phi_1(s_{i-1}, t_{i-1}, s_i, t_i) = logP_{LM}(s_i|s_{i-1}, t_{i-1}, t_i) \tag{3.2}$$

to measure the language model probabilities of two consecutive states. Where $P_{LM}(s_i|s_{i-1})$ is the bigram probability calculated by using Microsoft Web N-gram Service [3]. The computation of $P_{LM}(s_i|s_{i-1})$ may depend on the state types, such as in a merging state.

We have also defined a set of functions in the form of $f_k(s_i, t_i, q_{[1:n]})$, which are dependent on the query words and state type, measuring the emission probability of a state. For example, we define

$$f_1(s_i, t_i, q_{[1:n]}) = \begin{cases} logP_{err}(s_i, q_i) & \text{if } q_i \text{ is in-word transformed to } s_i \textbf{ and } q_i \notin \text{Lexicon } L \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

as a function measuring the emission probability given the state type is in-word transformation and q_i is out of dictionary. e.g. "goverment" → "government". $P_{err}(s_i, q_i)$ is the emission probability computed by an error model which measures the probability of mistyping "government" to "goverment".

$$f_2(s_i, t_i, q_{[1:n]}) = \begin{cases} logP_{err}(s_i, q_i) & \text{if } t_i \text{ is splitting } \textbf{and } q_i \in \text{Lexicon } L \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

to capture the emission probability if the state is of splitting type and q_i is in dictionary. e.g. "homepage" → "home page".

$$f_3(s_i, t_i, q_{[1:n]}) = \begin{cases} logP_{err}(s, q_i) & \text{if } t_i \text{ is Mis-use } \textbf{and } q_i \in \text{Lexicon } L \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

to get the emission probability if a valid word is transformed to another valid word.

Note that in Equation (3.3), (3.4), and (3.5), we use the same error model $P_{err}(s_i, q_i)$ (see Section **??** for detail) to model the emission probabilities from merging, splitting errors *etc.* in the same way as in-word transformation errors. However we assign different weights to the transformation probabilities resulted from different error types via discriminative training on a set of labeled query-correction pairs.

Overall, we have designed a set of feature functions that are all relied on local dependencies, ensuring that the top-K state sequences can be computed efficiently by Dynamic Programming.

After establishing the feature vector representation, the log-probability of a state sequence and its corresponding types $logP(z, t|q_{[1:n]})$ is proportional to:

$$\begin{aligned} Score(z,t) = &\sum_{i=1}^{n}\sum_{j=1}^{d}\lambda_j\phi_j(s_{i-1},t_{i-1},s_i,t_i) \\ &+\sum_{i=1}^{n}\sum_{k=1}^{d'}\mu_k f_k(s_i,t_i,q_{[1:n]}) \end{aligned} \tag{3.6}$$

where λ_j, μ_k are the component coefficients needed to be estimated. And the best state sequence can be found by:

$$z^*t^* = \arg\max_{z,t} Score(z,t) \tag{3.7}$$

Note that the form of $Score(z, t)$ is similar to the objective function of a Conditional Random Field model [51], but with an important difference that there is no normalization terms in our model. Such difference also enables the efficient search of top-K state sequences (equivalent to top-K corrections) using Dynamic Programming, which will be introduced shortly.

3.3.3 Discriminative Training

Motivated by ideas introduced in [25], we propose a perceptron algorithm to train the gHMM model. To the best of our knowledge, this is the first attempt to use discriminative approach to train a HMM on the problem of query spelling correction. Now we describe how to estimate the parameters λ_j, μ_k from a set of <query, spelling correction> pairs. The estimation procedure follows the perceptron learning framework. Take the λ_j for example. We first set all the λ_j at random. For each query q, we search for the most likely state sequence with types $z^i_{[1:n_i]}, t^i_{[1:n_i]}$ using the current parameter settings. Such search process is described in Algorithm 2 by setting $K = 1$. After that, if the best decoded sequence is not correct, we update λ_j by simple addition: we promote the amount of λ_j by adding up ϕ_j values computed between the query and labeled correction y', and demote the amount of λ_j by the sum of all ϕ_j values computed between the query and the top-ranked predictions. We repeat this process for several iterations until converge. Finally in step 11 and 12, we average all $\lambda_j^{o,i}$ in each iteration to get the final estimate of λ_j, where $\lambda_j^{o,i}$ is the stored value for the parameter λ_j after i's training example is processed in iteration o. Similar procedures can apply to μ_k. The detailed steps are listed in Algorithm *1*. Note that in step 7 and 8 the feature functions $\phi_j(q^i, y'^i, t'^i)$ and $f_k(q^i, y'^i, t'^i)$ depend on unknown types t'^i that are inferred by computing the best word-level alignment between q^i and y'^i. This discriminative training algorithm will converge after several iterations.

3.3.4 Query Correction Computation

Once the optimal parameters are obtained by the discriminative training procedure introduced above, the final top-K corrections can be directly computed, avoiding the need for a separate stage of candidate re-ranking. Because the feature functions are only relied on local dependencies, it enables the efficient search of top-K corrections via Dynamic Programming. This procedure involves three major steps: (1) candidate states generation; (2) score function evaluation; (3) filtering.

At the first step, for each word in query q, we generate a set of state candidates with types. The phrase representations in such states are in Lexicon L and within editing distance δ from the query word. Then a set of state sequences are created by combining these states.

Algorithm 1: Discriminative Training of gHMM

```
input  : A set of ¡query, spelling correction¿ pairs
         $q^i_{[1:n_i]}, y'^i_{[1:m_i]}$ for $i = 1...n$
output: Optimal estimate of $\hat{\lambda}_j, \hat{\mu}_k$, where $j \in \{1, ..., d\}, k \in \{1, ..., d'\}$

1  Init Set $\hat{\lambda}_j, \hat{\mu}_k$ to random numbers;
2  for $o \leftarrow 1$ to $O$ do
3      for $i \leftarrow 1$ to $n$ do
           /* identify the best state sequence and the associated types of the
              i'th query with the current parameters via Algorithm 2:        */
4          $z^i_{[1:n_i]}, t^i_{[1:n_i]} = \arg\max_{u_{[1:n_i]}, t_{[1:n_i]}} Score(u, t)$
           /* where $u_{[1:n_i]} \in \mathbb{S}^{n_i}, \mathbb{S}^{n_i}$ is all possible state sequences given $q^i_{[1:n_i]}$   */
5          if $z^i_{[1:n_i]} \neq y'^i_{[1:m_i]}$ then
6              update and store every $\lambda_j, \mu_k$ according to:
7              $\lambda_j = \lambda_j + \sum_{i=1}^{n_i} \phi_j(q^i, y'^i, t'^i) - \sum_{i=1}^{n_i} \phi_j(q^i, z^i, t^i)$
8              $\mu_k = \mu_k + \sum_{i=1}^{n_i} f_k(q^i, y'^i, t'^i) - \sum_{i=1}^{n_i} f_k(q^i, z^i, t^i)$
9          else
10             Do nothing
   /* Average the final parameters by:                                      */
11 $\hat{\lambda}_j = \sum_{o=1}^{O} \sum_{i=1}^{n} \lambda_j^{o,i}/nO$, where $j \in \{1, ..., d\}$
12 $\hat{\mu}_k = \sum_{o=1}^{O} \sum_{i=1}^{n} \mu_k^{o,i}/nO$, where $k \in \{1, ..., d'\}$
13 return parameters $\hat{\lambda}_j, \hat{\mu}_k$;
```

In addition, for each state sequence we have created, we also create another state sequence by adding a NULL state at the end, facilitating a (potential) following merging state. It is important to note that if the δ is too small, it will compromise the final results due to the premature pruning of state sequences. In this work $\delta = 3$ is chosen in order to introduce adequate possible state sequences.

At the score function evaluation step, we update the scores for each state sequence according to *Eq.* (3.6). The evaluation is different for sequence with different ending state types. Firstly, for a sequence ending with a NULL state, we don't evaluate the scoring function. Instead, we only need to keep track of the state representation of its previous state. Secondly, for a sequence ending with a merging state, it merges the previous one or more consecutive NULL states. And the scoring function takes into account the information stored in the previous NULL states. For instance, to $\phi_1(s_{i-1}, t_{i-1} = NULL, s_i, t_i = merging)$, we have

$$\phi_1(s_{i-1}, \text{NULL}, s_i, \text{merging}) = logP_{LM}(s_{i-2}|s_i) \tag{3.8}$$

i.e. skipping the NULL state and pass the previous state representation to the merging state. In this way, we can evaluate the scoring function in multiple consecutive NULL states followed by a merging state, which enables the correction by merging multiple query words. Thirdly, for a sequence ending with a splitting state, the score is accumulated by all bigrams within the splitting state. For example,

$$\begin{aligned} &\phi_1(s_{i-1}, t_{i-1}, s_i, t_i = splitting) \\ &= logP_{LM}(w_1|s_{i-1}) + \sum_{j=1}^{k-1} logP_{LM}(w_{i+1}|w_i) \end{aligned} \tag{3.9}$$

where $s_i = w_1 w_2 ... w_k$. On the other hand, the evaluation of $f_k(s_i, t_i, q_{[1:n]})$ is easier because it is not related to previous states. The error model from the state representation to the query word is used to calculate these functions.

At the final step, we filter most of the state sequences and only keep top-K best state sequences in each position corresponding to each query word. In sum, we have proposed and implemented an algorithm via Dynamic Programming (see Algorithm 2) for efficiently computing top-K state sequences (corrections). If there are n words in a query, and the maximum number of candidate states for each query word is M, the computational complexity for finding top-K corrections is $O(n \cdot K \cdot M^2)$.

Algorithm 2: Decoding Top-K Corrections

```
   input  : A query $q_{[1:n]}$, parameters $\vec{\lambda}$, $\vec{\mu}$
   output: top $K$ state sequences with highest likelihood
   /* $Z[i, s_i]$: top K state sequences for sub-query $q_{[1:i]}$ that ending with state
      $s_i$. For each $z \in Z[i, s_i]$, $phrase$ denotes the representation and $score$
      denotes the likelihood of $z$ given $q_{[1:i]}$.                              */
   /* $Z[i]$: top state sequences for all $Z[i, s_i]$.                              */
1  Init $Z[0] = \{\}$
2  for $i \leftarrow 1$ to $n$ do
       /* for term $q_i$, get all candidate states                                  */
3      $S \leftarrow s_i, \forall s_i : edit\_dist(s_i, q_i) \leq \delta$, $s_i$ has type $s_i.type$
4      for $s_i \in S$ do
5          for $z \in Z[i-1]$ do
6              $a \leftarrow$ new state sequence
7              $a.phrase \leftarrow z.phrase \cup \{s_i\}$
8              update $a.score$ according to $s_i.type$ and Eq. (3.6), Eq. (3.8) and Eq. (3.9)
9              $Z[i, s_i] \leftarrow a$
           /* delay truncation for $NULL$ states                                    */
10         if $s_i.type \neq NULL$ and $i \neq n$ then
11             sort $Z[i, s_i]$ by $score$
12             truncate $Z[i, s_i]$ to size $K$
13 sort $Z[n]$ by $score$
14 truncate $Z[n]$ to size $K$
15 return $Z[n]$;
```

3.4 Experiments and Results

In order to test the effectiveness and efficiency of our proposed gHMM model, in this section we conduct extensive experiments on two web query spelling datasets. We first introduce the datasets, and describe the evaluation metrics we use for evaluation. Then we compare our model with other baselines in terms of accuracy and runtime.

3.4.1 Dataset Preparation

The experiments are conducted on two query spelling correction datasets. One is the TREC dataset based on the publicly available TREC queries (2008 Million Query Track). This dataset contains 5892 queries and the corresponding corrections annotated by the MSR Speller Challenge [4] organizers. There could be more than one plausible corrections for a query. In this dataset only 5.3% of queries are judged as misspelled.

We have also annotated another dataset that contains 4926 MSN queries, where for each query there is at most one correction. Three experts are involved in the annotation process. For each query, we consult the speller from two major search engines (i.e. Google and Bing). If they agree on the returned results (including the case if the query is just unchanged), we take it as the corrected form of the input query. If the results are not the same from the two, as least one human expert will manually annotate the most likely corrected form of the query. Finally, about 13% of queries are judged as misspelled in this dataset, which is close to the error rate of real web queries. This dataset is publicly available to all researchers.

We divide the TREC and MSN datasets into training and test sets evenly. Our gHMM model as well as the baselines are trained on the training sets and finally evaluated on the TREC test set containing 2947 queries and MSN test set containing 2421 queries.

3.4.2 Evaluation Metrics

We evaluate our system based on the evaluation metrics proposed in Microsoft Speller Challenge [3], including expected precision, expected recall and expected F1 measure.

As used in previous discussions, q is a user query and $Y(q) = (y^1, y^2, , y^k)$ is the set of system output with posterior probabilities $P(y^i|q)$. Let $S(q)$ denote the set of plausible spelling variations annotated by the human experts for q. Expected Precision is computed as:

$$precision = \frac{1}{|Q|} \sum_{q \in Q} \sum_{y \in Y(q)} I_p(y, q) P(y|q) \tag{3.10}$$

where $I_p(y, q) = 1$ if $y \in S(q)$, and 0 otherwise. And expected recall is defined as:

$$recall = \frac{1}{|Q|} \sum_{q \in Q} \sum_{a \in S(q)} I_r(Y(q), a) / |S(q)| \tag{3.11}$$

where $I_r(Y(q), a) = 1$ if $a \in Y(q)$ for $a \in S(q)$, and 0 otherwise. Expected F1 measure can be computed as:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{3.12}$$

3.4.3 Overall Effectiveness

We first investigate the overall effectiveness of the gHMM model. For suitable query spelling correction baselines, especially approaches that can handle all types of query spelling errors, we first considered using the CRF model proposed in [35]. This method aims at a broad range of query refinements and hence might be also applicable to query correction. However, we decided not to compare this model for the following reasons. Firstly, we communicated with the authors of [35] and knew that the program is un-reusable. Secondly, as mentioned in Section 5.1 this work suffers from several drawbacks for query spelling correction: (1) it is unable to correct queries with mixed types of errors, such as substitution and splitting errors in one query, because the model treats query correction and splitting/merging as different tasks; (2) this model only allows 1 character error for substitution/insertion/deletion. And the error model is trained on the ¡query, correction¿ examples that only contain 1 character error. Such design is over simplified for real-world queries, in which more than 1 character errors are quite common. In fact, within the queries that contain spelling errors in the MSN dataset, there are about 40.6% of them contain more than 1 character errors. Therefore it is expected model in [35] will have in inferior performance

Because of the reasons stated above, the best baseline method that we can possibly compare with is the system that achieved the best performance in Microsoft Speller Challenge [63] (we call it Lueck-2011). This system relies on candidate corrections from third-party toolkits such as hunspell and Microsoft Wrod Breaker Service [93] , and it re-ranks the candidates by a simple noisy channel model. We communicated with the author and obtained the corrections by running the Web API of this baseline approach. We also include a simple baseline called *Echo*, which is just echoing the original query as the correction response with posterior probability 1. It reflects the basic performance for a naive method. Experiments are conducted on TREC and MSN datasets.

We report the results of all methods in Table 3.3. In this experiment up to top 10 corrections are used in all approaches. The results in Table 3.3 indicate that gHMM outperforms Lueck-2011 significantly on recall and F1 on the TREC dataset. Lueck-2011 has a small advantage on precision, possibly due to the better handling the unchanged queries. On the MSN dataset which is considered harder since it has more misspelled queries, gHMM also achieves high precision of 0.910 and recall of 0.966, which are both significantly better than that of the Lueck-2011 (0.896 and 0.921). On another important performance metric, which measures the F1 on misspelled queries (F1 Mis), gHMM outperforms Lueck-2011 by a large margin (0.550 vs. 0.391 on TREC and 0.566 vs. 0.363 on MSN). These results demonstrate that gHMM is very effective for handling all types of spelling errors in search queries overall.

Table 3.3: gHMM Compared to Baselines

Dataset	Method	Precision	Recall	F1	F1 Mis
TREC	Echo	0.949	0.876	0.911	N/A
	Lueck-2011	**0.963**	0.932	0.947	0.391
	gHMM	0.960	**0.976**	**0.968**	**0.550**
MSN	Echo	0.869	0.869	0.869	N/A
	Lueck-2011	0.896	0.921	0.908	0.363
	gHMM	**0.910**	**0.966**	**0.937**	**0.566**

3.4.4 Results by Error Types

Further, we also break down the results by error types that are manually classified so that we can see more clearly the distribution of types of spelling errors and how well our gHMM model addressing each type of errors. We present the results of this analysis in Table 3.4, only with our model on both datasets. Top 40 corrections are used since it achieves the best results. The breakdown results show that most queries are in the group of "no error", which are easier to handle than the other three types. As a result, the overall excellent performance was mostly because the system performed very well on the "no error" group. Indeed, the system has substantially lower precision on the queries with the other three types of errors. The concatenation errors seem to be the hardest to correct, followed by the splitting errors, and the in-word transformation errors (insertion, deletion and substitution, word mis-use) seem to be relatively easier.

3.4.5 gHMM for Working Set Construction

Since the gHMM can efficiently search in the complete candidate space and compute the top-K spelling corrections in a one-stage manner, it is very interesting to test its effectiveness for

Table 3.4: Results by Spelling Error Type

Dataset	Error Type	% Queries	Precision	Recall	F1
TREC	no error	94.9	0.990	0.982	0.986
	transformation	3.3	0.388	0.840	0.531
	concatenation	1.3	0.348	0.877	0.498
	splitting	0.5	0.500	0.792	0.613
MSN	no error	86.9	0.978	1.0	0.989
	transformation	11.1	0.493	0.762	0.599
	concatenation	1.7	0.150	0.600	0.240
	splitting	0.6	0.429	0.571	0.490

Note: % of queries might sum up to more than 100% since there might be multiple types of errors in one query.

constructing a working set of candidate corrections to enable more complex scoring functions to be used for spelling correction. For this purpose, we compare gHMM with the common used noisy channel model, whose parameters, namely error model probabilities and bigram language probabilities are estimated by the procedure mentioned in previous sections. We use recall to measure the completeness of the constructed working set, because it represents the percentage of true corrections given the number of predicted corrections. Table 3.5 shows the recall according to different number of outputs. It indicates that the recall of gHMM is steadily increasing by a larger number of outputs. By only outputting top-5 corrections, gHMM reaches recall of 0.969 in TREC and 0.964 in MSN. In contrast, the noisy channel model has a substantial gap in term of recall compared to gHMM. This result strongly demonstrates the superior effectiveness of gHMM in constructing a more complete working set of candidate corrections, which can be utilized by other re-ranking approaches which could further improve the correction accuracy.

3.4.6 Efficiency

The runtime requirement of query correction is very stringent. Theoretically, the gHMM with local feature functions can search top-K corrections efficiently by our proposed tok-K Viterbi algorithm. Here we make a directly comparison between the runtime of gHMM and

Table 3.5: gHMM vs. Noisy Channel Model on Recall

Dataset	Method	1	5	10	20	40
TREC	N-C	0.869	0.896	0.899	0.901	0.902
	gHMM	**0.887**	**0.969**	**0.976**	**0.981**	**0.983**
MSN	N-C	0.866	0.870	0.873	0.876	0.886
	gHMM	**0.920**	**0.964**	**0.966**	**0.9667**	**0.967**

Note: N-C refers to the noisy channel model

a basic noisy channel that only needs to compute the error model probabilities and bigram language probabilities. Such a basic model is also implemented with Viterbi algorithm. It is run on a Windows server equipped with 2 Quad-core 64 bit 2.4 GHz CPUs and 8 GB RAM. All necessary bigram language probabilities are crawled from Microsoft Web N-gram Service and cached in local memory. We plot the runtime per query (in milliseconds) according to the number of predicted corrections in Figure 3.2. According to Figure 3.2, the computation of top-1 correction by gHMM is fast (34 ms) if the number of output is set to 1. It increases as the number of output increases because the search space is increased. Interestingly, the runtime of gHMM and the noisy channel model is of the same order of magnitude. This empirical evidence confirms the theoretical result that top-K spelling corrections can be computed efficiently via our proposed top-K Viterbi algorithm.

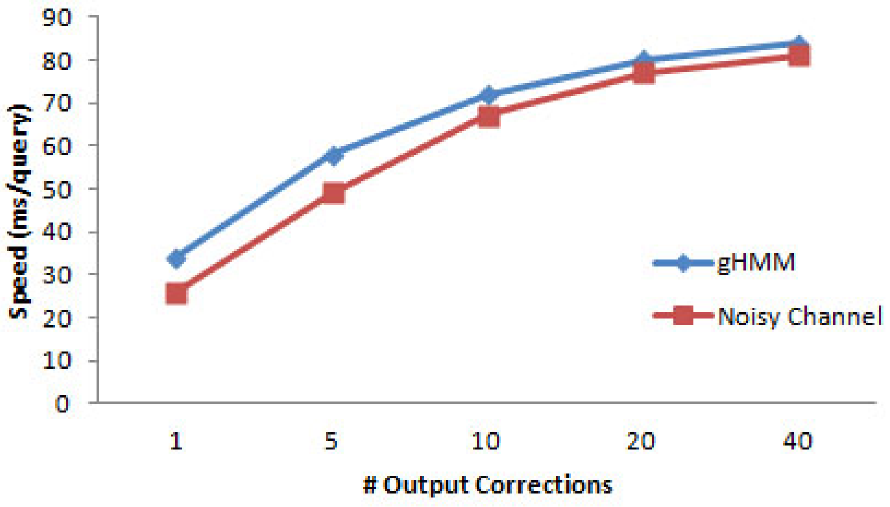


Figure 3.2: Runtime Comparison

3.5 Conclusions and Future Works

In this chapter, we presented a novel generalized hidden Markov model (gHMM) for query spelling correction that can address two major deficiencies of previous approaches to query spelling correction, i.e., inability of handling all the major types of spelling errors, and inability of searching efficiently in the complete candidate space. We have also proposed a novel discriminative training method for the gHMM model which enables us to go beyond regular HMM to incorporate useful local features for more effective query spelling correction. Experiment results on two query correction datasets show that gHMM can effectively handle all the major types of spelling errors and outperforms a state-of-the-art baseline by a large margin.

Moreover, our generalized HMM, equipped with the discriminative training, scores the query corrections directly and output a final ranked list of spelling corrections, without needing a filtering stage to prune the candidate space as typically required by an existing method. We have demonstrated that as an efficient one-stage approach, the proposed gHMM can also be used as a filter to construct a more complete working set than the existing noisy channel filter, making it possible to combine it with any complicated spelling correction methods to further improve accuracy. In other words, the proposed gHMM model can serve as a better candidate generation method in a two-stage framework where any sophisticated and potentially more effective spelling correction method can be applied to re-rank the generated candidates for more accurate corrections. In this work, we only focused on local feature functions in order to ensure efficient evaluation of all the candidates in the search space. However, some global features, such as the overall editing distance, frequency of the query in a query log can be potentially utilized to further improve the correction accuracy. How to add the global features to the gHMM model while still ensuring efficient search and evaluation of all the candidates in the search space, is an interesting direction for future work.

Chapter 4

Query Segmentation Using Clickthrough

4.1 Introduction

One difficulty toward deeper level query intent understanding is that web search query is usually lack of explicit linguistic signals such as segmentation, capitalization, punctuation and standard grammar. Successful detection of such latent query linguistic signals can help improve the retrieval performance. In this chapter we focus on the identification of the most important type of query linguistic signals – query segmentation.

Query segmentation is to separate the query words into disjoint and semantic meaning segments. It is an important tasks in modern information retrieval. For example, accurate query segmentation is the prerequisite for semantic retrieval models, phrase-based query reformulation and automatic relevance feedback. Supervised techniques have been used to solve the query segmentation problem in the past [15, 101]. However, they require lots of segmentation labels which are expensive to collect. An unsupervised approach based on a large text corpus and Wikipedia has been reported to achieve competitive performance [84]; but its accuracy without Wikipedia is still low, partly due to the lack of relevant information about the query structure.

In a modern search engine, there is a large amount of relevant data in the form of clickthroughs. Such data reflects users' implicit preference of documents, and can be leveraged to infer the underlying segmentation of the queries. In this chapter, we propose a unsupervised probabilistic model to exploit user clickthroughs for query segmentation. Model parameters are estimated by an efficient EM algorithm. Segmentation results on a standard dataset demonstrate that our model significantly outperforms the EM model in [84] without the use of Wikipedia. Additionally, by combining more data from external resources, such as the Microsoft Web N-gram [94], our model can outperform state-of-the-art baselines.

One of the most important applications for query segmentation is to improve the retrieval models by incorporating query segmentation. Most information retrieval techniques, such as vector space models and language modeling approaches, rely on the bag-of-words assumption that every query term is independent in the relevance computation. But this assumption is over simplified; users have an order in mind when formulating queries to search for information. One of the reasons why bag-of-words based methods remain popular is because data sparsity makes it harder to estimate models imposing term dependencies [18, 33, 81]. Successful query segmentation has a great potential to lead to better retrieval models that can utilize higher-order term dependencies.

However, query segmentation is ambiguous in nature – the same query can be segmented in different ways by different people. Although several methods for query segmentation have been proposed, surprisingly little research has been performed to address the segmentation ambiguity and incorporate this information into retrieval models. In this chapter, we propose a query segmentation model that quantifies the uncertainty in segmentation by probabilistically modeling the query and clicked document pairs. We further incorporate the probabilistic query segmentation into a unified language model for information retrieval. Experiments on a large web search dataset from a major commercial search engine show that the integrated language model with query segmentation (QSLM) outperforms both the BM25 model and other language models.

4.2 Problem Setup

The task of query segmentation is to separate the query words into disjoint segments so that each segment maps to a semantic unit. Given a query $Q = w_1, w_2, ..., w_n$ of length

n, a segmentation $S = S_1 S_2 ... S_M$ of length M is consistent with the query Q if $S_m = w_{b_m} w_{b_m+1} ... w_{b_{m+1}-1}$ for $1 = b_1 < b_2 < ... < b_{M+1} = n + 1$. We define $B = b_1, b_2, ..., b_{M+1}$ as the segmentation partition, independent of the actual query words, and $\mathbb{B}_n$ as the set of all possible segmentation partitions consistent with any query of length n. There are a total of 2^{n-1} segmentation partitions in $\mathbb{B}_n$. Note that given a query Q, the segmentation partition B and the query segments S can be uniquely derived from each other.

Because query segmentation is potentially ambiguous, we are interested in assessing the probability of a query segmentation under some probability distribution: $P(S|\theta)$. With such a probabilistic model, we can then select those segmentations with high probabilities and use them to construct models for information retrieval. For example, for the query "*bank of america online banking*", {[*bank of america*] [*online banking*], 0.502}, {[*bank of america online banking*], 0.428} and {[*bank of*] [*america online*] [*banking*], 0.001} are all valid segmentations, where brackets [] are used to indicate segment boundaries and the number at the end is the probability of that particular segmentation. In this example, the first two segmentations are likely segmentations with high probabilities, whereas the last one is a rare segmentation, as reflected by the low probability. In the next section, we discuss how to compute the probability $P(S|Q)$ of a segmentation S given a query Q.

4.3 Query Segmentation

The search log in a modern search engine usually contains a lot of user clickthrough data, where user-issued queries and corresponding clicked documents are recorded. This kind of data contains rich information about users' preferences for each query. By carefully modeling the clickthroughs, we can assess the likelihood of a segmentation structure according to the collective user behavior. Table 4.1 shows examples of the clicked documents for two real-world queries from the search log. In these examples, although there are variations in the query words and documents, the sub-sequence "bank of america" remains intact in all clicked documents. The evidence strongly suggests that "*bank of america*" should be a segment. This observation motivates us to model the query segmentation using the query and clicked document pairs, a previously unexplored idea.

Table 4.1: Examples of Query and Clicked Documents

Query	Clicked document title
bank of america invesment	1. bank of america associate banking investments homepage 2. bank of america investment services inc investments overview 3. bank of america associate banking investments banking services ...
credit card bank of america	1. bank of america credit cards contact us overview 2. secured visa credit card from bank of america 3. credit cards overview find the right bank of america credit card for you ...

We now propose an unsupervised query segmentation model using user clickthroughs. We first describe the model for generating queries and will later extend it to query-click document pairs. The process of generating a query can be described as follows:

1. Pick a query length n under a length distribution.
2. Select a segmentation partition $B \in \mathbb{B}_n$, according to a segmentation partition model $P(B|n, \psi)$.
3. Generate query segments S_m consistent with B, according to a segment unigram model $P(S_m|\theta)$.

Recall that given a query Q of length n, the query segments S and the segmentation partition B can be derived from each other. Thus, we can compute the probability of a segmentation as:

$$\begin{aligned} P(S|Q,\theta,\psi) &= P(B|Q,n,\theta,\psi) \\ &= \frac{P(Q|B,\theta)\cdot P(B|n,\psi)}{\sum_{B'\in\mathbb{B}_n} P(Q|B',\theta)\cdot P(B'|n,\psi)}, \end{aligned} \tag{4.1}$$

$$P(Q|B,\theta) = \prod_{m=1}^{M} P(S_m|\theta) \tag{4.2}$$

where $P(Q|B,\theta)$ is the probability of generating a query Q given segmentation partition B. The $P(B|n)$ can be estimated by an expectation maximization algorithm described in the following section. However, in this work we set $P(B|n)$ to a particular form by imposing an infinite strong prior that penalizes longer segments:

$$P(B|n,\psi) = \frac{\prod_{m=1}^{M} P(|S_m(B)|\ |\psi)}{\sum_{B'\in\mathbb{B}_n} \prod_{m'=1}^{M(B')} P(|S_{m'}(B')|\ |\psi)} \tag{4.3}$$

$$P(|S_m(B)|\ |\psi) = e^{-|S_m(B)|^f} \tag{4.4}$$

where $|S_m(B)|$ is the length of the m^{th} segment specified by B, and f is a factor controlling the segment length penalty. Note that the denominator is constant for a fixed length n. Since the probability of a segmentation is the product of all segment probabilities $P(S_m|\theta)$ and $P(B|n,\phi)$, such a segment length penalty is crucial to counter the bias for longer segments as they result in fewer segments and hence fewer terms in the final product. This need for segment length penalty is also discussed by Peng *et. al* in [71].

To extend the model to observed pairs of the query Q and the clicked document D, we consider Q to be generated from an interpolated model, consisting of the global component $P(S_m|\theta)$ and a document-specific component $P(S_m|\theta_D)$. Specifically, we redefine the query probability given a segmentation partition in Equation (4.2) as:

$$P(Q|B, \theta, \theta_D) \propto \prod_{m=1}^{M} P(S_m|\theta)P(S_m|\theta_D) \tag{4.5}$$

Mathematically, this is equivalent to generating each query segment using a log-linear interpolation of the global and document-specific models. Figure 4.1 illustrates the segmentation partition and the process of generating a query given the model.

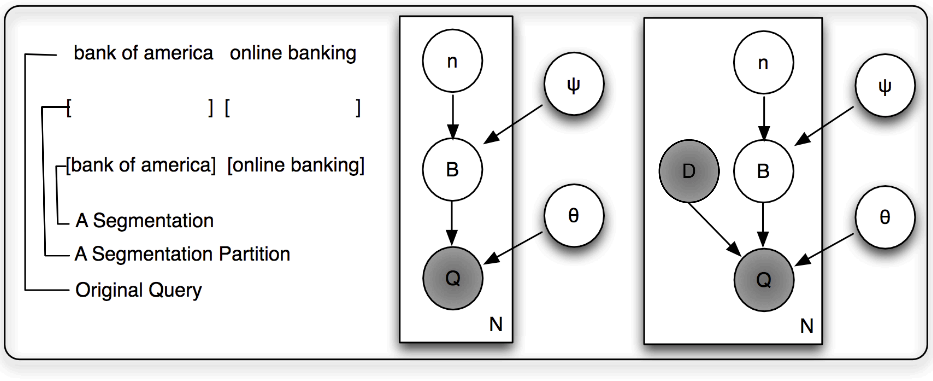


Figure 4.1: The Generative Model of Segmentation. Left: the query segmentation partition; middle: the process of generating a query Q; right: the process of generating query Q with clicked document D

For $P(S_m|\theta_D)$, we employ a smoothed bigram language model trained from the document D and interpolated with the global document collection statistics θ_C to model the probability of $S_m = w_{b_m} w_{b_m+1} ... w_{b_{m+1}-1}$:

$$\begin{aligned} P(S_m|\theta_D) &= \prod_{l=b_m}^{b_{m+1}-1} P(w_l|w_{l-1}, \theta_D) \\ &= \prod_{l=b_m}^{b_{m+1}-1} [(1-\lambda)P_{bi}(w_l|w_{l-1}, \theta_D) \\ &\qquad + \lambda P_{bi}(w_l|w_{l-1}, \theta_C)] \end{aligned} \tag{4.6}$$

where

$$P_{bi}(w_l|w_{l-1},\theta_D) = \frac{f^D_{w_{l-1}w_l} + \mu_D \frac{f^D_{w_l}}{|D|}}{f^D_{w_{l-1}} + \mu_D},$$
$$P_{bi}(w_l|w_{l-1},\theta_C) = \frac{f^C_{w_{l-1}w_l} + \mu_C \frac{f^C_{w_l}}{|C|}}{f^C_{w_{l-1}} + \mu_C},$$

λ is the mixture weight, μ_C and μ_D are the bigram backoff weights, and f_{w_l}, $f_{w_{l-1}w_l}$ are the n-gram counts in document D or corpus C.

Overall, we want to estimate $\hat{\theta}$ to maximize the log likelihood of observing all the query-clicked document pairs in the dataset:

$$\log P(\mathbb{Q}|\theta,\theta_D) = \sum_l log \sum_{B\in\mathbb{B}_{n_l}} P(B|n_l) \cdot \prod_{m=1}^{M_l} [P(S_m(Q_l)|\theta) \cdot P(S_m(Q_l)|\theta_{D_l})] \tag{4.7}$$

With $\hat{\theta}$, we can compute the most probable segmentations for any query according to Equation (4.1).

4.3.1 Model Parameter Estimation by EM

Since the joint probability in Equation (4.7) involves the logarithm of a summation over hidden variables B, there is no exact analytical solution for $\hat{\theta}$. However, we can apply the expectation maximization (EM) algorithm to maximize the joint probability of all observed data. In the E-step, we evaluate the posterior probability of a valid segmentation of Q given the previous model parameter estimate $\theta^{(k-1)}$:

$$P(B|Q,\theta^{(k-1)},\theta_D,\psi) = \frac{P(Q|B,\theta^{(k-1)},\theta_D,\psi)\cdot P(B|n,\psi)}{\sum_{B'\in\mathbb{B}_n} P(Q|B',\theta^{(k-1)},\theta_D,\psi)\cdot P(B'|n,\psi)} \tag{4.8}$$

where

$$P(Q|B,\theta^{(k-1)},\theta_D,\psi) = \prod_{m=1}^{M}[P(S_m|\theta^{(k-1)}) \cdot P(S_m|\theta_D)]$$

In the M-step, we update the estimate of θ according to:

$$\begin{aligned} P(w_1...w_r|\theta^{(k)}) = & \frac{1}{Z} \cdot \sum_l \sum_{B \in \mathbb{B}_{n_l}} [P(B|Q_l,\theta^{(k-1)},\theta_D,\psi) \cdot \\ & \sum_{m=1}^{M_l} \delta(S_m(B,Q_l) = w_1...w_r)] \end{aligned} \quad (4.9)$$

where Z is the normalization factor and $\delta()$ is an indicator function checking if segment S_m is equal to n-gram $w_1...w_r$. For a query of n keywords, a naive computation for the M-step requires summing of all 2^{n-1} possible segmentations, which is computationally impractical for longer queries. Fortunately, it can be computed efficiently using the Baum-Welch algorithm [13].

Here we introduce a graph representation for query segmentation. Given a query Q of length n, all segmentations consistent with Q can be represented by a graph G with $n+1$ nodes. Figure 4.2 illustrates a graph representation for two valid segmentations of the query "*bank of america online banking*". For a graph with $n+1$ nodes, there are a total of 2^{n-1} ways to connect node 1 to node $n+1$, each corresponding to a valid segmentation. For example, in the upper panel of Figure 4.2, there is a connection from node 1 to 4, corresponding to a segmentation boundary between *america* and *online.* In this case, the arc from node 1 to 4 corresponds to the segment "bank of america".

Using this graph representation, Equation (4.9) in the M-step can be rewritten as:

$$P(w_1...w_r|\theta^{(k)}) = \frac{\sum_l \sum_i \sum_j \xi_l(i,j) \cdot \delta_l(S_{i \to j} = w_1...w_r)}{\sum_l \sum_i \sum_j \xi_l(i,j)} \quad (4.10)$$

where $\xi(i,j) = P(S_{i\to j}|Q,\theta^{(k-1)})$ is the probability of the segment S_m represented by the

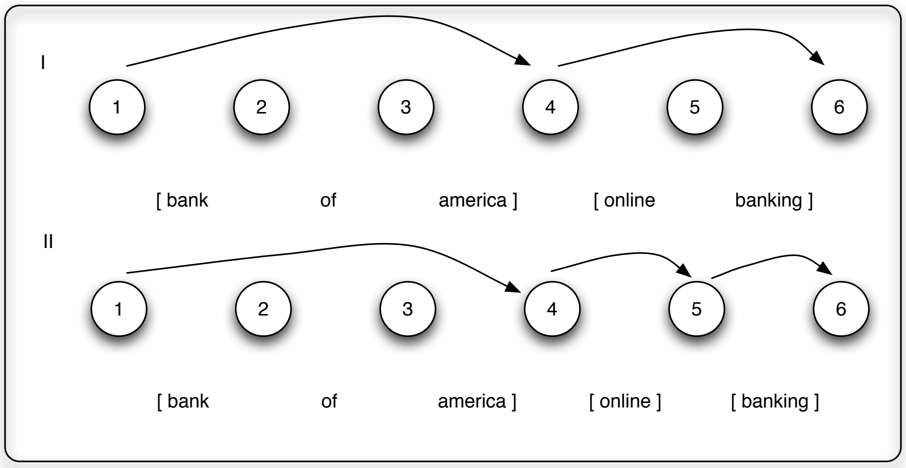


Figure 4.2: Graph Representation of Segmentations

arc from node i to node j for the query Q, $\delta(S_{i\to j} = w_1...w_r)$ is an indicator function with a value of 1 when $S_{i\to j} = w_1...w_r$ and 0 otherwise.

In order to compute $\xi_l(i,j)$, we introduce
$\alpha(i) = P(Q, i|\theta^{(t-1)})$, the probability of observing the query Q from the beginning of the graph to node i, and $\beta(j) = P(Q|j, \theta^{(t-1)})$, the probability of observing Q from node j to the end of the graph:

$$\alpha(i) = \sum_{k:k<i} \alpha(k) \cdot P(S_{k\to i}|\theta^{(t-1)}) \cdot e^{-|S_{k\to i}|^f} \cdot P_D(S_{k\to i}),$$

$$\beta(j) = \sum_{k:k>j} \beta(k) \cdot P(S_{j\to k}|\theta^{(t-1)}) \cdot e^{-|S_{j\to k}|^f} \cdot P_D(S_{j\to k}),$$

$$\xi_l(i,j) = \alpha_l(i) \cdot \beta_l(j) \cdot P(S_{i\to j}|\theta^{(t-1)}) \cdot e^{-|S_{i\to j}|^f} \cdot P_{D_l}(S_{i\to j})$$

with the initial condition $\alpha_l(1) = 1, \beta_l(n) = 1$. Algorithm 3 summarizes the steps for estimating θ. For a set of queries with equal length, the computation complexity for each iteration is $O(Ln^2)$, where L is the number of input query-document pairs and n is the number of words in each query. Once the optimal θ is obtained, the probability of a segmentation $P(S|Q, \theta, \psi)$ can be computed by Equation (4.1).

Algorithm 3: N-gram concept probability estimation

```
input  : A set of query-clicked document pairs
         $\mathbb{O} = \{< Q_l, D_l >\}, l \in [1, N]$
output: Optimal estimate of $\theta = \{P(S_m)\}$
1  Init $P(S_m|\theta^{(0)}) \leftarrow \frac{Count(S_m)}{Count(total\ Ngrams\ in\ query\ collection)}$;
2  for $t \leftarrow 1$ to $T$ do
3      $P(S_m|\theta^{(t)}) \leftarrow 0$;
4      $\xi_{total} \leftarrow 0$;
5      for $l \leftarrow 1$ to $N$ do
6          $G_l$ is a graph representing query $Q_l$, with $n+1$ nodes; $\alpha_l(1) = 1$; $\beta_l(n+1) = 1$;
7          for node $i \leftarrow 2$ to $n+1$ do
8              $\alpha_l(i) \leftarrow \sum_{k:k<i} \alpha_l(k) \cdot P(S_m|\theta^{(t-1)})\cdot$
9                          $e^{-|(S_{k\to i})|^f} \cdot P_{D_l}(S_{k\to i})$;
10         for node $j \leftarrow n$ to $1$ do
11             $\beta_l(j) \leftarrow \sum_{k:k>j} \beta_l(k) \cdot P(S_{j\to k}|\theta^{(t-1)})\cdot$
12                         $e^{-|S_{j\to k}|^f} \cdot P_{D_l}(S_{j\to k})$;
13         for node $i \leftarrow 1$ to $n+1$ do
14             for node $j \leftarrow i+1$ to $n+1$ do
15                 $\xi_l(i,j) \leftarrow \alpha_l(i) \cdot \beta_l(j) \cdot P(S_{i\to j}|\theta^{(t-1)})\cdot$
16                             $e^{-|S_{i\to j}|^f} \cdot P_{D_l}(S_{i\to j})$;
17                 $P(S_m = S_{i\to j}|\theta^{(t)}) \leftarrow$
18                             $P(S_m = S_{i\to j}|\theta^{(t)}) + \xi_l(i,j)$;
19                 $\xi_{total} \leftarrow \xi_{total} + \xi_l(i,j)$;
20     $P(S_m|\theta^{(t)}) \leftarrow \frac{P(S_m|\theta^{(t)})}{\xi_{total}}$;
21 return $\theta = \{P(S_m)\}$;
```

4.3.2 Utilizing Other Resources

N-gram statistics from a very large scale of text resources can also be utilized to improve query segmentation. In fact in [84], the biggest improvement in segmentation accuracy is achieved by utilizing information from Wikipedia. In addition, [36] also reports a well-performing naive query segmentation method using Google Web N-gram. Here we propose a simple approach utilizing the Microsoft Web N-gram service. MS Web N-gram is essentially a distribution of n-gram probability θ' over the web. The probability of a segmentation given Q is defined as:

$$\begin{aligned} P(B|Q,\theta',\psi') &\propto P(Q|B,\theta',\psi') \cdot P(B|\psi') \\ &= \prod_{m=1}^{M} P(S_m|\theta') \cdot P(B|\psi') \\ &\propto \prod_{m=1}^{M} P(S_m|\theta') \cdot e^{-|S_m(B)|^{f'}} \end{aligned} \tag{4.11}$$

Furthermore, we can combine our query segmentation model with clickthrough and the simple model with Web N-gram into an interpolated model:

$$\begin{aligned} logP(B|Q,\theta,\theta',\psi,\psi') =& (1-\omega) \cdot logP(B|Q,\theta,\psi) \\ & + \omega \cdot logP(B|Q,\theta',\psi') \end{aligned} \tag{4.12}$$

we find the setting of $\omega = 0.5, f = 2.0, f' = 2.0$ results in a model with good segmentation accuracy.

4.4 Segmentation Experiments

In this section we report the query segmentation results obtained by our model and other baselines on two datasets. One is from a standard dataset established by previous research, and the other is constructed by ourselves. We also conduct extensive analysis on several aspects of the results.

4.4.1 Data Preparation and Evaluation Metrics

We use two sets of queries for evaluating the query segmentation models. The first set (Set 1) is a standard query segmentation dataset established by Bergsma and Wang [15], which is also applied in [84]. In this dataset, annotator A, B, and C independently segmented 500 queries which are sampled from the AOL 2006 query log. Among these 500 human queries, the 220 where the 3 judges agree are called the "Intersection" set.

The above segmentation dataset is focused on noun queries. But in this work we are also interested in web queries. Therefore we prepare another set of 1,000 queries sampled from the search log of a major commercial search engine, which we name the 1000-query dataset. We invite three domain experts to segment the queries independently, employing the same evaluation metrics as Set 1. Besides expert annotations, this dataset also has clickthrough information and relevance judgments for the top documents, which is used by subsequent experiments when comparing retrieval models.

To measure the segmentation effectiveness, we report results on three evaluation metrics. (1) Query accuracy: the percentage of queries for which the predicted segmentation matches the gold standard completely; (2) Classification accuracy: the ratio of correctly predicted boundaries in between every two consecutive words; (3) Segment accuracy: how well the predicted segments match the gold standard under the information retrieval measures of precision, recall, and F-score.

As baseline, we include the three models in [84]: Mutual information, EM + corpus (query log), and EM + corpus + Wikipedia. We also include a method using Google Web N-gram [36] and a simple model with MS Web N-gram, as defined in Section 4.3.2. Our model + clickthrough and our model + clickthrough + MS Web N-gram are included in the comparison. The parameters of our segmentation model is trained on a large set of search log containing about 20 millions query-clicked document pairs.

4.4.2 Query Segmentation Results

Table 4.2 shows the results of our model as well as the baseline models on the standard dataset. Columns 3 to 5 represent models without using external data source (basic models), while columns 6 to 9 are models utilizing large external sources, such as Wikipedia and web-scale n-gram (extended models). Among the basic models, our model performs the best according to annotator A, C and the intersection of these annotators. These results are significantly better than the corresponding results by the EM + corpus model in [84]. For the result based on annotator B, our model is comparable to that of [84] (0.571 vs 0.573 on segment F score). For the extended models, simple model + MS Web N-gram performs well, similar to the results for simple model with Google Web N-gram as reported in [36]. It indicates the positive impact of n-gram statistics on query segmentation. However, our model, as well as EM model + Wikipedia in [84] outperforms the simple models consistently in all annotators' judgments; and our extended model performs better than that of [84]. For example, in the intersection judgments, the F score of our model is 0.779, while model in [84] is 0.774. Compared to the simple model + MS Web N-gram, whose intersection F score is 0.728, our model achieves a 7.0% gain on the same measure. It suggests the effectiveness of our model and the benefit from combining additional large scale N-gram statistics.

Table 4.2: Segmentation Performance on the Standard Dataset

Annotator	Measure	MI	[84] EM + Corpus	Our Model	Simple Model + MS Web N-gram	[36] Simple Model + Google Web N-gram	[84] EM + + Corpus +Wiki	Our Model + MS Web N-gram
A	query accuracy	0.274	0.414	0.440	0.482	0.536	0.526	0.540
	classify accuracy	0.693	0.762	0.776	0.782	0.807	0.810	0.803
	segment precision	0.469	0.562	0.598	0.645	0.665	0.657	0.669
	segment recall	0.534	0.555	0.639	0.602	0.708	0.657	0.713
	segment F	0.499	0.558	**0.618**	0.622	0.686	0.657	**0.690**
B	query accuracy	0.244	0.440	0.410	0.466	0.380	0.494	0.485
	classify accuracy	0.634	0.774	0.750	0.777	0.752	0.802	0.776
	segment precision	0.408	0.568	0.521	0.568	0.519	0.623	0.591
	segment recall	0.472	0.578	0.631	0.601	0.626	0.640	0.650
	segment F	0.438	**0.573**	0.571	0.584	0.568	**0.631**	0.619
C	query accuracy	0.264	0.416	0.402	0.460	0.454	0.494	0.465
	classify accuracy	0.666	0.759	0.756	0.772	0.772	0.796	0.803
	segment precision	0.451	0.558	0.548	0.597	0.581	0.634	0.624
	segment recall	0.519	0.561	0.619	0.590	0.653	0.642	0.655
	segment F	0.483	0.559	**0.582**	0.594	0.615	0.638	**0.639**
Intersect	query accuracy	0.343	0.528	0.586	0.636	0.627	0.671	0.682
	classify accuracy	0.728	0.815	0.842	0.847	0.851	0.871	0.855
	segment precision	0.510	0.640	0.681	0.736	0.718	0.767	0.770
	segment recall	0.550	0.650	0.747	0.721	0.778	0.782	0.788
	segment F	0.530	0.645	**0.713**	0.728	0.746	0.774	**0.779**

4.4.3 Results on the 1000-query Dataset

We compare our query segmentation model with the simple model + MS Web N-gram on the 1000-query dataset. Table 4.3 shows the segmentation results on the this set. Although the simple segmentation model with web n-gram works very well in the standard dataset, it performs inferior to our model in the 1000-query dataset. In 2 out of 3 annotator judgments, our model outperforms the simple model. And in the intersection judgments our model also works better than the simple model by 4.6%. Since this dataset is sampled from a set of web search queries, results in this experiment indicate that our model fits web search queries, whose characteristics are different from noun queries, better.

4.4.4 Effect of the Penalty Factor

The factor f in Equation (4.4), which controls how much penalty is given to a segment of length $|S_m|$, is important to the our proposed model. We now investigate how the segmentation result changes according to different values of f. For this purpose, we re-run our model (without web n-gram) on the standard dataset with f ranging from 1.5 to 3.0 in steps of 0.25. Figure 4.3 summarizes the results. There are common trends across annotator A, B, C and their intersection. The F score increases when f increases from 1.5 to 2.0, and decreases afterwards. It suggests that too little penalty (small f) favors long segments and hurts

Table 4.3: Results on the 1000-query Dataset

Annotator	Measure	Our Model	Simple Model + MS Web N-gram
A	query accuracy	0.386	0.316
	classify accuracy	0.631	0.538
	segment precision	0.434	0.368
	segment recall	0.540	0.552
	segment F	**0.481**	0.441
B	query accuracy	0.447	0.403
	classify accuracy	0.690	0.619
	segment precision	0.533	0.476
	segment recall	0.602	0.648
	segment F	**0.565**	0.549
C	query accuracy	0.472	0.545
	classify accuracy	0.703	0.749
	segment precision	0.670	0.693
	segment recall	0.582	0.730
	segment F	0.623	**0.713**
Intersection	query accuracy	0.624	0.567
	classify accuracy	0.761	0.642
	segment precision	0.372	0.301
	segment recall	0.405	0.395
	segment F	**0.388**	0.342

segmentation accuracy, while too much penalty (big f) negatively impacts on the results since it favors segments with very short length. It also indicates that a moderate penalty at $f = 2.0$ is a reasonable choice for the proposed model.

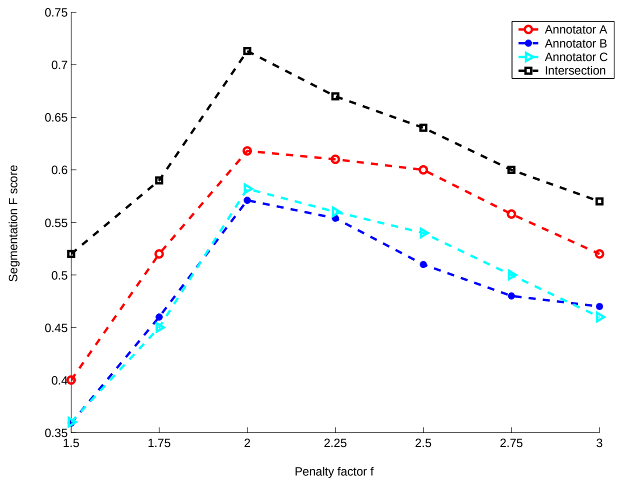


Figure 4.3: Query Segmentation Performance with Respect to Penalty Factor

4.5 Integrated Language Model

In this section we will introduce the proposed integrated language model with query segmentation (QSLM). We first motivate QSLM with an oracle experiment, then describe the derivation of QSLM, and finally conduct extensive experiments on a large scale web search dataset.

4.5.1 Oracle Ranker

To motivate the formulation of QSLM, we have carried out an intuitive and interesting experiment. Given an oracle ranker, we let the ranker choose the bigram or unigram language model for each query, whichever gives a better NDCG score. Table 4.4 lists the result of the oracle ranker compared to other models. As such a simple oracle performs significantly better than either the bigram or unigram language models, it suggests that it may be possible to improve the search ranking if one can successfully emulate the behavior of the Oracle – to accurately predict when to use a unigram model and when to use a bigram model. We will show that query segmentation can help achieve a similar effect.

Table 4.4: Oracle Ranker

Method	NDCG@1	NDCG@3	NDCG@10
BM25	0.3108	0.3358	0.3986
Unigram LM	0.3117	0.3366	0.3999
Bigram LM	0.3141	0.3380	0.3999
Oracle Ranker	0.3471	0.3628	0.4186

4.5.2 Integrated Language Model

Given a model for computing the probability of a segmentation S for a query Q, we can exploit this information and develop a new retrieval model incorporating the query segmentation structure. Note that the retrieval model proposed here is independent of the query segmentation technique. We start by formulating the integrated language model with query segmentation based on the probabilistic ranking principle [76]. Specifically, we can rewrite the probability that a document is relevant to a query as follows:

$$
\begin{aligned}
&P(R=1|Q,D)\\
&=\sum_B P(B|Q,D)P(R=1|B,Q,D)\\
&=\sum_B P(B|Q,D)\frac{P(Q|B,D,R=1)P(R=1|B,D)}{\sum_{r=\{0,1\}}P(Q|B,D,R=r)P(R=r|B,D)}\\
&=\sum_B P(B|Q,D)\frac{\frac{P(Q|B,D,R=1)}{P(Q|B,D,R=0)}}{\frac{P(Q|B,D,R=1)}{P(Q|B,D,R=0)}+\frac{P(R=0|B,D)}{P(R=1|B,D)}}\\
&\equiv\sum_B P(B|Q,D)\frac{a}{a+b}
\end{aligned}
$$

where:

$$
a=\frac{P(Q|B,D,R=1)}{P(Q|B,D,R=0)},\qquad b=\frac{P(R=0|B,D)}{P(R=1|B,D)}.
$$

As the query segmentation is performed independently of the document, $P(B|Q,D)=P(B|Q)$. Furthermore, when a document is irrelevant, we can approximate the query as being generated from the background corpus statistics, independent of the document:

$$
a \approx \frac{P(Q|B,D,R=1)}{P(Q|B,\theta_C)}
$$

Finally, as the relevance of a document is independent of the segmentation partition without knowing the query, we will assume that all document has an equal probability of being relevant. Thus, we can approximate b as the average ratio of irrelevant to relevant documents over a set of queries.

$$
b\approx\frac{P(R=0)}{P(R=1)}\approx\sum_Q\frac{|\text{Irrelevant}(Q)|}{|\text{Relevant}(Q)|}
$$

In this work, we apply a language model approach to estimate the the probability ratio a:

$$
\begin{aligned}
a &\approx \frac{P(Q|B,D,R=1)}{P(Q|B,\theta_C)} \\
&= \prod_{m=1}^{M} \frac{P(S_m|D,R=1)}{P(S_m|\theta_C)} \\
&= \prod_{m=1}^{M} \frac{P(w_{k_m},...,w_{k_{m+1}-1}|\theta_D)}{P(w_{k_m},...,w_{k_{m+1}-1}|\theta_C)} \\
&= \prod_{m=1}^{M} \prod_{i=k_m}^{k_{m+1}-1} \frac{P(w_i|w_{i-1},\theta_D)}{P(w_i|w_{i-1},\theta_C)}
\end{aligned}
$$

For irrelevant documents, the query segments are generated from the an n-gram language model trained from the background corpus. For relevant documents, the query segments are modeled using a smoothed bigram model trained from the document, interpolated with the background corpus. Specifically:

$$P(w_i|w_{i-1},\theta_C) = \frac{f_{w_{i-1}w_i,C} + \mu_C \frac{f_{w_i,C}}{|C|}}{f_{w_{i-1},C} + \mu_C}, \tag{4.13}$$

$$P_{bi}(w_i|w_{i-1},\theta_D) = \frac{f_{w_{i-1}w_i,D} + \mu_D \frac{f_{w_i,D}}{|D|}}{f_{w_{i-1},D} + \mu_D}, \tag{4.14}$$

$$P(w_i|w_{i-1},\theta_D) = (1-\lambda)P_{bi}(w_i|w_{i-1},\theta_D) + \lambda P(w_i|w_{i-1},\theta_C) \tag{4.15}$$

4.6 Retrieval Experiments

In this section we conduct a set of experiments for the QSLM model on the web search task. The main reason why we did not carry out experiments on the TREC datasets is due to the lack of clickthrough data for TREC queries, which is important to our study. In the following sections, we invest several variants of the model and discuss the choice in model parameters.

4.6.1 Evaluation Metrics and Baselines

We evaluate the retrieval models on a large-scale real world dataset, containing 12,064 English queries sampled from the query log of a major commercial search engine. On average, each query is associated with 74 web documents, with each query-document pair manually assigned a relevance label on a 5-level scale: 0 means that the document D is detrimental to query Q, 4 means that the document D is most relevant to Q. For comparison, we include 3 baseline models in the results: BM25 [77], unigram LM with Dirichlet smoothing [103], and bigram LM as specified in Equation (4.15). In order to obtain the optimal parameters in our model as well as in the baselines, we divide the whole dataset evenly into a training set and a test set, each containing 6,032 queries, and estimate the parameters from the training set using grid search, as proposed in [87]. The optimal parameters of the models are reported in Table 4.5. Finally we also list the simple oracle results as reference. The performance of all the retrieval models is measured by mean normalized discounted cumulative gain (NDCG) [40] at truncation levels 1, 3, and 10. We list the dataset statistics in Figure 4.4 and report detailed results of the retrieval models in Table 4.6.

Table 4.5: Optimal values of parameters

Model	NDCG
Unigram LM	$\mu = 2.702$
Bigram LM	$\mu_C = 425026, \mu_D = 0.51, \lambda = 0.681$
QSLM	$\mu_C = 500213, \mu_D = 0.50, \lambda = 0.90, b = 720$

4.6.2 Retrieval Results

In Table 4.6, we report the results by query length. For short queries, there are few variations in the segmentation. Thus, there is little room for improvement by exploiting segmentation

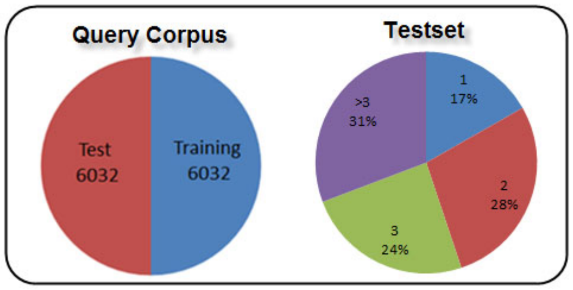


Figure 4.4: Query Distribution in the Datasets

information. However, the effect of query segmentation is more pronounced when the query contains 4 or more words, which we consider as a long query. In this case, the NDCGs of BM25 and unigram LM are similar, both outperformed by the bigram LM. However, QSLM's performance (0.3419, 0.3539 and 0.4040) is significantly better than all other models at all levels of NDCG truncation. In fact, we have conducted a paired t-test between QSLM and the other models. At confidence level $\alpha = 0.01$, the difference between QSLM and BM25/unigram LM at all three levels of NDCG truncation is statistically significant. The difference between QSLM and bigram LM at both NDCG@1 and NDCG@3 are significant.

4.7 Conclusions and Future Works

In this chapter we have proposed a novel unsupervised query segmentation model by jointly modeling the query-clicked documents from the search log. Experimental results on two datasets confirm the effectiveness of our model. Furthermore, we develop a unified language model with query segmentation to improve the search ranking. The implicit relevance information in the clickthrough data is the bridge between our query segmentation model and QSLM. Thorough experiments on a large-scale web search dataset show that search relevance can be improved by leveraging the query segmentations. As there is still a large gap in retrieval performance between the oracle ranker and the QSLM model, we plan to further refine the model to reduce gap in the future. Specifically, we would like to explore the use of QSLM as features to other advanced retrieval models [65].

Table 4.6: Results of IR Models on Web Search

Length	#Queries	Model	NDCG@1	NDCG@3	NDCG@10
1	1012	BM25	0.2515	0.2773	0.3496
1	1012	Unigram LM	0.2515	0.2767	0.3497
1	1012	Bigram LM	0.2462	0.2737	0.3452
1	1012	QSLM	0.2462	0.2737	0.3452
2	1694	BM25	0.3125	0.3391	0.4068
2	1694	Unigram LM	0.3131	0.3393	0.4076
2	1694	Bigram LM	0.3132	0.3392	0.4074
2	1694	QSLM	**0.3169**	**0.3404**	**0.4078**
2	1694	Oracle Ranker	0.3488	0.3656	0.4266
3	1471	BM25	0.3273	0.3603	0.4226
3	1471	Unigram LM	0.3293	0.3607	0.4242
3	1471	Bigram LM	0.3322	0.3617	0.4244
3	1471	QSLM	**0.3332**	**0.3619**	**0.4251**
3	1471	Oracle Ranker	0.3657	0.3877	0.4423
>3	1855	BM25	0.3287	0.3454	0.3988
>3	1855	Unigram LM	0.3294	0.3476	0.4009
>3	1855	Bigram LM	0.3354	0.3500	0.4009
>3	1855	QSLM	**0.3419**	**0.3539**	**0.4040**
>3	1855	Oracle Ranker	0.3651	0.3752	0.4222

Chapter 5

Mining Entity Attribute Synonyms via Compact Clustering

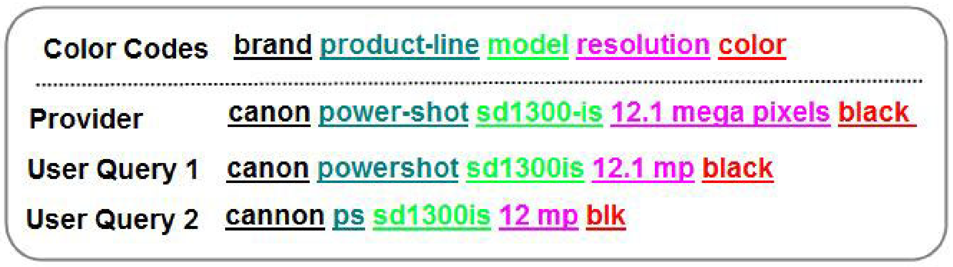
5.1 Introduction

The bag-of-words query representation has been a great success in document retrieval. However, as the search has been expanded to many other types of applications, bag-of-words representation is not sufficient to support the requirements of these applications. One such application is entity search. Nowadays the web contains a wealth of structured data, such as various entity databases, web tables, etc. There is a growing trend in search engines to match unstructured user queries to these structured data sources. In the entity centric search, schema annotation of queries is required to match the schema of the structured data sources. Another application is to present direct answer and facts to queries. Examples include the instant answer box of modern web search engines, and the computational knowledge engine Wolfram Alpha. In order to understand the intention of the user and judge whether a direct answer should be triggered, the query has to be transferred to semantic components and these components are further precessed and matched against the knowledge bases. This level of query intent understanding goes beyond the bag-of-words representation of queries. It aims at deciphering the semantic structure of queries, that is the meaning of every piece of query segment and their relation. It involves tasks such as target type classification which is to infer the category/domain of the query, name entity and attribute recognition and disambiguation, schema matching to a catalog which is to match a query to predefined catalog schema like product catalog tables, semantic role labelling in queries *etc.*

If the query and structured data sources are all written in well-formed texts, the task of semantic annotation of queries is manageable. However a big challenge of query semantic annotation is to handle the variation of texts. In entity search, user expressions of entities often do not match the canonical specifications from the data providers. For example, in the *movie* domain, the full title "the lord of the rings: the return of the king" can be specified by users as "lotr 3", "lotr: return of the king", or "the return of the king". For shoes, people may describe the standard gender value "infant" as "baby" or "toddler". Thus, entity synonym identification, the discovery of alternative ways people describe entities, has become a critical problem to bridge the above mentioned gap between data providers and consumers. In this chapter we focus on the problem of mining entity attribute synonyms.

Traditionally, entity synonym research has focused on finding synonyms of named entities, where the entity itself is completely specified by the referent string. Here we are interested in finding synonyms of entity attribute values (also referred to as entity attribute synonyms throughout this paper). While the attribute values can be entity mentions, they can also be arbitrary strings (adjectives, verbs, etc). In fact, our problem definition is a generalization of finding named entity synonyms, because the named entity expression is often just an attribute of the entity. Fig. 5.1 illustrates an example of such general cases collected from a product title and two user issued queries. Here "canon" is a named entity, but it also matches the attribute *digital-camera.brand*. And "12.1 mega pixel" is an attribute value from the same domain, but cannot be interpreted as a standalone entity. As seen in Fig. 5.1, there are a lot of variations in describing the same attribute values. Successful identification of their surface forms will enable better query intent understanding and better normalization of products from different providers, *etc*.

In the case the attribute value itself is an entity mention, our problem setup is the same as traditional entity synonym finding. Previous research has addressed the synonym identification problem from multiple perspectives. For example, [28, 14] tried to reconcile different references to the same database record. Other works identified alternative forms of a query for web search by measuring query similarity [19, 45], query-document click-graphs



Color Codes	brand product-line model resolution color
Provider	canon power-shot sd1300-is 12.1 mega pixels black
User Query 1	canon powershot sd1300is 12.1 mp black
User Query 2	cannon ps sd1300is 12 mp blk

Figure 5.1: Entity Attribute Value Variations

[7] and query-entity click-graphs [24]. For non-entity attribute values (arbitrary strings), there are also research efforts from the Natural Language Processing community on finding semantic synonyms based on distributional similarity [60, 61], syntactic patterns [37, 16] *et. al.*

However, two major challenges remain. First, finding synonyms without context can not handle semantic ambiguities. There are recent research attempts to identify synonyms with additional context, such as paragraph context in [91]. But for structured database, such information is not always readily available. Second, previous approaches usually focus on utilizing a single signal, such as distributional similarity [60, 61], syntactic patterns [37, 16], or query-entity clicks [24]. Some recent works explored two or more information sources [66, 20]. However, the weights for combining these information sources are usually manually tuned and largely based on experience.

In this chapter we focus on finding synonyms for a set of entity attribute values simultaneously. Our problem setup is a generalization of the entity synonym identification problem, in which the input can be an entity mention or an arbitrary string. To address the deficiencies of existing approaches discussed above, we propose a compact clustering model that enables the integration of multiple heterogeneous information sources. Our main contributions are summarized as follows:

- ***Joint synonym mining from a set of attribute values***. Most previous synonym identification methods search for synonyms one entity at a time. However, processing a set of entity attribute values simultaneously has several advantages. First, as the values are from the same attribute, ie. *movie.title* or *shoe.brand*, they exhibit distinctive contextual patterns within queries and documents. Mining such patterns allows us to define a novel **categorical pattern similarity** function for addressing the semantic ambiguity problem. Second, joint modeling of multiple attribute values also provides prior knowledge about the relationship among candidates. For example, for entity mention "lord of the rings 2", "lord of the rings 3" could be identified as synonym mistakenly. But if we learn synonyms from those two values jointly, this error could be corrected easily because of the awareness of "lord of the rings 3".

- ***Integrating multiple information sources.*** Synonym values generally exhibit similarities in more than one aspect. Some synonym values only differ in a few characters, due to spelling errors or morphological differences. Also, queries that differ only in synonym values tend to have clicks on similar sets of documents. In addition, synonym values generally have similar surrounding contexts within queries and documents. Among these signals, some are more important than others in determining synonym relations. Furthermore, the relative importance of these signals also depends on the domain: a feature that is crucial in the movie domain might be only marginal in the camera domain. Therefore automatic determination of the weights of different information sources is critical. In this work we propose to automatically learn these weights via **compact clustering** – a novel clustering procedure that maximizes the similarity of points within a cluster.

- ***Exploiting additional known information.*** Our compact clustering model can be further enhanced by incorporating additional information. Such information is usually noisy but cheap to get, containing information such as which objects tend to be similar with each other or which objects should be far away from each other. We don't model such additional information as explicit constraints as in [90]. Instead, we extend our model to accommodate such information via a regularization framework.

5.2 Problem Definition

In our problem setup, the input consists of (1) a set of canonical entity attribute values from a domain; (2) candidate synonyms of the canonical attribute values. And the output is the true synonyms of this set of attribute values. As there are multiple interpretations of "alternative expressions", we focus on synonyms that convey the equivalent meaning of the canonical value in the (implied) domain, including semantic alterations, abbreviations, acronyms, permutations, spelling errors, *etc.* Fig. 5.2 illustrates the inputs and outputs. For example, for the input "IBM", the synonyms include "International Business Machines", "Big blue", "IBM corporation" *etc.* In theory, the candidate synonyms can be any arbitrary strings. Yet in practice, it is critical to reduce the search space of synonyms. In Section **??** we describe the process of finding the candidates in detail.

Formally, given a set of K semantically distinct values $\mathbb{V} = \{v_1, v_2, ..., v_K\}$ from an unspecified entity attribute, where each value $v \in \mathbb{V}$ is represented by a canonical string expression, such as "5d mark iii" for *camera.model*. From a set of N candidate synonym values $\mathbb{X} = \{x_1, \ldots, x_N\}$, we can define an oracle mapping $\mathcal{F} : \mathbb{X} \rightarrow \mathbb{V} \cup \{v_0\}$, which assigns each candidate value x to its unique canonical synonym value v, or if x is not a synonym of any value in $\mathbb{V}$, to the special background value v_0. Note that we assume each candidate synonym expression maps to at most one canonical value. Now, we can define the synonym identification problem as follows:

Definition 1: For each canonical attribute value $v \in \mathbb{V}$, find the subset $\mathbb{X}_v = \{x \in \mathbb{X} | \mathcal{F}(x) = v\}$, representing the set of synonym expressions for value v.

Note that for the set of input attribute values $\mathbb{V}$, we have the following assumption:

***Assumption 1*:** values in $\mathbb{V}$ should be semantically distinct and homogeneous.

This assumption is resonable in several application scenarios. For instance, for product providers and online market places like eBay and Amazon, a set of distinct and homogeneous canonical attribute values can be easily obtained from the product catalog. The homogeneous assumption implies that the inputs are from the same domain, which can be leveraged for mining their synonyms collectively.

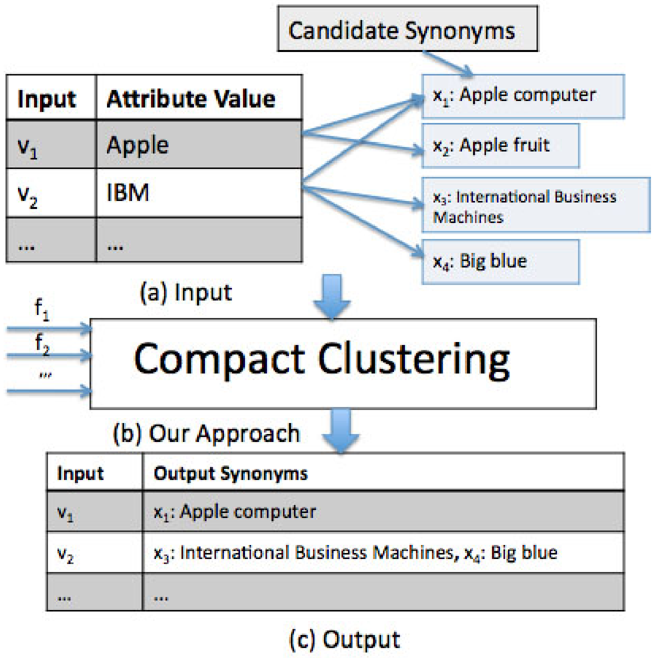


Figure 5.2: Synonym Identification Architecture

5.3 Compact Clustering

As mentioned in the introduction, most previous synonym identification methods search for synonyms one input at a time. However, such strategy have two major drawbacks. First, without modeling the attribute values jointly, it's very difficult to tackle the ambiguation problem since the category context implied by a set of attribute values is lost. Second, this strategy doesn't leverage the prior knowledge multiple attribute values bring to the candidates, especially the should-link and should-not link constraints. In order to take advantage of a set of canonical attribute values, we propose to identify synonyms of attribute values by a clustering model with multiple similarity kernels called ***compact clustering***. In this model, attribute values $\mathbb{X} = \{x_1, x_2, ..., x_N\}$ are modeled as data points. And points are connected with others with similarity function f. Data points form clusters such that points in the same cluster are considered synonyms. The canonical attribute values $v_1, v_2, ..., v_K$ also belong to $\mathbb{X}$, but they have hard assignments to their respective clusters. Fig. 5.2 illustrates the architecture of our proposed clustering framework.

Prior work took advantage of the properties of synonyms such as reflexivity, symmetry, and similarity [60, 61, 20]. We further consider transitivity and compactness in our model. This means that we choose a cluster assignment by considering a committee of points in a cluster rather than a single medoid. Whereas previous methods have only a few similarity features, we want to support arbitrary features. Thus, manual tuning of parameters is not sufficient. Though there are existing works in similarity metric learning [98, 80, 9], we are also interested in unsupervised techniques. Hence, we use the heuristic of compactness to guide our parameter optimization. In this section we first define several similarity kernel functions according to available information. Then we introduce a basic model by motivating the concept of cluster compactness. By addressing the limitations of this model, we propose several extensions that lead to the standard compact clustering model. Finally, we further extend the standard model by leveraging additional noisy information.

5.3.1 Similarity Kernels

Any clustering framework has to define distance/similarity functions between data points. Data points (attribute values) are related to each other in different aspects. For example, in the domain of *movie.title*, two titles are similar if people click on the same set of documents after querying for these titles. Two titles also are similar if they follow similar lexical distribution of the left and right contexts where they appear. In fact, there are heterogeneous types of information that can be leveraged to infer the synonym relationship. Such resources include query log, query-document clickthrough, anchor text, to name a few. Suppose from information type t, the similarity of points x_i and x_j is defined as a similarity kernel $f_t(x_i, x_j) \in [0, 1]$, and each type of similarity kernel is associated with a weight w_t reflecting its relative importance. , then the overall similarity/distance between x_i and x_j can be defined by these similarity kernels. Since our basic model resembles the K-medoids clustering [48], here we follow the nomenclature in the clustering literature and define the distance between x_i and x_j as the combination of the similarity kernels with weights:

$$\begin{aligned} d(x_i, x_j) &= \sum_{t=1}^{T} w_t^{\alpha} \cdot d_t(x_i, x_j) \\ &= \sum_{t=1}^{T} w_t^{\alpha} \cdot (1 - f_t(x_i, x_j)) \end{aligned} \tag{5.1}$$

where $f_t(x_i, x_j) \in [0, 1]$ is the similarity kernel of x_i and x_j calculated based on evidence from information source $t \in \{1, ..., T\}$. Likewise $d_t(x_i, x_j) = 1 - f_t(x_i, x_j) \in [0, 1]$ is the distance between x_i and x_j. α is a constant whose value is set to 2 in this work. And $w_t \geq 0$ are the weights needed to be learned, following constraint:

$$\sum_{t=1}^{T} w_t = 1 \tag{5.2}$$

The special choice of α is to make the optimal w_t easier to solve under the above constraint, as introduced in previous work [21]. In the following, we specifically define four similarity kernels according to four types of information. Note that our framework is not restricted to these kernels. In fact our model can support arbitrary number of similarities from different information sources.

1. **Categorical pattern similarity**. This is a novel similarity kernel which leverages a

set of attribute values simultaneously. A key insight is that as the canonical values are in the same category, they should share common lexical or semantic patterns. Table. 5.1 illustrates the pattern distribution over 50 attribute values from *shoe.brand*. These patterns are found by extracting the left and right lexical context from a set of search queries that contain these brand names. It clearly shows that the brand names are much more likely to appear at the beginning of a query (#EMPTY# pattern on the left); and the word "shoes" is also very frequent following the brand name. By mining the context from the queries containing these attribute values, we are able to discover categorical patterns, which would otherwise be impossible had we looked for synonyms one attribute value at a time due to data sparseness. Specifically, given data points x_i, x_j and the left and right categorical pattern distributions $\bar{\Omega}_l, \bar{\Omega}_r$ derived from the canonical attribute values, we define the categorical pattern similarity between x_i and x_j as:

$$f_1(x_i, x_j) = 1 - |Jaccard(\Omega_i, \bar{\Omega}) - Jaccard(\Omega_j, \bar{\Omega})| \tag{5.3}$$

where $Jaccard(\Omega_i, \bar{\Omega})$ is the average $Jaccard$ similarity of the left context and right context between x_i $(\Omega_{i,l}, \Omega_{i,r})$ and the category $(\bar{\Omega}_l, \bar{\Omega}_r)$:

$$Jaccard(\Omega_i, \bar{\Omega}) = \frac{1}{2} \cdot \left(\frac{||\Omega_{i,l} \cap \bar{\Omega}_l||}{||\Omega_{i,l} \cup \bar{\Omega}_l||} + \frac{||\Omega_{i,r} \cap \bar{\Omega}_r||}{||\Omega_{i,r} \cup \bar{\Omega}_r||} \right) \tag{5.4}$$

Note that the categorical pattern similarity kernel is large only if both x_i and x_j share similar context distributions with the categorical patterns, which is especially effective for excluding the ambiguous candidate strings. For example, for the canonical value "Apple" in the domain of IT companies (implied by inputs "Apple", "IBM", *etc.*), a candidate "Apple fruit" will have very low categorical pattern similarity because this candidate has very different query context.

Table 5.1: Categorical Patterns in Shoe.brand

Left Patterns	Count	Right Patterns	Count
#TOTAL#	4472	#TOTAL#	4472
1. #EMPTY#	3823	1. #EMPTY#	333
2. www	55	2. shoes	109
3. cheap	38	3. com	67
4. discount	35	4. boots	60
5. women	30	5. sandals	42
...		...	

2. **Coclick similarity.** Two attribute values are similar if users click on similar documents when they issue queries containing the two attribute values (proxy queries). Let the set of proxy queries of x_i be $Q_i = \{q_1^i, q_2^i, ..., q_{n_i}^i\}$. For each query q_l^i, the users have clicks on a set of documents, which is denoted as $\Phi^l = \{\phi_1^l, \phi_2^l, ..., \phi_M^l\}$, where M is the total number of documents. And let the accumulation of these clicks be:

$$\Phi = \sum_l \phi^l = \left\{ \sum_l \phi_1^l, \sum_l \phi_2^l, ..., \sum_l \phi_M^l \right\} \tag{5.5}$$

Then for points x_i and x_j, we define their coclick similarity as the cosine similarity of Φ_i and Φ_j:

$$f_2(x_i, x_j) = \frac{\Phi_i \cdot \Phi_j}{||\Phi_i|| \cdot ||\Phi_j||} \tag{5.6}$$

3. **Lexical context similarity.** Under the distributional similarity assumption [64], two strings will carry similar meaning if they share similar context. We observe that for true synonyms, the two attribute values will share common left and right context in web search queries. However this similarity is different from the categorical pattern similarity in that the lexical context similarity is more specific to a particular attribute value while the categorical pattern similarity is related to the patterns of a set of values. Thus the categorical pattern similarity has more power in discriminating the ambiguous attribute values. We define the lexical context similarity of x_i and x_j as

the Jaccard similarity of their left and right context:

$$f_3(x_i, x_j) = \frac{1}{2} \cdot \left(\frac{||\Omega_{i,l} \cap \Omega_{j,l}||}{||\Omega_{i,l} \cup \Omega_{j,l}||} + \frac{||\Omega_{i,r} \cap \Omega_{i,r}||}{||\Omega_{i,r} \cup \Omega_{j,r}||} \right) \tag{5.7}$$

4. **Pseudo document similarity.** This similarity kernel has been successfully applied to finding entity synonyms [20]. Here we first define what is a pseudo document. For a real document, the title and body are not always easy to get. Thus we use the set of queries having clicks on a document as its representation. And this set of queries is called pseudo document. The Pseudo document similarity esentially measures the similarity between two attribute values based on the number of co-occurrences in the query-clicked pseudo document pairs. For example, for attribute values "IBM" and "Big blue", if query q contains "IBM", and it has clicks on a set of pseudo documents $\mathbb{D}$. The percentage of $\mathbb{D}$ which contains "Big blue" measures how similar these two values are. Please refer to [20] for more detail.

5.3.2 Basic Model

After defining the overall distance function and similarity kernels, we now describe the formulation of the clustering model. As for a clustering model, we must specify the cluster centers. For the attribute synonym finding problem it's natural to nominate the canonical attribute values as the cluster centers since they should be close to their synonyms. Moreover, synonymous attribute values should be close with each other in a cluster and far away from other clusters, which motivates our **compact clustering**. Formally, in the basic model we aim at minimizing the following objective function:

$$
\begin{aligned}
& g_0(R, Z, W) && (5.8) \\
&= \sum_{k=1}^{K} \sum_{i=1}^{N} r_{i,k} \cdot d(x_i, z_k) + \sum_{i=1}^{N} r_{i,0} \cdot d(x_i, z_0) \\
&= \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{t=1}^{T} r_{i,k} \cdot w_t^2 \cdot d_t(x_i, z_k) + \sum_{i=1}^{N} r_{i,0} \cdot \gamma
\end{aligned}
$$

subject to:

$$
\begin{cases}
\sum_{k=0}^{K} r_{i,k} = 1, 1 \leq i \leq N \\
r_{i,k} \in \{0, 1\}, 1 \leq i \leq N, 0 \leq k \leq K \\
w_t \geq 0, \sum_{t=1}^{T} w_t = 1, 1 \leq t \leq T
\end{cases}
\tag{5.9}
$$

The above objective function minimizes the sum of within-cluster dispersions. In *Eq.* (5.8), the first term is the overall within-cluster distances of the normal clusters, and the second term is the within-cluster distances in the background cluster. Such formulation is to make the resulting clusters more compact. Note that in our model there is no need to represent data points with explicit feature vectors, instead, we only require that $d(x_i, x_j) \geq 0$. The notations of variables in the formula are listed below:

- $d(x_i, z_k)$ is the overall distance function between x_i and z_k, as defined in *Eq.* (5.1);
- R is an $N \times (K+1)$ partition matrix, where N is the total number of points and $K+1$ is the number of clusters; $r_{i,k} \in \{0, 1\}$ is a binary variable; $r_{i,k} = 1$ indicates object x_i is in k^{th} cluster;

- $Z = \{z_0, z_1, ..., z_K\}$ are the medoids of the clusters. In the basic model, the first K medoids are fixed to the target attribute values $\{v_1, v_2, ..., v_K\}$ for which we look for synonyms;
- $W = \{w_1, w_2, ..., w_T\}$ are the weights of different distance kernels;
- γ is a constant measuring the distance of $x \in \mathbb{X}$ to the background cluster.

Rationale of the objective function: The above objective function is similar to the formulation of K-medoids[48]. The advantage of employing the K-medoids framework rather than K-means is that the distance function between data points can be defined in arbitrary form. However, there are important differences between our basic model and the K-medoids model: firstly, the first K medoids in our model are fixed to the canonical attribute values, assuming they are best representatives of these clusters. This also implies that there is no need to update the medoids. Secondly, in our model the distance between points is a weighted distance function, which is very different from the standard K-medoids model. Such weights measure the relative contribution of the kernels, and they are estimated in an unsupervised manner. Thirdly, in our model we add a background cluster in order to attract the random points. And we assume that the distance of any point to the background cluster is a constant. Although the basic compact clustering model can partition the data points into synonym clusters, it suffers from the following limitations: (1) Using a single fixed representative for a cluster may be problematic. First, the canonical value is not always the most popular or most representative. It may have idiosyncrasies that are not shared by other members of the cluster. Second, because the similarity features are noisy, if we only compare a candidate against the canonical value, a noisy feature may bias it towards an incorrect cluster. (2) Manually setting the constant γ is very difficult. Nominating a good γ at the beginning is hard; and further, since the distance between data points depends on the weights, it makes it even harder to choose the appropriate γ inside the algorithm. Therefore a rough estimation of this constant is desirable. (3) No measurement of uncertainties of a point belonging to the background. We can adjust the precision/recall of our model via the constant γ. However, as γ is hard to set manually, it's beneficial to have another parameter for setting the prior probability of a point belonging to background, so as to adjust the operating point in the precision/recall curve.

5.3.3 Standard Model

Generally, limitation 1 can be addressed by employing a flexible representative or a small set of representatives for each cluster. However it's not desirable to have flexible medoids since in our problem setup the canonical values are good representatives and it is more robust to include them into the medoids. Therefore we propose to use a small subset of points, including the canonical value, to form a new pseudo-medoid. The subset is viewed as a committee that determines which other points belong to the cluster. A similar idea of clustering with committees of points has been successfully applied to the document clustering problem [70]. As the optimal solution for K-medoids cluster is NP-hard, we can only find the local optimum of medoids by algorithms such as PAM [49]. However, this algorithm takes $O(m^2)$ time to update a medoid where m is the number of points in a cluster, which is inefficient. Also it doesn't take the advantage of the canonical values. In our new proposal, we form the new pseudo-medoid by including the $L-1$ most similar values to the canonical value as well as the canonical value itself. The advantage of this nearest neighbors approach is that it forms a very compact pseudo-medoid around the canonical value efficiently (requiring only $O(m)$ time).

To address the limitation 2, we propose to randomly select μ proportion of points from the background cluster, and estimate γ by taking the average of the distance from x to this random subset. Results show that the final synonyms are stable with respect to different setting of μ.

We address the limitation 3 by introducing a prior probability p that a given point x belongs to the background cluster. If we further assume that x follows a uniform prior distribution for normal clusters, then the prior probability of x belonging to a normal cluster is $\frac{1-p}{K}$.

Based on these new proposals, we present the **standard compact clustering** model by minimizing the updated objective function:

$$\begin{aligned}
&g_1(R, Z', W) \\
=&\frac{1-p}{K}\sum_{k=1}^{K}\sum_{i=1}^{N} r_{i,k} \cdot d(x_i, z'_k) + p\sum_{i=1}^{N} r_{i,0} \cdot d(x_i, z_0) \\
=&\frac{1-p}{K}\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{x_j \in z'_k}\sum_{t=1}^{T} \frac{1}{|z'_k|} \cdot r_{i,k} \cdot w_t^2 \cdot d_t(x_i, x_j) \\
&+ p\sum_{i=1}^{N}\sum_{x_j \in A}\sum_{t=1}^{T} \frac{1}{|A|} r_{i,0} \cdot w_t^2 \cdot d_t(x_i, x_j)
\end{aligned} \tag{5.10}$$

subject to *Eq.* (5.9). Where z'_k is the pseudo-medoid, A is the subset of random points in the background cluster, whose size is controlled by the parameter μ. And the prior probability p is a tunable parameter. The standard compact clustering model aims at inducing more compact clusters.

5.3.4 Solving the Standard Model

In the standard model there are three sets of unknown variables: R, Z' and W, which are dependent on each other. There is no exact solution to solve all of them at the same time. Instead we solve this optimization problem by iteratively solving the following minimization problems:

1. Fix $Z' = \hat{Z}'$ and $W = \hat{W}$; find the best R that minimizes $g_1(R, \hat{Z}', \hat{W})$
2. Fix $W = \hat{W}$ and $R = \hat{R}$; find the best medoids Z' that minimizes $g_1(\hat{R}, Z', \hat{W})$
3. Fix $Z' = \hat{Z}'$ and $R = \hat{R}$; solve the best parameters W that minimizes $g_1(\hat{R}, \hat{Z}', W)$

Sub-problem 1 (cluster assignment) can be solved by:

$$\begin{cases} r_{i,k} = 1 \quad \text{if} \quad d'(x_i, z'_k) \le d'(x_i, z'_l), 0 \le k, l \le K \\ r_{i,k} = 0 \quad \text{otherwise} \end{cases} \tag{5.11}$$

where

$$\begin{cases} d'(x_i, z'_k) = \dfrac{1-p}{K} \cdot d(x_i, z'_k) \quad \text{if} \quad k > 0 \\ d'(x_i, z'_k) = p \cdot d(x_i, z'_0) \qquad \text{if} \quad k = 0 \end{cases}$$

For **sub-problem 2**, we update the pseudo-medoids of first K clusters by including up to the top $L-1$ most similar values to the canonical value as well as the canonical value itself:

$$z'_k \leftarrow v_k \cup \{L-1 \ \text{ nearest neighbors of } \ v_k \ \text{ in cluster } \ k\} \tag{5.12}$$

For the background cluster, there is no need to calculate the updated medoid. We follow the basic ideas from weighted K-means [21] to solve **sub-problem 3**. Because after fixing R and Z', *Eq.* (5.10) is a convex quadratic function, we apply the Lagrange Multiplier method and obtain a closed form solution to W as:

$$\hat{w}_t = \frac{1}{\sum_{j=1}^{T} \frac{D_t}{D_j}} \tag{5.13}$$

where

$$\begin{aligned} D_t = & \frac{1-p}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{x_j \in z'_k} \frac{1}{|z'_k|} \cdot r_{i,k} \cdot w_t^2 \cdot d_t(x_i, x_j) \\ & + p \sum_{i=1}^{N} \sum_{x_j \in A} \frac{1}{|A|} r_{i,0} \cdot w_t^2 \cdot d_t(x_i, x_j), \end{aligned}$$

and $D_t \neq 0$ for $1 \leq t \leq T$. Intuitively, a larger weight is assigned to a feature function which makes the clusters more compact (with smaller sum of within-cluster dispersions).

The optimal allocation of points to clusters and the best distance kernel weights can be found by iteratively solving the above sub-problems. This algorithm procedure assumes that we have a set of candidate strings $\{x_1, x_2, ..., x_N\}$ as input. Here we describe how we obtain this set of candidates efficiently. For the synonym identification problem, reducing the search space is critical since the set of all potential candidates are all arbitrary strings. In this work, we confine the search space using a large query log from a commercial search engine. Given a set of canonical values $\mathbb{V} = \{v_1, v_2, ..., v_K\}$, we identify candidates as follows:

1. For each $v \in \mathbb{V}$, get the top queries which contain the target canonical value v from a compact Trie data structure [39]. For each query q in this set, get the top clicked documents $\mathbb{D}$. Then for each d in $\mathbb{D}$, get the most clicked queries which also have clicks on these documents. This process is basically a one-step forward-backward random walk;

2. Once the set of coclicked queries have been found, we generate all n-grams from these coclicked queries, and filter the n-grams with too low counts. The rest are maintained as initial candidates;

3. For each similarity kernel, from the initial candidates we select the top M (M=100 in this work) candidates with highest similarities to the canonical attribute value;

4. Merge all top candidates and remove duplicate ones to form the final candidates.

5.3.5 Incorporating Additional Information

So far, we have utilized the principle of compactness to learn attribute synonyms as well as kernel weights W without requiring any additional labeled data beyond the canonical values. In some situations, we may have some labeled synonym data. Ideally, a fair amount of labeled data can help boost the performance of the model. Yet such labeled data is often difficult to collect, as domain knowledge is required to annotate synonyms. On the other hand, there are other forms of labels, which are noisy but easier to collect. For the problem of synonym identification, one source of such additional information is the Wikipedia redirect. A redirect usually indicates synonymous or strong relationship between two strings. However, a redirect may also provide incorrect synonyms, such as ones from another interpretation of the canonical value. For instance, "Apple Fruit" and "Apple Computer Inc." both have redirects to "Apple". However for the domain of computer companies, "Apple Fruit" is not a synonym for "Apple". Specifically the Wikipedia redirects provide synonym evidence as follows:

- $Q = \{Q_k\}_{k=1}^{K}$, and for each target attribute v_k:
  $Q_k = \{(x_i, v_k)$ — $x_i \in \mathbb{X}$, x_i and v_k have Wikipedia redirects that $x_i \rightarrow v_k\}$. Q_k provides a small subsets of points known to belong to the same cluster as v_k;

We call this information the should-link constraints. In addition, because we are looking for synonyms on a set of attribute values, these values together provide another form of information: the should-not-link constraints:

- $D = \{D_k\}_{k=1}^{K}$, for each target attribute v_k:
  $D_k = \{(x_i, v_k)$ — $x_i \in \mathbb{X}$, x_i and v_k should not be in the same cluster$\}$.

The should-not-link constraints suggest that a canonical value and its lexical variants (*i.e.* spelling variants) should not be in the same cluster as other canonical values. For example, "nike" is not likely a synonym of "adidas" although they might share high coclick similarity. With the additional information in the forms of Q and D, we propose the **a-compact clustering** (compact clustering with additional information) model that extends the standard model to include information from Q and D as regularization terms. With this extension, the objective function becomes:

$$\begin{aligned} g_2(R, Z', W) =& g_1(R, Z', W) + \gamma_1 \cdot \sum_{k=1}^{K} \sum_{x_i \in Q_k} d(x_i, v_k) \\ & - \gamma_2 \cdot \sum_{k=1}^{K} \sum_{(x_i, x_j) \in D_k} d(x_i, v_k) \end{aligned} \tag{5.14}$$

subject to *Eq.* (5.9). Where γ_1 and γ_2 are the coefficients of these regularization terms. This new objective function tries to minimize the overall within cluster distances, minimize the sum of dispersion contributed from pairs in Q, and maximize the sum of dispersion from pairs in D. The a-compact cluster objective function is still convex as long as the coefficient of $W_t^2 \geq 0$, so it has exact solution for W in sub-problem 3. We don't go into the details of solving the sub-problems since they are similar to the standard model.

5.4 Experiments and Results

We have conducted a series of experiments on datasets across multiple categories to test the effectiveness of our proposed compact clustering model. We first make direct comparison of our model to the baselines on the traditional setting that the attribute values are also entities mentions. We then conduct another set of experiments on the setting that the attribute values are arbitrary strings. After that, we will show results in cases where the attribute values have ambiguous senses. Furthermore, we investigate the relative importance of similarity kernels. Finally we discuss the sensitivity of our model when parameters change and when additional noisy labels from Wikipedia are available.

5.4.1 Datasets

The datasets consist of two major parts, one is data sources from which the similarity functions are computed; the other is the test attribute values and the corresponding labeled synonyms. Firstly, all of the similarity kernels are computed on a large query log and a query-click log from Bing search engine. There are more than 100 millions unique queries in the query log and about 600 millions query-clicked document pairs in the query-click log. All these query and click logs are preprocessed and indexed in a compression trie data structure [39]. so that the similarities can be computed efficiently. For the noisy labeled data, we have collected all redirects from Wikipedia on January 2013. Secondly, in order to test the proposed models, we have collected several attribute synonym datasets from multiple categories (see Table 5.2). Specifically, 3 datasets are constructed to test the traditional entity synonym finding. 3 other sets are selected to test the synonym identification where the attribute values don't look like entity mentions. Furthermore, we have collected a set of ambiguous attribute values to discuss the challenging issue of ambiguity. Because obtaining a set of ambiguous values from a single category is hard, we get the results from 5 datasets, select 18 such ambiguous values and then label them.

Because it's almost impossible to annotate all true synonyms of the selected attribute values, we employ the TREC style pooling strategy to obtain the initial pool of candidate ground-truths. We first choose 3 competing methods, then for each attribute value we select, up to 50 best synonyms output from each approach are pooled. Domain experts are then asked to label whether they are true synonyms by majority voting.

Table 5.2: Test Datasets

Type	Dataset	#Values	#Labels	%Positive
entity mentions	movie.title	50	3272	15.9
	shoe.brand	50	3370	17.2
	doctor.specialty	50	2105	12.5
arbitrary strings	shoe.gender	5	96	19.8
	babyclothing.age	15	129	21.0
	movie.genre	21	340	15.4
ambiguous values	shoe.brand	6	410	16.8
	movie.title	3		
	movie.genre	3		
	itcompany.name	3		
	insurance.provider	3		

5.4.2 Evaluation Metrics

We evaluate our system based on the evaluation metrics of expected precision, expected recall and expected F1 measure. Specifically, for an canonical attribute value $v \in V$, $O(v) = \{o_1, o_2, ..., o_{K_v}\}$ is the set of synonym outputs in ranked order by similarity score. Let $S(v)$ denote the set of true synonyms annotated by the human experts for v. Expected precision is computed as:

$$precision = \frac{1}{|V|} \sum_{v \in V} \sum_{o \in O(v)} I_p(o, v)/|O(v)|$$

where $I_p(o, v) = 1$ if $o \in S(v)$, and 0 otherwise. Expected recall is defined as:

$$recall = \frac{1}{|V|} \sum_{v \in V} \sum_{x \in S(v)} I_r(O(v), x)/|S(v)|$$

where $I_r(O(v), x) = 1$ if $x \in O(v)$ for $x \in S(v)$, and 0 otherwise. The F1 measure can be computed accordingly.

5.4.3 Baselines

1. ***Individual features***. Individual features are included as baselines so as to reveal their strength and weakness on identifying entity attribute synonyms both in the form of entity mentions as well as arbitrary strings. Synonyms are identified by single attribute value at a time. We try several settings and manually choose the best thresholds for these feature functions.

2. ***Chakrabarti-2012***. We also include a strong baseline proposed by Chakrabarti *et. al* [20], which identifies entity synonyms by combining multiple similarity scores with manually tuned thresholds. We consider it a state-of-the-art multi-feature, single value at a time approach. For a fair comparison, this system works on the same set of query log and clickthroughs as our approach for calculating similarities. We collect final outputs in the form of an unordered list of synonyms for each input attribute value

via the system interface provided by the authors of [20].

3. ***Clustering with Fixed Weights***. In order to reveal the effectiveness of the kernel weights learning, we add a baseline that uses the same clustering model, yet with fixed (equal) kernel weights. This way of combining the weights are similar to that in *Chakrabarti-2012*.

5.4.4 Entity Mentions

We first evaluate the performance of the compact clustering model on attribute values that are also entity mentions. It is important to investigate our model's performance on this setting because it is a classical setting and a good model should at least be competitive in this setting. Among the three test datasets, *movie.title* and *shoe.brand* are from popular domains while *doctor.specialty* is from tail domain since the values such as "interventional cardiology" are not common query terms. Table 5.3 shows the expected precision, recall, and F1 scores. Firstly, the results show consistently across three datasets that using single feature doesn't achieve competitive results. Specifically, *categorical pattern* has somewhat good precision but suffers from very low recall. *pseudo document* similarity is a relatively robust method achieving balanced precision and recall. However it fails to get competitive performance compared to methods combining multiple features such as *Chakrabarti-2012* and our model. Secondly, the *Chakrabarti-2012* approach achieves relatively high on precision but low on recall, confirming its precision orientated nature. Thirdly, learning synonyms jointly in our clustering framework clearly demonstrates advantages: it achieves better F1 scores than *Chakrabarti-2012* across three datasets by simply fixing the weights to be all equal. Finally, our proposed *compact clustering* model is consistently obtaining balanced precision and recall, resulted in best F1 scores. It clearly outperforms the baseline of clustering with fixed weights, showing the benefit of automatic weight learning. Moreover, its F1 score also consistently outperforms *Chakrabarti-2012*. In fact, in two of the three datasets, the statistical T-Test indicates that there is statistically significant difference between our model and *Chakrabarti-2012* at confidence level $p = 0.01$. This reveals the effectiveness of our proposed model that identifies synonyms jointly with kernel weights automatically tuned.

Table 5.3: Attribute Values as Entity Mentions

Dataset	Method	Precision	Recall	F1
movie.title	categorical pattern	0.463	0.202	0.2811
	coclick	0.381	0.405	0.393
	lexical context	0.398	0.372	0.385
	pseudo document	0.412	0.437	0.424
	Chakrabarti-2012	**0.706**	0.400	0.470
	clst. w. fixed weights	0.503	0.525	0.514
	compact clustering	0.541	**0.572**	**0.556***
shoe.brand	categorical pattern	0.455	0.258	0.329
	coclick	0.425	0.446	0.435
	lexical context	0.431	0.418	0.424
	pseudo document	0.454	0.477	0.465
	Chakrabarti-2012	0.762	0.470	0.545
	clst. w. fixed weights	0.713	0.510	0.595
	compact clustering	**0.768**	**0.560**	**0.647***
doctor.specialty	categorical pattern	0.398	0.19	0.257
	coclick	0.359	0.337	0.348
	lexical context	0.365	0.328	0.346
	pseudo document	0.380	0.359	0.369
	Chakrabarti-2012	**0.683**	0.520	0.590
	clst. w. fixed weights	0.665	0.543	0.598
	compact clustering	0.673	**0.558**	**0.610**

Note: The F1 score marked by * means it has statistically significant difference compared to *Chakrabarti-2012* at confidence level $p = 0.01$.

5.4.5 Arbitrary Strings

We then compare the results on attribute values that don't look like entity mentions. Such values include interesting instances like infant, women in *shoe.gender*, thriller in *movie.genre*, 2 year in *babyclothing.age*. We summarize the results in Table 5.4. As expected, *categorical pattern*, *pseudo document* behave similarly as in the previous experiment, confirming using them individually is not effective in both forms of attribute values. Also, the clustering with fixed weights performs slightly better than *Chakrabarti-2012*. Further, the compact clustering model achieves significantly better results than *Chakrabarti-2012* across three datasets. And the difference between our model and *Chakrabarti-2012* is statistically significant. We list some interesting cases that our proposed model identifies successfully but *Chakrabarti-2012* fails. For example, for infant, *Chakrabarti-2012* identifies {newborns, neonates, baby, infants etc.} as it's synonyms; not only can our method identify all of them, but it also finds more interesting synonyms such as {toddler, toddlers} which are more specific synonyms in the domain of *shoe.gender*. And for thriller, *Chakrabarti-2012* finds "michael jackson thriller" as its synonym while compact cluster doesn't. In fact, "michael jackson thriller" is not the synonym of thriller in the particular domain of *movie.genre*. And our model identifies "scary" as its synonym, which is more appropriate. The superior performance of compact clustering might be due to two reasons: first is that we aggregate all referent strings of the attribute value as proxies, therefore resulting in more robust estimate of similarity measures. And second, the joint modeling of multiple attribute values from the same implied domain effectively handles the ambiguity problem, which we will further discuss below.

5.4.6 Ambiguous Attribute Values

Ambiguous synonyms handling is important for finding domain specific synonyms. Here we compare our model to *Chakrabarti-2012* on a set of attribute values that are ambiguous. They include {jordan, coach, lv} from *shoe.brand*, {app, sun, adobe} from *itcompany.name*, {aarp, advantage, aim} from *insurance.provider*, {thriller} from *movie.genre*, {matrix} from *movie.title*. Results on Table clearly indicate that compact clustering is much more effective than *Chakrabarti-2012* on handling ambiguous attribute values. Interestingly, the baseline of clustering with fixed weights also significantly outperforms *Chakrabarti-2012* in this case, suggesting that joint modeling of multiple attribute values is particularly effective for ambiguous synonyms handling.

Table 5.4: Attribute Values as Arbitrary Strings

Dataset	Method	Precision	Recall	F1
shoe.gender	categorical pattern	0.371	0.179	0.242
	coclick	0.343	0.362	0.352
	lexical context	0.360	0.336	0.348
	pseudo document	0.370	0.400	0.384
	Chakrabarti-2012	0.489	0.372	0.423
	clst. w. fixed weights	0.485	0.502	0.493
	compact clustering	**0.500**	**0.550**	**0.524***
babyclothing.age	categorical pattern	0.382	0.190	0.254
	coclick	0.412	0.455	0.432
	lexical context	0.462	0.421	0.441
	pseudo document	0.401	0.544	0.462
	Chakrabarti-2012	0.592	0.380	0.463
	clst. w. fixed weights	0.562	0.466	0.510
	compact clustering	**0.669**	**0.543**	**0.599***
movie.genre	categorical pattern	0.355	0.140	0.201
	coclick	0.325	0.355	0.340
	lexical context	0.343	0.312	0.327
	pseudo document	0.331	0.362	0.346
	Chakrabarti-2012	0.591	0.482	0.531
	clst. w. fixed weights	0.580	0.554	0.567
	compact clustering	**0.594**	**0.588**	**0.591***

Note: The F1 score marked by * means it has statistically significant difference compared to *Chakrabarti-2012* at confidence level $p = 0.01$.

Table 5.5: Ambiguous Attribute Values

Method	Precision	Recall	F1
Chakrabarti-2012	0.581	0.465	0.517
clst. w. fixed weights	0.613	0.574	0.593*
compact clustering	**0.677**	**0.582**	**0.626***

Note: The F1 score marked by * means it has statistically significant difference compared to *Chakrabarti-2012* at confidence level $p = 0.01$.

5.4.7 Contribution of Similarity Kernels

Our proposed model is able to learn the weights of similarity kernels. In this experiment we look into the learnt weights to see whether they reflect the relative importance of the similarity kernels. For this purpose, we have conducted the ablation test, in which we **remove** one similarity kernel at a time and run the model. We also report the weights learnt without removing any kernels. Results on three domains are shown in Table 5.6. These results indicate that pseudo document similarity seems to play relatively higher importance than other kernels. For example, both in *movie.title* and *doctor.specialty*, it carries the highest weights; and the F1 measures drop to the lowest when removing this kernel (the lowest F1 is marked in bold). Interestingly, the categorical pattern similarity plays an important role in *shoe.brand*. Note that in this domain there are more ambiguous inputs (6 values) than other domains, suggesting the importance of categorical pattern similarity for disambiguation.

Table 5.6: Relative Importance of Similarity Kernels

Dataset	W/F1	categorical pattern	coclick	lexical context	pseudo document
movie.title	W	0.20	0.20	0.27	0.33
	F1	0.505	0.510	0.489	**0.464**
shoe.brand	W	0.29	0.16	0.25	0.3
	F1	**0.569**	0.601	0.589	0.573
doctor.specialty	W	0.13	0.22	0.30	0.35
	F1	0.573	0.567	0.55	**0.538**

5.4.8 Adding Noisy Labeled Data

The following experiment aims at quantifying the improvement from utilizing additional information with the a-compact clustering model over one without such information. In this experiment we add *Wikipedia redirects* as a weak baseline in which the redirects are treated as predictions. Due to the space limitation, we again only summarize the results on datasets of the entity mentions in Table 5.7. As expected, the F1 measure of the a-compact clustering model significantly outperforms Wikipedia redirects and Chakrabarti-2012 methods on these datasets. Similarly, compared to the compact clustering model, the a-compact clustering model achieves substantially better F1 on all datasets. These results demonstrate that the a-compact clustering model effectively exploits the information from the additional noisy signals.

Table 5.7: a-Compact Clustering Model vs. Baselines

Dataset	Method	Prec	Recall	F1
movie.title	Wikipedia redirects	**0.847**	0.178	0.294
	Chakrabarti-2012	0.706	0.400	0.470
	compact clust	0.541	0.572	0.556
	a-compact clust	0.609	**0.649**	**0.594***
shoe.brand	Wikipedia redirects	**0.820**	0.186	0.303
	Chakrabarti-2012	0.762	0.470	0.545
	compact clust	0.768	0.560	0.647
	a-compact clust	0.802	**0.686**	**0.712***
doctor.specialty	Wikipedia redirects	**0.790**	0.22	0.344
	Chakrabarti-2012	0.683	0.52	0.590
	compact clust	0.639	0.544	0.587
	a-compact clust	0.685	**0.621**	**0.651***

Note: The F1 score marked by * means it has statistically significant difference compared to *Chakrabarti-2012* at confidence level $p = 0.01$.

5.5 Conclusions and Future Works

For the problem of finding entity attribute synonyms, we propose a **compact clustering** framework to simultaneously identify synonyms for a set of attribute values. In this framework, multiple sources of information are integrated into a kernel function between attribute values and synonyms are learned via unsupervised clustering. We have also proposed a novel similarity kernel called Categorical Pattern Similarity, which has proven to be effective for improving the performance of the compact clustering model. Furthermore, the clustering performance can be enhanced by leveraging limited amount of additional information. Extensive experiments demonstrate the effectiveness of our clustering framework over previous approaches for identifying entity attribute synonyms, both in the cases where they are entity mentions or are arbitrary strings. We have also demonstrated the effectiveness of our model for ambiguity handling for identifying domain specific synonyms.

Further, besides attribute value synonym identification, our unsupersived framework of simultaneously modeling multiple inputs and integrating multiple kernels can be potentially applied to other applications, such as looking for related queries, product recommendation, question paraphrasing *et. al.*.

Chapter 6

Modeling Query Auto-completion by a Two-dimensional Click Model

6.1 Introduction

The previous levels of query understanding mentioned in this thesis is somewhat static, meaning that we have to know the entire query in advance. However in many scenarios this is not possible: the users want to be assisted when they just give a tiny amount of query hint, which is called dynamic query understanding. One example is to predict users intended queries based on partial queries in the task of query auto-completion. In the chapter we focus on modeling the query auto-completion.

Query auto-completion (QAC) is one of the most important components of a modern web search engine which facilitates faster user query input by predicting the users' intended queries. It is offered by most of the search engines, e-commerce portals and major browsers. With the prevalence of mobile devices, it becomes more critical because typing takes more effort in mobile devices than in PCs. Previous studies addressed the QAC problem in different perspectives, ranging from designing more efficient indexes and algorithms [12, 38, 96, 41], leveraging context in long term and short term query history [10], learning to combine more personalized signals such as gender, age and location [79], suggesting queries from a mis-spelled prefix [29].

The query auto-completion process starts when a user enters the first character into the search box. After that, she goes through a series of interactions with the QAC engine until she clicks on an intended query. Such interactions include examining the suggested results, continuing to type, and clicking on a query in the list. Although previous approaches model the relevance ranking with many features, these models are usually only trained on the final submitted queries, ignoring the entire user interactions from the first character she types to the final query she has clicked.

One difficulty for improving the QAC quality is the lack of data about fine-grain user interactions in QAC. Recent work has attempted to leverage all prefixes of a submitted query [10, 79]. However the only available data is the submitted query; while the prefixes are *simulated* from all possible prefixes of the query. Lack of associated information, such as the suggested list, user typing speed and other real user interactions, prevents such methods from further improving their performance.

For this purpose, we have collected a high-resolution QAC dataset from both PC and mobile phones, in which each keystroke of a user and the system response are recorded. As far as we know, this is the *first* dataset with this level of resolution specifically for QAC. Extensive studies have already demonstrated the importance of query log for web document retrieval [74, 42, 5, 44, 78]. Therefore, it is reasonable to believe this new kind of QAC log could potentially enable a full spectrum of researches for QAC, such as user behavior analysis, relevance ranking, interactive system design for QAC, just to name a few.

Given many possibilities for mining this new data, in this chapter we focus on leveraging it for understanding users' behavior in QAC. Specifically based on our QAC log, we have observed a phenomenon that in QAC users frequently skip several suggestion lists, even though such lists contain the final submitted query. The exact reason why this happens and how frequent it happens is largely unknown. Besides, we also observed that most of the clicked queries are concentrated at top positions. Better understanding of these behaviors has a strong implication to the relevance modeling. For instance, we assume that a user does not click a suggested query due to the lack of relevance; however the skipping behavior complicates this hypothesis. So, if we know the positions where such skipping behavior happens, we could improve the candidate ranking in QAC by taking into account the examples in the positions where they are more likely to be examined. Despite its importance, little research has been done in explaining such behaviors.

The QAC process shares similarities with the web document retrieval: in QAC people look for intended queries with a prefix, while in document retrieval people look for relevant documents with a query. And in document retrieval, click models are widely used to model the users' examination and clicking behavior [75, 26, 31, 62, 105, 22]. Thus we could potentially adopt an existing click model to shed light on the QAC user behavior. However, there are major differences between the QAC process and document retrieval. For example, in document retrieval a user usually examines one result page before she lands on a click, while in QAC she usually types in a series of prefixes and examines multiple lists of suggestions before landing on a click. Due to these differences, most current click models are not applicable to the QAC problem without significant modification.

Therefore in this work we propose a novel two-dimensional click model for understanding the user behaviors in QAC. This click model is consisted of a horizontal component that captures the skipping behavior, a vertical component that depicts the vertical examination bias, and a relevance model that reflects the intrinsic relevance between a prefix and a suggested query.

We have performed a set of experiments on our QAC datasets from PC and mobile phones. Results show that our proposed model can effectively model the user behavior in QAC. The resulting relevance model significant improves the QAC performance over existing click models. We also show that the learned knowledge about users' behavior, especially the probability of skipping a column of suggestion candidates, could serve as labeling information to improve the performance of existing learning-based approaches. Furthermore, with the learned model we demonstrated some interesting insights of the user behaviors in QAC on both platforms.

We summarize our contributions as follows:

- We have collected the first set of high-resolution query log specific for the QAC process, which could enable many studies on deeper understanding of QAC.

- Based on the new QAC log, we analyze two important types of user behavior in QAC, namely the horizontal skipping bias and vertical position bias. The horizontal skipping bias is unique to QAC and is formally introduced here for the first time.
- We propose a novel Two-Dimensional Click Model to model these types of user behavior. Our model outperforms the state-of-the-art click models on relevance ranking. We also utilize our model to derive interesting insights about the QAC user behavior on PC and mobile devices.

6.2 Data and User Behavior Analysis

In this section we will introduce the high-resolution query log for QAC and the user behavior analysis based on this new kind of data.

6.2.1 A High-Resolution QAC Log

As mentioned above, the QAC process is under-explored because there is no appropriate dataset. Previous studies rely on search query log in which only the submitted query and associated information are recorded. In order to analyze the subtle user behavior in a whole QAC process we need to record system response and user interactions for each keystroke leading to the final clicked query. With this motivation we have collected a large set of QAC sessions with real user interactions from the Yahoo! search engine. This QAC log contains millions sessions on PC and mobile phone platforms. The dataset in this study is a random sample of the original QAC log dating from Nov. 2013 to Jan. 2014.

As illustrated in Table 6.1, the recorded information in each QAC session includes the final clicked query, every keystroke a user has entered, timestamp of a keystroke, corresponding top 10 suggested queries to a prefix, and the anonymous user ID. It also records the user's submitted query in previous session. Table 6.2 lists the basic statistics of the dataset studied in this work. In PC platform each session contains 11.80 prefixes in average; while the average clicked query length is 19.68, which is substantially larger than the average prefix length, indicating the usefulness of QAC for facilitating faster user query input. We observe similar statistics on the iPhone 5 platform, with lower average prefixes in one session (9.43), suggesting that users rely even more on mobile devices where typing takes more effort.

Table 6.1: High-resolution QAC Log

Data Type	Example
anonymized user id	9qtfnj195p5ta
session id	rFzqRUgeurCd
time stamp	11/02/2013 23:02
prefix	oba
final submitted query	obama care
previous query	charm and charlie's
clicked URL	https://www.healthcare.gov/
top 10 suggested queries	obama care—obama—oba— obamacare—obama approval rating—...

Table 6.2: Dataset Basic Statistics

Platform	# Sessions	Ave Prefixes	Ave Clicked Qry Len	# Unique User IDs
PC	125,392	11.80	19.79	111,783
iPhone 5	31,227	9.43	16.98	17,331

Significance: With this QAC log, for the first time we have the opportunity to look into the real user interactions at the level of every keystroke. Such high-resolution dataset, when combined with traditional query log about user demographics and query history, could enable many new researches on QAC. For example, we could potentially utilize all lists of suggested queries to improve the QAC relevance ranking. Also, we could leverage this data to get better understanding of user behavior in the QAC process.

6.2.2 QAC User Behavior Analysis

Given the new QAC log, there are many possibilities to mine valuable knowledge. In this work we aim at leveraging the data for user behavior modeling in QAC. When a user clicks on a suggested query with the help of a QAC engine, she undergoes a series of interactions

Table 6.3: An Example of the Skipping Behavior

Prefix	obamacar— ⇒	obamacare— ⇒	obamacare — ⇒	obamacare h—
q_1	obamacare glitches	obamacare glitches	obamacare glitches	obamacare healthcare bill $\surd$
q_2	obamacare	obamacare	obamacare healthcare bill $\surd$	obamacare healthcare insurance
q_3	obamacare healthcare bill $\surd$	obamacare healthcare bill $\surd$	obamacare facts	obamacare health plan 2014
q_4	obamacare facts	obamacare facts	obamacare rates 2014	obamacare hotline
q_5	obamacare fines	obamacare fines	obamacare fines	obamacare health exchanges

Note: query with a $\surd$ mark is the final clicked query by the user.

with the QAC engine before she finally selects a preferred query. Such interactions are of great value for improving the quality of the QAC service. In this section we conduct two experiments to verify the need of modeling user behavior in QAC.

The first important user behavior in the QAC process is the skipping behavior. We have observed that a user frequently skips several intermediate lists of candidates even though these lists contain her final selected query. Table 6.3 illustrates this skipping behavior from a real user-interaction sample. In this example, clearly the query *obamacare healthcare bill* is preferred by the user. However, though this query is listed within top 3 positions in each of the suggestion list, the user has skipped all but the last appearance. A plausible explanation for the skipping behavior is that the user didn't examine it due to some reasons, such as fast typing speed, too deep to look up the query, *etc.*

We performed an experiment on the dataset described in Table 6.2 to verify how often the skipping behavior happens. In this experiment we define that a skipping behavior happens when the final clicked query is ranked within top 3 in the suggestion list of any of the prefixes except the final prefix. Results in Table 6.4 show that this behavior is frequent: it happens in 60.7% of all sessions in PC platform. Further, this behavior is consistent in all session groups with different final prefix length (57.6%, 64.8%, 59.1% and 60.2% respectively), indicating its prevalence in all queries. This result suggests a common skipping behavior in the QAC process. We observe very similar phenomenon in the iPhone 5 platform.

In another experiment, we investigated the vertical examination bias in QAC. Using the same set of QAC sessions, we computed the distribution of clicks according to their positions in the final suggestion list and the final prefix length. Figure 6.1 shows the 2-dimensional click distribution on both PC and iPhone 5 platforms. Similar to the findings

Table 6.4: Frequency of the Skipping Behavior

Category	# Sessions	% Sessions Having Skipping Behavior
All Sessions	125,392	60.7%
Sess with FPL in [1, 5]	39,405	57.6%
Sess with FPL in [6, 10]	39,882	64.8%
Sess with FPL in [11, 15]	22,892	59.1%
Sess with FPL in [16, 50]	23,213	60.2%

Note: FPL means Final Prefix Length.

in the traditional click models, most of the clicks concentrate on top positions. In fact, 75.4% of clicks is located within the top 2 positions on PC and 77.5% on iPhone 5. Such vertical positional bias suggests that we should boost the estimated relevance for queries which are clicked at lower ranks. Compared to PC, the clicks on iPhone 5 distribute more evenly with-in positions from 1 to 3. In addition, Figure 6.1 also indicates that most of the clicks are located in prefix length ranging from 3 to 12 on both platforms. Interestingly, the click probability at short prefix length (1 and 2) is very low, suggesting that users tend to skip the suggested queries at the beginning.

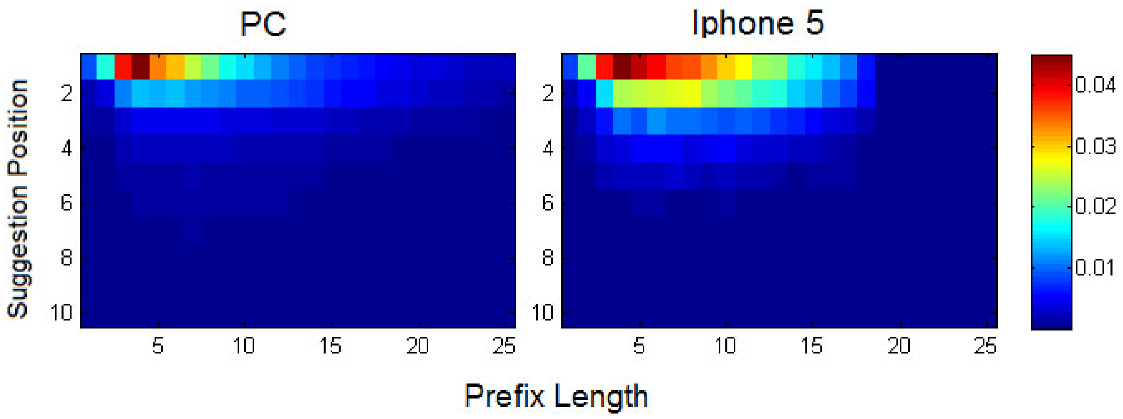


Figure 6.1: Distribution of clicks. Red color corresponds to high click probability, while blue corresponds to low click probability.

The reason why we focus on these two behaviors is their important implications to relevance ranking. Recent research attempts to improve the relevance model by training on all simulated columns (lists) of suggestions [10, 79]. However, not all of columns are examined by the users in reality. As a result, it might introduce many false negative examples that hurt the performance. To validate this claim, we have conducted an experiment on our QAC dataset from PC platform (see Section 6.4.1 for detailed description) in which we adopt the same training strategy as [79]. For the learning-to-rank algorithm, we use the RankSVM [42]. We also adopt very similar features as [79] (see Table 6.6). *MostPopularCompletion* (MPC) is used as a baseline. Another baseline is to train RankSVM only by the last column (suggestion list corresponding to the last prefix). In addition, we add the third baseline, which is also RankSVM, but trained by last 2 columns. Same as [79], we evaluate MRR across all columns where the final submitted query is within the candidates. Results in Table 6.5 indicate that training on all columns is inferior to the same model trained on last column. And it is even worse than the MPC baseline. Interestingly, the same model trained on only the last 2 columns achieves slightly better result than only using last column, suggesting that adding more (useful) columns might be beneficial. We hypothesize that columns that are likely to be examined are useful for training.

Table 6.5: A Pilot Experiment on Relevance Training

Method	MRR@All
RankSVM - trained by all columns	0.436
RankSVM - trained by last column	0.514
RankSVM - trained by last 2 columns	0.518
MPC	0.447

6.3 Modeling Clicks in Query Auto-Completion

Based on the results of the above experiments, we demonstrate that the skipping behavior and vertical click position bias are prevalent and important for improving QAC quality. How to model these behaviors is a new research problem. Given the similarity between QAC and document retrieval, first we sought to apply the existing click models to this problem. But we found most of these click models are not appropriate for the following reasons: (1) most existing click models only model a single query at a time. But in QAC, a session contains a series of prefixes that are correlated. (2) traditional click models are unable to model unseen query-document pairs. However in our QAC log we observe that a large portion (67.4% in PC and 60.5% in iPhone 5) of the prefix-query pairs are unseen. Therefore we propose a new click model for QAC, with emphasis on modeling these two types of user behaviors. We first formally define the assumptions on these two types of bias; then we will describe our click model in detail. After that we will discuss the parameter estimation via Expectation Maximization.

6.3.1 QAC Click Bias Assumptions

Here we define two basic assumptions for the QAC problem. One is to address the click bias due to the skipping behavior, and the other is to address the click bias on vertical positions.

- **SKIPPING BIAS ASSUMPTION:** A query will not receive a click if the user didn't stop and examine the suggested list of queries, regardless of the relevance of the query.

This assumption explains why there are no clicks to intermediate prefix even though a relevant query is ranked at the top of the list.

- **VERTICAL POSITION BIAS ASSUMPTION:** A query on higher rank tends to attract more clicks regardless of its relevance to the prefix.

Similar to the click modeling for document retrieval, this assumption explains why top ranked queries receive more clicks even though they are not necessarily relevant to the given prefix.

Table 6.6: Features of the H, D and R Models

Category	Feature	Feature Group	Description
H Model	CurrPosition	Prefix	Ratio of length between current prefix and the final prefix
	IsWordBoundary	Prefix	Binary indicator, whether the end of current prefix is at word boundary
	NbSuggQueries	Query	Number of suggested queries
	ContentSim	Query	Content similarity of suggested queries
	TypingSpeed	User	Typing speed at this keystroke
	QueryIntent	User	Whether the final submitted query is a navigational query
D Model	Is@Depth d	Query	Binary indicators, whether the query candidate is at depth $d, d = \{1, ..., 10\}$
R Model	MPC	Query	Candidate frequency computed based on past popularity
	TimeSense	Query	Candidate popularity measure in one day
	GeoSense	Query	Candidate popularity measure at the city where the query is issued
	QryHistFreq	User	The number of times the candidate is issued as query by the user in the past
	SameGenderFreq	Demographics	Candidate frequency over queries submitted by users of the same gender
	SameGenderLikelihood	Demographics	SameGenderFreq normalized by MPC
	SameAgeGroupFreq	Demographics	Candidate frequency over queries submitted by users of same age group
	SameAgeGroupLikelihood	Demographics	SameAgeGroupFreq normalized by MPC

6.3.2 Model Formulation

Based on the assumptions defined above, in this section we propose a Two-Dimensional Click Model (TDCM) to explain the observed clicks. This click model is consisted of a horizontal model (H Model) that explains the skipping behavior, a vertical model (D Model) that depicts the vertical examination behavior, and a relevance model (R Model) that measures the intrinsic relevance between the prefix and a suggested query. Figure 6.2 is a flowchart of user interactions under the TDCM model. The user interacts with the QAC engine horizontally and vertically according to the H, D and R models. Because in every QAC session, there is no click before the user leaves the process, we employ the Cascade Model assumption [26] that specifies the relations between the H, D and R models. We list the notations of TDCM in Table 6.7. According to the TDCM, the generative process of observing a click in a QAC session is described as follows (see Figure 6.2 also):

1. For a QAC session, let's assume the user has entered several characters and she is at prefix i, then she will decide whether to stop and look down to examine the list of suggested queries at ith column. This whether-to-look-down event is governed by a hidden random variable H_i, $H_i = 1$ means stop and examine, $H_i = 0$ means skip and continue to type. The task of the horizontal model (H Model) is to estimate the distribution of H: $P(H)$.

2. Once the user decides to examine vertically, following the cascade model assumption

she will examine one query at a time from top to bottom. The depth of the examination is determined by another hidden random variable D_i. $D_i = j$ means the user examines the query at position j at ith column. While being equivalent to introducing a set of binary variables at each depth, this formulation is more convenient in parameter estimation. The task of the vertical model (D Model) is to estimate the distribution of D: $P(D)$.

3. If a query candidate is examined and it is relevant, according to the cascade model assumption, the user will click it. The probability a query being relevant to the given prefix is determined by the relevance model: $P(C_{ij} = 1|H_i = 1, D_i \geq j)$. The task of the relevance model (R Model) is to estimate the distribution of $P(C_{ij} = 1|H_i, D_i)$.

4. If the depth D_i is reached and no relevant queries are found, she will go back to Step 1 and continue to type another character.

5. Once a click event happens, she will end the auto-completion session, which implies there will never be more than one click observed in a session.

Table 6.7: Major Notations

Symbol	Description
p_i	Prefix at ith column.
$q_{i,j}$	Query at position (i, j).
n	Number of columns in a QAC session.
H_i	Whether the user stops to examine the column i.
H	A vector of variables: $H = \{H_1..., H_n\}$.
D_i	Depth of examination at column i.
D	$D = \{D_1..., D_n\}$.
$C_{i,j}$	Whether the query at (i, j) is clicked.
C_i	A vector of variables: $C_i = \{C_{i,1}..., C_{i,M_i}\}$.
C	The click matrix: $C = \{C_1, ..., C_n\}$.
M_i	# queries in the suggestion list at column i.
w_H, w_D, w_R	Feature weights of the H, D and R model.
x_H, x_D, x_R	Features of the H, D and R model.

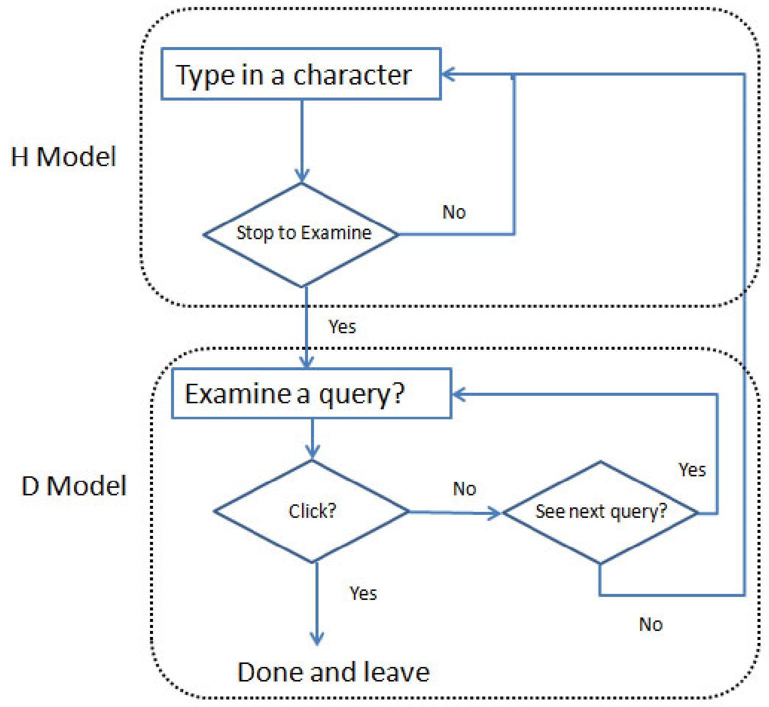


Figure 6.2: TDCM Flowchart

6.3.3 Click and Conditional Probabilities

Based on the above generative process, the probability of observing a click C in a session can be formulated as:

$$P(C) = \sum_{H,D} P(C, H, D) \tag{6.1}$$

In TDCM, $H = \{H_1..., H_n\}$, $D = \{D_1..., D_n\}$ are hidden variables. $C = \{C_1, ..., C_n\}$ is the click observation matrix in which only one click is observed: $C_{n,J} = 1$, n is the number of columns in the QAC session. Figure 6.3 depicts the relation between the hidden and observed variables. According to the Cascade Model assumption and the real observations of a QAC session, there is always only one click observed, which implies other columns don't receive any click:

$$C_{n,J} = 1 \Leftrightarrow \{C_1 = 0, ...C_{n-1} = 0, C_{n,J} = 1, C_{n,j} = 0, j \neq J\},$$

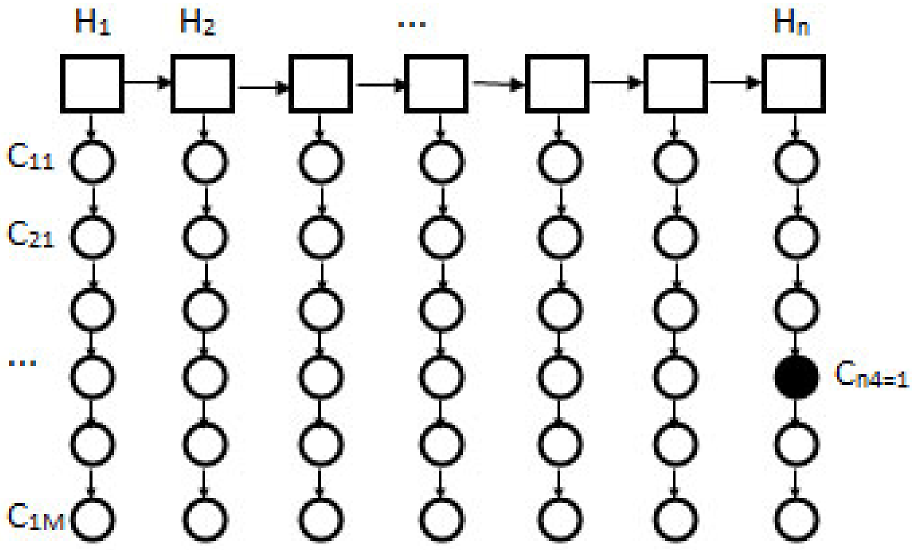


Figure 6.3: TDCM Model Structure

So:

$$P(C_{n,J}=1)=P(C_1=0,...C_{n-1}=0,C_{n,J}=1,C_{n,j}=0,j\neq J). \tag{6.2}$$

Our model also follows a set of conditional probabilities:

$$P(C_{ij}=1|H_i=0)=0 \tag{6.3}$$

$$P(C_{ij}=1|H_i=1,D_i<j)=0 \tag{6.4}$$

$$P(C_{ij}=0|H_i,D_i)=1-P(C_{ij}=1|H_i,D_i) \tag{6.5}$$

$$P(D_i>d|q_d:C_{n,d}=1)=0. \tag{6.6}$$

The TDCM assumption 1 (SKIPPING BIAS ASSUMPTION) is modeled by 6.3 and 6.6. The assumption 2 (VERTICAL POSITION BIAS ASSUMPTION) is modeled by 6.4 and 6.6. Equation 6.6 states that if a relevant query is ranked in depth d, the examination depth at ith column must not exceed d.

6.3.4 The Form of Distributions

Now we introduce the form of distributions for H, D and R model. Different from most of the click models and similar to [92], we define the distributions via logistic functions:

$$P(H_i = 1) = \sigma({w_H}^T \cdot x_H), \tag{6.7}$$

where $\sigma(z)$ is a logistic function: $\sigma(z) = \frac{1}{1+e^{-z}}$.

So

$$P(H_i = 0) = 1 - \sigma({w_H}^T \cdot x_H), \tag{6.8}$$

Similarly, for D_i, we have:

$$P(D_i = j) = \frac{e^{{w_D}^T \cdot x_{Dj}}}{\sum_{l=1}^{M_i} e^{{w_D}^T \cdot x_{Dl}}}, \tag{6.9}$$

where $j \in \{1, .., M_i\}$, M_i is the number of queries in the suggestion list at ith column.

And for the R model, we have:

$$P(C_{ij} = 1 | H_i = 1, D_i \geq j, \theta) = \sigma({w_R}^T \cdot x_{Ri,j}), \tag{6.10}$$

$$P(C_{ij} = 0 | H_i, D_i) = 1 - P(C_{ij} = 1 | H_i, D_i) \tag{6.11}$$

In the above formulations, x_H, x_D, x_R are features characterizing the H, D, R distributions. And $\theta = \{w_H, w_D, w_R\}$ are the corresponding weights for the features. As stated in [92], using this form of distribution has the advantage of incorporating more useful signals from diverse sources. And it also make it feasible for predicting the unseen prefix-query pairs.

6.3.5 Features

Table 6.6 summarizes the features used in the H, D and R models. Specifically, for the H model, we adopt these features for the following reasons. *TypingSpeed*: an expert user is less likely to use QAC than an inexperienced user. *CurrPosition*: a user tend to examine the queries at the end of typing. *IsWordBoundary*: a user is more likely to lookup queries at word boundaries. *NbSuggQueries*: it's more likely to be examined if the number of suggested queries is small. *ContentSim*: a user may be more likely to examine the list if all queries are coherent in content. *QueryIntent*: a user tends to skip the list more when searching for navigational queries.

The feature for D model are the positions a query candidate is ranked. The purpose of using this feature is that we want to use the D model to measure the pure vertical position bias. Note that the form of D model allows us to incorporate more complex features in the future.

For the R model, we have designed 8 features in total, reflecting diverse aspects of the relevance model. It includes the query popularity counts, which is widely used in the current search engines, the long term query history query counts, geo-location and time related query frequencies, and 4 other demographics features. Similar features are reported in [79], therefore comparing our model to that in [79] is meaningful.

6.3.6 Model Estimation via E-M Algorithm

In this section we discuss the estimation of model parameters $\theta = \{w_H, w_D, w_R\}$. A straightforward way is to take the log of Equation 6.1 and estimate θ by Maximum Likelihood. However since Equation 6.1 involves the summation of the H and D vectors, the estimation is quite complicated. Based on the form of distributions and the choice of features, we make some independent assumption of variables at different columns in order to simplify the model estimation:

$$P(H_i|H_j, i \neq j, \theta) = P(H_i|\theta) \tag{6.12}$$

$$P(D_i|D_j, i \neq j, \theta) = P(D_i|\theta) \tag{6.13}$$

$$P(C_i|H_i, D_i, H_j, D_j, i \neq j, \theta) = P(C_i|H_i, D_i, \theta) \tag{6.14}$$

This assumption breaks the interdependency between columns. And the likelihood of different columns are still related because they share common parameters. Under these assumptions, the log likelihood of observing a click given the model parameters θ is:

$$logP(C|\theta) = \sum_{i=1}^{n} log \sum_{H_i, D_i} P(C_i, H_i, D_i|\theta) \tag{6.15}$$

Model parameters $\theta = \{w_H, w_D, w_R\}$ can be estimated by maximizing Equation 6.15. However, direct estimation of the model parameters θ is still hard because of the summation inside the logarithm. Instead, we sought to maximize the lower bound of Equation 6.15:

$$\begin{aligned} logP(C|\theta) &= \sum_{i=1}^{n} log \sum_{H_i, D_i} P(C_i, H_i, D_i|\theta) \\ &\geq \sum_{i=1}^{n} \sum_{H_i, D_i} P(H_i, D_i|C_i, \theta^{old}) \cdot logP(C_i, H_i, D_i|\theta) \\ &= Q(\theta, \theta^{old}) \end{aligned} \tag{6.16}$$

After fully formulating the Q function, model parameters can be updated iteratively by the E-M algorithm. In the E step, we aim at calculating the posterior distribution:

$$\begin{aligned} &P(H_i, D_i|C_i, \theta^{old}) \\ =&\frac{P(C_i|H_i = l, D_i = j, \theta^{old}) \cdot P(H_i = l, D_i = j|\theta^{old})}{\sum_{l=0}^{1} \sum_{j=1}^{M_i} P(C_i|H_i = l, D_i = j, \theta^{old}) \cdot P(H_i = l, D_i = j|\theta^{old})} \end{aligned} \tag{6.17}$$

And in the M step, we maximize the expectation of the complete data probability in Equation 6.16 by gradient descent. Since the forms of distributions in H, D, R are all convex, the E-M algorithm is guaranteed to converge. And because each QAC session is independent, the above E-M algorithm for parameter estimation can be easily expanded to the whole set of sessions. Here we skip the detailed formulations due to the space limitation.

6.4 Experiments and Results

In this section, we conduct a series of experiments to validate our major claims on the TDCM model. Firstly, due to the difference between document retrieval and QAC, we claim that most of the existing click models are not effective in modeling the QAC. We will compare our model with the state-of-the-art click models on the relevance modeling. Besides testing on PC and iPhone 5 datasets, we also experiment on a random bucket dataset which provides an unbiased evaluation of the relevance ranking. Secondly, as we have mentioned, for training a QAC relevance model, previous studies either use the last column as training data which might not have enough training cases, or use all columns as training examples [79] which might introduce too much noise. We will demonstrate that our model can be leveraged to improve existing learning-based methods by providing more appropriate training examples. Further, we will investigate the vertical position bias via our model on a side-by-side comparison of such bias on PC and iPhone 5 platforms. Finally we discuss some interesting insights about the user behavior on both platforms.

6.4.1 Datasets and Metrics

We use the same datasets for evaluation as in user behavior analysis (Section 6.2.2). The whole dataset is divided evenly into a training set and a test set. See Table 6.8 for detailed statistics.

As reported in previous work [10, 79], manual labeling of relevance for QAC is very

Table 6.8: Training and Test Sets

Platform	Dataset	# Sessions	Ave Prefixes
PC	All	125,392	11.80
	Training	62,534	11.77
	Test	62,858	11.83
iPhone 5	All	31,227	9.43
	Training	15,394	9.46
	Test	15,833	9.40
Random Bucket	Test	21,154	16.15

difficult since it's hard to find consensus between individuals on the preferred queries given the same prefix. Instead a common practice of evaluating the QAC performance is to measure the prediction accuracy of the users' clicked queries [10]. In this work we adopt this evaluation strategy. In addition, because the user clicks are a biased estimate of relevance, we also set up a random bucket to collect clicks from a small portion of traffic during the same period. In this random bucket, for every prefix top-10 ranked queries are randomly shuffled and displayed to the users. By doing so, it reduces the vertical position bias and the collected user clicks can be treated as the unbiased estimation of relevance of queries [53].

For evaluation metrics, we employ the Mean Reciprocal Rank (MRR) as the main measurement of relevance. It is the standard practice in measuring QAC performance [10, 79]. Specifically, for a QAC session, the list of candidates are generated from a commercial search engine and recorded in our dataset. Columns (suggested queries associated with a prefix) in which the final submitted query does not appear among the top-10 candidates are removed from the analysis. Then we compute the average MRR across all remaining columns. In addition, we also report the MRR of the last column since this is the column where real user click happens. Paired t-test is adopted for validating the statistical significance with *p-value* cutoff 0.05.

For baselines, *MostPopularCompletion* (MPC) is used as a baseline. Despite its simplicity, MPC has been reported as a very competitive baseline and widely used as a main feature in the QAC engines [10, 79]. We also compare our approach to three state-of-the-art click models, including User Browsing Model (UBM)[31], Dynamic Bayesian Network model (DBN) [22] and Bayesian Sequential State model (BSS) [92]. The UBM and DBN rely on the counting of prefix-query pairs thus they are unable to predict unseen prefix-query pairs. On the other hand BSS is a content-aware method and it can predict unseen prefix-query pairs. We adopt the source code of these approaches from [92]. Since our model makes use of all columns of data, to make a fair comparison, we train these click models on the last column as well as all columns. All baselines and their description are listed in Table 6.9.

Table 6.9: Baselines for Comparison

Model	Description	Training Data
MPC	Most Popular Completion	no training is needed
UBM-last	User Browsing Model	last column
UBM-all	User Browsing Model	all columns
DBN-last	Dynamic Bayesian Network model	last column
DBN-all	Dynamic Bayesian Network model	all columns
BSS-last	Bayesian Sequential State model	last column
BSS-all	Bayesian Sequential State model	all columns
TDCM	Our model	all columns

6.4.2 Evaluating the Relevance Model

Normal Bucket test. We first investigate whether our click model has advantage over existing click models on improving the QAC relevance ranking. For this purpose, we compare our model to the MPC baseline and other click models on normal buckets from both PC and iPhone 5 platforms. The results are summarized in Table 6.10. Firstly, counting-based click models (UBM and DBN) are generally not effective for modeling the relevance in QAC. For example, the UBM-last and DBN-last methods under-perform the MPC baseline on both PC and iPhone 5 datasets. Although UBM-all DBN-all perform a little better than UBM-last and DBN-last, training on all columns of data still doesn't give an edge to these methods over MPC baseline. This is not surprising because UBM and DBN rely on counting the prefix-query pairs. However, in the dynamic environment of QAC, the percentage of unseen prefix-query pairs is large (67.4% in PC and 60.5% in iPhone 5 as observed in our datasets), which is very different from that in document retrieval. Presumably, training on all columns will add more seen prefix-query pairs, which leads to the improved MRR. However, since these models are not designed to model the whole QAC process, they are unable to capture useful signals in all columns and thus unable to improve the performance much.

In addition, the BSS model performs better then UBM and DBN. For example, when trained on last column, it achieves 0.515 on MRR@All and 0.543 on MRR@Last on the PC dataset, indicating its effectiveness of capturing relevance by content-based modeling of relevance. One advantage of content-based modeling is that it's able to interpolate the relevance model to unseen prefix-query pairs, which is critical in QAC. But training on all columns doesn't boost its performance, suggesting the importance of modeling the user behavior in the whole QAC process so as to filter out the noise. We also note that BSS is not consistent on these two platforms: for example, it doesn't work well in the iPhone 5 dataset (0.510 on MRR@All on 0.537 on MRR@Last by BSS-last).

On the other hand, our TDCM model achieves significant better results on both platforms. For example it achieves 0.525 on MRR@All and 0.573 on MRR@Last on the PC dataset. And on iPhone 5 dataset, it gets 0.580 on MRR@All and 0.668 on MRR@Last. All of these results are statistically significant better than MPC. Compared to UBM and DBN, our model overcomes their limitation by adopting the content-aware relevance model. And compared to BSS, our model takes advantage of all columns of data by properly modeling the user behavior in the whole QAC process, leading to much better and stable results on both platforms.

Random Bucket test. Using normal traffic to evaluate the relevance model might be biased because a model could be optimized by chasing the clicks rather than the intrinsic relevance/utility. To make an unbiased evaluation we also test all the methods on a random bucket dataset containing 21,154 QAC sessions. We summarize the results in Table 6.11. Overall the MPC baseline performs worse than that in normal bucket. It's expected because

Table 6.10: Click Models Comparison on Normal Bucket

	PC MRR@All	PC MRR@Last	iPhone 5 MRR@All	iPhone 5 MRR@Last
MPC	0.447	0.534	0.542	0.646
UBM-last	0.416	0.449	0.409	0.432
UBM-all	0.445	0.452	0.431	0.432
DBN-last	0.418	0.437	0.405	0.427
DBN-all	0.454	0.442	0.435	0.423
BSS-last	**0.515**‡	0.543	0.510	0.537
BSS-all	0.495	0.523	0.480	0.479
TDCM	**0.525**‡	**0.573**‡	**0.580**‡	**0.668**‡

Note: ‡ indicates *p-value*¡0.05 compared to MPC

as the position bias is reduced, users have more chance to click on queries that are not the most popular. Similarly, UBM and DBN models fail to outperform MPC baseline and the BSS model achieves reasonable results compared to MPC. Again, our model achieves the best results on both MRR@All (0.493) and MRR@Last (0.508) metrics, which is statistical significant compared to MPC. These results are consistent with that observed in normal traffic, confirming the superiority of our TDCM model on relevance modeling.

Table 6.11: Click Models Comparison on Random Bucket

	MRR@All	MRR@Last
MPC	0.429	0.485
UBM-last	0.381	0.402
UBM-all	0.397	0.393
DBN-last	0.373	0.391
DBN-all	0.388	0.391
BSS-last	**0.471**‡	0.488
BSS-all	0.460	0.469
TDCM	**0.493**‡	**0.508**‡

Note: ‡ indicates *p-value*¡0.05 compared to MPC

6.4.3 Relevance Model Performance by Query Length

In order to investigate whether our model is robust in all sectors of queries, we break the relevance results into 5 groups according to their clicked query length. MPC results are also shown for comparison. Results in Figure 6.4 reveal that on both PC and iPhone 5 datasets, our model's performance decreases gradually as the submitted query length increases. There is no abrupt drop of performance in a sector of queries, indicating that our model is robust to queries with different length. In addition, the MPC baseline has similar trend as our model. This common trend suggests the importance of the query popularity count because the shorter queries generally have higher popularity counts. In the random bucket, the MRR of our model drops when the query length increase, and starts to increase again when the query length becomes larger. This trend suggests that in random bucket, MPC feature becomes less important than in normal bucket; thus longer queries will still have good MRR even though their MPC scores are smaller.

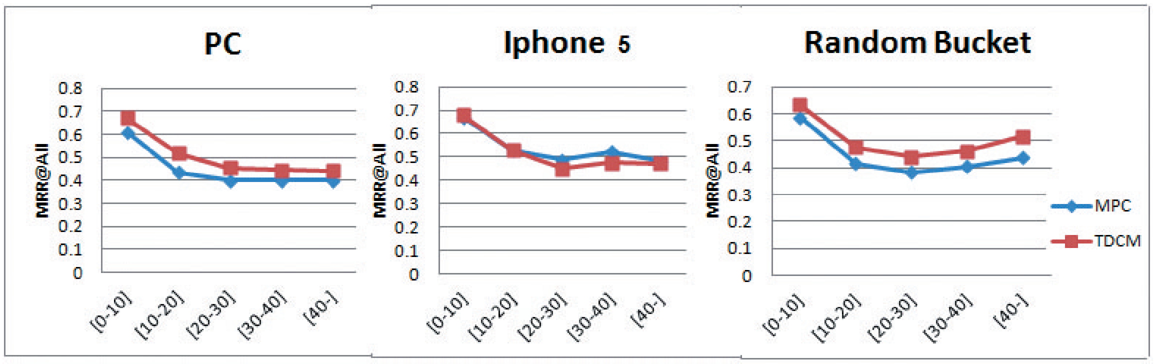


Figure 6.4: MRR Evaluation by Query Length. All sessions are aligned to groups based on the submitted query length. Performance is measured by MRR@All

6.4.4 Validating the H Model: Automatic Labeling by TDCM

Another advantage of our model over existing click models is that we can utilize the learned user behavior in QAC to enhance other learning-based methods. In the pilot experiment in Section 6.2.2, we have shown that even though RankSVM is a state-of-the-art ranker, when trained by all columns, its performance doesn't even beat MPC (-2.46% on MRR@All). The reason is probably because although we are sure that a user has viewed and examined the last column, it's uncertain that she has viewed other columns; so the information in previous columns is not reliable. The noise in all columns may outweigh the useful information they bring about. So by simply training on all columns it is generally not effective. In this experiment, we test whether the user examination behavior inferred by our model can be used to improve other methods. In order to achieve this, we first run TDCM on the training dataset, obtain all $P(H)$ probabilities for each session. After that, we keep the columns with high $P(H = 1)$ (¿0.7). Finally we use these columns to train the corresponding models again. The labeling criteria is simple: if the candidate equals the final submitted query, we label it as positive, and other candidates are all labeled as negative. Results of this experiment are shown in Figure 6.5. It is indicated that using this simple automatic labeling strategy, RankSVM achieves better MRR@All across three datasets. For example, on PC dataset RankSVM achieves 0.523 on MRR@All, compared to 0.514 by training on last column only. Similar improvements are observed in iPhone 5 and Random Bucket. These results suggest that the user behavior information inferred by our model can be applied to other models. Particularly, the information whether a query has been examined is very useful for improving other models' performance.

6.4.5 Validating the D Model

Here we seek to evaluate the accuracy of the D model, that is the vertical examination distribution. Intuitively the probability of examining a position should be correlated to the clickthrough rate. In our feature instantiation, all features for the D model are vertical positions. So it is possible to draw the distribution of D according to the feature weights w_D, which corresponds to the probability of examining a particular position. In this experiment we run the TDCM model on both PC and iPhone 5 dataset, and draw the distribution along with the real click through rate (CTR) in Figure 6.6. From Figure 6.6 we see that the shape

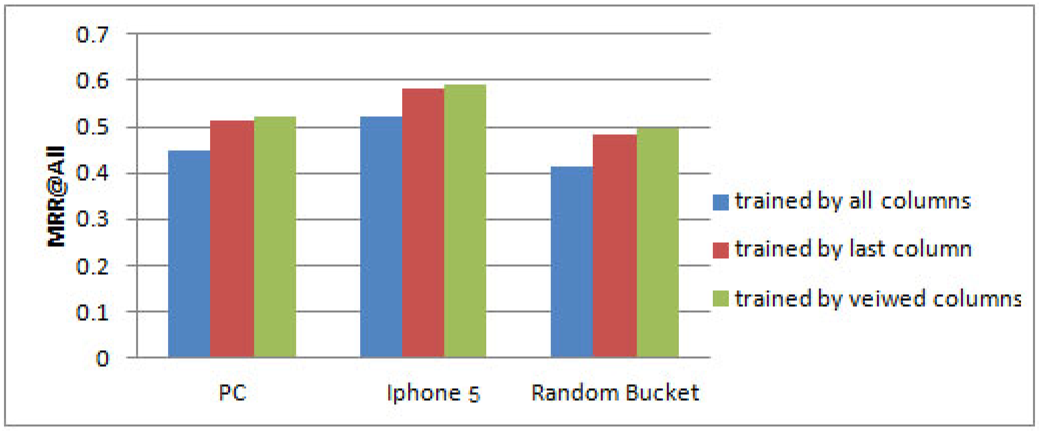


Figure 6.5: Model Training on Selected Columns. Viewed columns: columns whose $P(H = 1) > 0.7$. Performance is measured by MRR@All

of the D model distribution is similar to the real CTR. Both distributions are very steep, attracting more probabilities in top positions. Our estimation of the D model is a little flatter than the real CTR. For example, on PC platform, at the top 1 position our model estimates the examination probability to be 0.397, while the real CTR is 0.500. And in the 2nd position we predict more probability (0.314) than the real CTR (0.254). Compared to PC platform, in iPhone 5 platform both the real CTR and our estimated examination distribution are flatter. This suggests an interesting conclusion that in mobile devices people tend to examine deeper down the suggestion list.

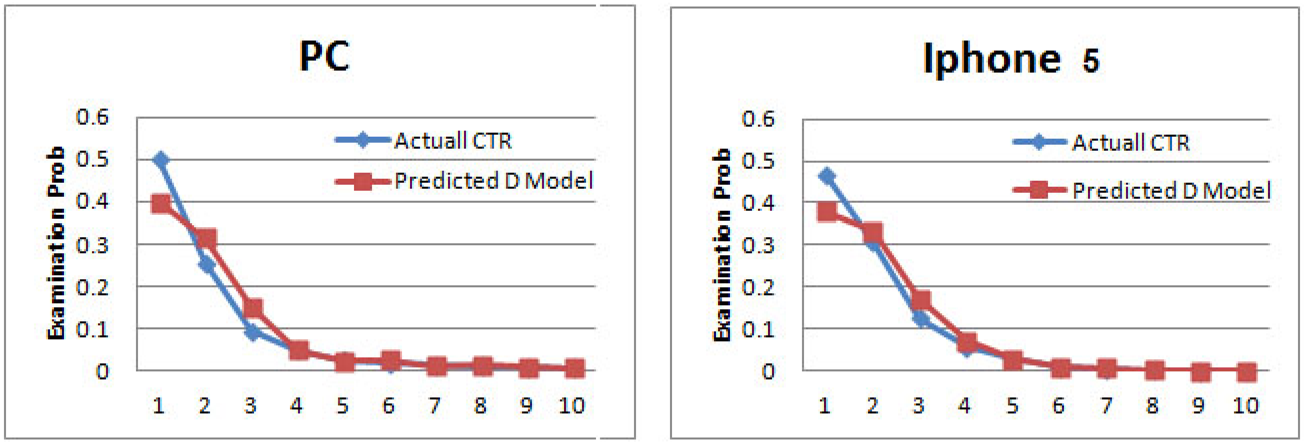


Figure 6.6: The D distribution VS real CTR. Positions

6.4.6 Understanding User Behavior via Feature Weights

An additional benefit of our proposed model is that the learned feature weights reveal the influence of different factors on users' behavior in the QAC process, which is not available in most of the existing click models. To explore this, we list a subset of learned weights in Table 6.12. Although the absolute values of these weights don't reflect exactly the importance of features because scales of the features are different, we can still tell their relative importance by comparing them on PC and iPhone 5 side by side.

Firstly, in the H model related features, *TypingSpeed* is the most important feature both on PC and iPhone 5. *TypingSpeed* is reversely proportional to $P(H) = 1$. Interestingly, the absolute weight of *TypingSpeed* is larger in PC than in iPhone 5, suggesting that people tend to skip more when using QAC in PC because they type faster in PC. Another important feature is *IsWordBoundary*. Intuitively it makes sense since people tend to stop and look for query completions when they are typing at word boundaries. The *QueryIntent* feature also plays a role, indicating that people tend to skip more when looking for navigational queries; while they need more help from the QAC engine when they are seeking information and uncertain how to formulate the queries.

Secondly, the features of the D model is examination probabilities. As mentioned in the previous experiment, these probabilities are higher at top positions. In PC, the estimated examination probabilities concentrate more on the 1st position. On iPhone 5 the 2nd and 3rd positions receive more examination probabilities than PC. This suggests that in mobile devices people will look deeper down the suggestion list.

Thirdly, for the R model, people pay more attention on long query history in iPhone 5 than in PC. This might be because typing is harder in mobile devices; so people rely on the QAC engine to store and retrieve their past queries. Another interesting finding is that geo-location related signals and time-sense signals are both important, revealing that people prefer location-relevant and fresh queries.

Table 6.12: Feature Weights Learned by TDCM

	w_H	TypingSpeed	IsWordBoundary	CurrPosition	QueryIntent
		-0.86	0.55	0.32	-0.20
	w_D	Position@1	Position@2	Position@3	
PC		0.397	0.314	0.152	
	w_R	MPC	QryHistFreq	GeoSense	TimeSense
		1.790	0.973	0.962	1.115
	w_H	TypingSpeed	IsWordBoundary	CurrPosition	QueryIntent
		-0.57	0.50	0.20	-0.28
	w_D	Position@1	Position@2	Position@3	
iPhone 5		0.3782	0.334	0.171	
	w_R	MPC	QryHistFreq	GeoSense	TimeSense
		4.139	3.918	0.947	1.595

6.5 Conclusion and Future Work

The QAC problem is under-explored because of the lack of suitable query logs. In this work we have collected a large set of QAC sessions with fine-grain user interaction information, which enables us to analyze and model the user behavior in QAC. Based on two key observations, namely the horizontal skipping bias and vertical examination bias, we have proposed a novel Two-Dimensional Click Model for modeling the QAC process. Extensive experiments on our datasets demonstrated that our TDCM model can accurately explain the user behavior in QAC. The resulting relevance model significantly outperforms all existing click models. In addition, user behavior information learned by our model can be incorporated into other learning-based methods to further improve their performance. Using our model, we also discover some interesting user behavior on PC and mobile devices.

As the first click model for QAC, our TDCM model could be extended in several ways in the future. For example, the independent assumption between different columns can be relaxed to capture multi-column interdependency. In addition, more complex models can replace the D model to better explain the vertical position bias.

Chapter 7

Summary and Future Directions

Understanding a web search query is a significant task toward further improving the quality of current web search engines. However accurate query understanding is a non-trivial task. This thesis is a systematic study of multi-level query understanding and representation. I proposed that the grand task of query understanding can be broken down to multiple, logically dependent, levels of query understanding: (1) Query Alteration; (2) Latent Query Linguistic Signal Discovery; (3) Semantic Annotation of Queries; (4) Dynamic Understanding and Representation of Queries. The logical dependency of these levels of query understanding is evident. For example, query spelling correction, as an important form of query alteration, affects the quality of all other levels of query understanding. And the latent query linguistic signal discovery, such as detecting the query segmentation structure, is a prerequisite of accurate query semantic annotation. In this thesis I addressed the most important research questions in each level of query understanding and representation. The contributions of this thesis are summarized as follows:

Query Spelling Correction by a Generalized Hidden Markov Model. Query spelling correction is a crucial type of query alteration in modern search engines. Existing methods in the literature for search query spelling correction have two major drawbacks. First, they are unable to handle certain important types of spelling errors, such as concatenation and splitting. Second, they cannot efficiently evaluate all the candidate corrections due to the complex form of their scoring functions, and a heuristic filtering step must be applied to select a working set of top-K most promising candidates for final scoring, leading to non-optimal predictions. We addressed both limitations and proposed a novel generalized

Hidden Markov Model with discriminative training that can not only handle all the major types of spelling errors, including splitting and concatenation errors, in a single unified framework, but also efficiently evaluate all the candidate corrections to ensure the finding of a globally optimal correction. Experiments on two query spelling correction datasets demonstrate that the proposed generalized HMM is effective for correcting multiple types of spelling errors. The results also show that it significantly outperforms the current approach for generating top-K candidate corrections, making it a better first-stage filter to enable any other complex spelling correction algorithm to have access to a better working set of candidate corrections as well as to cover splitting and concatenation errors, which no existing method in academic literature can correct.

Query Segmentation using Clicktrhough. In Latent Query Linguistic Signal Discovery, we focus on the problem of Query Segmentation. Existing segmentation models either use labeled data to predict the segmentation boundaries, for which the training data is expensive to collect, or employ unsupervised strategy only based on a large text corpus, which might be inaccurate because of the lack of relevant information. In this work, I proposed a probabilistic model to exploit click-through data for query segmentation. I further proposed an integrated language model based on the standard bigram language model to utilize the probabilistic structure obtained through query segmentation. The resulting language model with query segmentation outperforms BM25, standard unigram and bigram language models.

Entity Attribute Synonyms Mining. Discovering such alternative surface forms of entities and attributes is crucial for improving query semantic annotation. In this work we proposed a novel compact clustering framework to jointly identify synonyms for a set of entities. The framework can integrate signals from multiple information sources into a similarity function between attribute values. And the weights of these signals are optimized in an unsupervised manner.

Modeling Query Auto-completion by a Two-dimensional Click Model. Query auto-completion is a typical type of dynamic query understanding in which the users want to be assisted by giving dynamically changing short prefixes in real time. In this work, for the first time we collected a high-resolution QAC query log that records every keystroke in a QAC session. Based on this data, we discovered two user behaviors, namely the horizontal skipping bias and vertical position bias which are crucial for relevance prediction in QAC. Particularly the horizontal skipping bias was introduced for the first time; and it's unique to the query auto-completion process. In order to better explain them, we proposed a novel two-dimensional click model for modeling the QAC process with emphasis on these behaviors. Extensive experiments show that the resulting relevance model significantly improves the relevance ranking in QAC than most of the existing click models.

Query understanding is a broad research area under active investigation, and the following would be among the most interesting directions for further research:

- **Understanding and Representation of Complex Queries**. In some scenarios query understanding is difficult when the user query is complex in meaning. One scenario is the task query which is usually consisted of multiple steps in a logical or timed order. For instance, a user may issue a query "learning about depression" in order to finish a task of learning the disease of depression. In this case, information about several aspects of the disease such as causes, symptom, treatment and recent research advancement should be presented as results ordered by logical order. Likewise, in another example, a user query is one of the series queries she issued in a long-term task. In this case, knowing the whole spectrum of the user's inaction of the search engine in this long-term task can potentially improve the user's satisfaction of the returned results.

- **Query Understanding in Mobile Search**. Another emerging and interesting direction is understanding and representing user queries in mobile search. Mobile devices are gradually surpassing PCs and becoming the major platform for information search and browsing. The current search engines in mobile devices are mainly adopting the successful models from that in PCs. However, because of the distinction of mobile devices and PCs – for example the small screen size, the mobility, and the ecosystem change – the user intents might be very different between mobile devices and PCs. Thus, it is very interesting to investigate the query understanding the representation in mobile search. For example, how can we dynamically represent and recommend queries in ever changing locations and time? And how the limited screen size affects the query spelling behaviors of mobile users? *etc.*.

References

[1] http://lemurproject.org/clueweb09/.

[2] http://lemurproject.org/clueweb12/.

[3] http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx.

[4] http://research.microsoft.com/en-us/projects/spellerchallenge/.

[5] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26. ACM, 2006.

[6] Farooq Ahmad and Grzegorz Kondrak. Learning a spelling error model from search query logs. In *HLT/EMNLP*. The Association for Computational Linguistics, 2005.

[7] Ioannis Antonellis, Hector Garcia Molina, and Chi Chao Chang. Simrank++: query rewriting through link analysis of the click graph. *Proc. VLDB Endow.*, 1(1):408–421, August 2008.

[8] Mario Arias, José Manuel Cantera, Jesús Vegas, Pablo de la Fuente, Jorge Cabrero Alonso, Guido García Bernardo, César Llamas, and Álvaro Zubizarreta. Context-based personalization for mobile web search. In *PersDB*, pages 33–39, 2008.

[9] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *J. Mach. Learn. Res.*, 6:937–965, December 2005.

[10] Ziv Bar-Yossef and Naama Kraus. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web*, pages 107–116. ACM, 2011.

[11] Marco Baroni and Sabrina Bisi. Using cooccurrence statistics and the web to discover synonyms in technical language. In *In Proceedings of LREC 2004*, pages 1725–1728, 2004.

[12] Holger Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 364–371. ACM, 2006.

[13] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[14] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: A generic approach to entity resolution. Technical Report 2005-5, Stanford InfoLab, 2005.

[15] Shane Bergsma and Qin I. Wang. Learning noun phrase query segmentation. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 819–826.

[16] Matthew Berland and Eugene Charniak. Finding parts in very large corpora. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 57–64, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

[17] E. Brill and R. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, 2000.

[18] Guihong Cao, Jian-Yun Nie, and Jing Bai. Integrating word relationships into language models. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 298–305, New York, NY, USA, 2005.

[19] Guihong Cao, Stephen Robertson, and Jian-Yun Nie. Selecting query term alternations for web search by exploiting query contexts. In *Proceedings of ACL-08: HLT*, pages 148–155, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[20] Kaushik Chakrabarti, Surajit Chaudhuri, Tao Cheng, and Dong Xin. A framework for robust discovery of entity synonyms. In *In Proc. of the 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'12)*, 2012.

[21] Elaine Y. Chan, Wai-Ki Ching, Michael K. Ng, and Joshua Zhexue Huang. An optimization algorithm for clustering using weighted dissimilarity measures. *Pattern Recognition*, pages 943–952, 2004.

[22] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, pages 1–10. ACM, 2009.

[23] Qing Chen, Mu Li, and Ming Zhou. Improving query spelling correction using web search results. In *EMNLP-CoNLL*, pages 181–189. ACL, 2007.

[24] Tao Cheng, Hady Wirawan Lauw, and Stelios Paparizos. Fuzzy matching of web queries to structured data. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *ICDE*, pages 713–716. IEEE, 2010.

[25] Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[26] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 87–94. ACM, 2008.

[27] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[28] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 85–96, New York, NY, USA, 2005. ACM.

[29] Huizhong Duan and Bo-June Paul Hsu. Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, pages 117–126. ACM, 2011.

[30] Huizhong Duan, Yanen Li, ChengXiang Zhai, and Dan Roth. A discriminative model for query spelling correction with latent structural svm. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1511–1521. ACM, 2012.

[31] Georges E Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338. ACM, 2008.

[32] Jianfeng Gao, Xiaolong Li, Daniel Micol, Chris Quirk, and Xu Sun. A large scale ranker-based system for search query spelling correction. In Chu-Ren Huang and Dan Jurafsky, editors, *COLING*, pages 358–366. Tsinghua University Press, 2010.

[33] Jianfeng Gao, Jian-Yun Nie, Guangyuan Wu, and Guihong Cao. Dependence language model for information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 170–177, New York, NY, USA, 2004. ACM.

[34] Laura A Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 478–479. ACM, 2004.

[35] Jiafeng Guo, Gu Xu, Hang Li, and Xueqi 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.

[36] Matthias Hagen, Martin Potthast, Benno Stein, and Christof Braeutigam. The power of naive query segmentation. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 797–798, New York, NY, USA, 2010.

[37] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[38] Bo-June Paul Hsu and Giuseppe Ottaviano. Space-efficient data structures for top-k completion. In *Proceedings of the 22nd international conference on World Wide Web*, pages 583–594. International World Wide Web Conferences Steering Committee, 2013.

[39] Bo-June Paul Hsu and Giuseppe Ottaviano. Space-efficient data structures for top-k completion. In *Proceedings of the 22nd international conference on World Wide Web*, pages 583–594. International World Wide Web Conferences Steering Committee, 2013.

[40] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446, October 2002.

[41] Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. Efficient interactive fuzzy keyword search. In *Proceedings of the 18th international conference on World wide web*, pages 371–380. ACM, 2009.

[42] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[43] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 387–396, New York, NY, USA, 2006.

[44] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396. ACM, 2006.

[45] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 387–396, New York, NY, USA, 2006. ACM.

[46] Biing Hwang Juang and Laurence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

[47] T. Mikolajewski K. M. Risvik and P. Boros. Query segmentation for web search. In *Proceeding of the 12th international conference on World Wide Web*, WWW '03, 2003.

[48] L. Kaufman and P. Rousseeuw. *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. Fac., Univ., 1987.

[49] L. Kaufman and P.J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley, 1990.

[50] K. Kukich. Techniques for automatically correcting words in text. *ACM computing surveys*, 24(4), 1992.

[51] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, San Fransisco, 2001. Morgan Kaufmann.

[52] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. 1965.

[53] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM, 2011.

[54] Yanen Li, Anlei Dong, Hongning Wang, Hongbo Deng, ChengXiang Zhai, and Y-i Chang. A two-dimensional click model for query auto-completion. In *Proceedings of the 37th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2014.

[55] Yanen Li, Huizhong Duan, and ChengXiang Zhai. Cloudspeller: query spelling correction by using a unified hidden markov model with web-scale resources. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 561–562. ACM, 2012.

[56] Yanen Li, Huizhong Duan, and ChengXiang Zhai. A generalized hidden markov model with discriminative training for query spelling correction. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 611–620. ACM, 2012.

[57] Yanen Li, Bo-Jun Paul Hsu, ChengXiang Zhai, and Kuansan Wang. Unsupervised query segmentation using clickthrough for information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 285–294, New York, NY, USA, 2011. ACM.

[58] Yanen Li, Bo-June Paul Hsu, and ChengXiang Zhai. Unsupervised identification of synonymous query intent templates for attribute intents. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2029–2038. ACM, 2013.

[59] Yanen Li, Bo-June Paul Hsu, ChengXiang Zhai, and Kuansan Wang. Mining entity attribute synonyms via compact clustering. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 867–872. ACM, 2013.

[60] Dekang Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*, ACL '98, pages 768–774, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[61] Dekang Lin, Shaojun Zhao, Lijuan Qin, and Ming Zhou. Identifying synonyms among distributionally similar words. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 1492–1493, 2003.

[62] Chao Liu, Fan Guo, and Christos Faloutsos. Bayesian browsing model: Exact inference of document relevance from petabyte-scale data. *ACM Trans. Knowl. Discov. Data*, 4(4):19:1–19:26, October 2010.

[63] Gord Luec. A data-driven approach for correcting search quaries. In *Spelling Alteration for Web Search Workshop*, July 2011.

[64] Scott Mcdonald and Michael Ramscar. Testing the distributional hypothesis: The influence of context on judgements of semantic similarity. In *In Proceedings of the 23rd Annual Conference of the Cognitive Science Society*, pages 611–6, 2001.

[65] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 472–479, New York, NY, USA, 2005.

[66] Shachar Mirkin, Ido Dagan, and Maayan Geffet. Integrating pattern-based and distributional similarity methods for lexical entailment acquisition. In *Proceedings of the COLING/ACL on Main conference poster sessions*, COLING-ACL '06, pages 579–586, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[67] G. Mishne and M. de Rijke. Boosting Web Retrieval through Query Operations. In *In Proc. 27th European Conference on Information Retrieval (ECIR '05)*, pages 502–516, 2005.

[68] Mandar Mitra, Amit Singhal, and Chris Buckley. Improving automatic query expansion. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 206–214. ACM, 1998.

[69] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[70] Patrick Pantel and Dekang Lin. Document clustering with committees. In *In Proc. of SIGIR02*, pages 199–206. ACM Press, 2002.

[71] Fuchun Peng and Dale Schuurmans. Self-supervised chinese word segmentation. In *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, IDA '01, pages 238–247, London, UK, 2001. Springer-Verlag.

[72] Yonggang Qiu and Hans-Peter Frei. Concept based query expansion. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 160–169. ACM, 1993.

[73] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[74] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248. ACM, 2005.

[75] Matthew Richardson. Predicting clicks: Estimating the click-through rate for new ads. In *In Proceedings of the 16th International World Wide Web Conference (WWW-07*, pages 521–530. ACM Press, 2007.

[76] S. E. Robertson. *The probability ranking principle in IR*, pages 281–286. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[77] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[78] Mehran Sahami and Timothy D Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, pages 377–386. ACM, 2006.

[79] Milad Shokouhi. Learning to personalize query auto-completion. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 103–112. ACM, 2013.

[80] Luo Si, Rong Jin, Steven C. H. Hoi, and Michael R. Lyu. Collaborative image retrieval via regularized metric learning, 2006.

[81] Munirathnam Srikanth and Rohini Srihari. Biterm language models for document retrieval. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 425–426, New York, NY, USA, 2002.

[82] Xu Sun, Jianfeng Gao, Daniel Micol, and Chris 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.

[83] Bin Tan, Yuanhua Lv, and ChengXiang Zhai. Mining long-lasting exploratory user interests from search history. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1477–1481. ACM, 2012.

[84] Bin Tan and Fuchun Peng. Unsupervised query segmentation using generative language models and wikipedia. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 347–356, New York, NY, USA, 2008. ACM.

[85] Bin Tan, Xuehua Shen, and ChengXiang Zhai. Mining long-term search history to improve search accuracy. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 718–723. ACM, 2006.

[86] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *SIGIR '07*, pages 295–302, New York, NY, USA, 2007.

[87] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 585–593, New York, NY, USA, 2006. ACM.

[88] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, pages 491–502, London, UK, UK, 2001. Springer-Verlag.

[89] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics, 1996.

[90] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[91] Chi Wang, Kaushik Chakrabarti, Tao Cheng, and Surajit Chaudhuri. Targeted disambiguation of ad-hoc, homogeneous sets of named entities. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 719–728, New York, NY, USA, 2012. ACM.

[92] Hongning Wang, ChengXiang Zhai, Anlei Dong, and Yi Chang. Content-aware click modeling. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1365–1376. International World Wide Web Conferences Steering Committee, 2013.

[93] Kuansan Wang, Christopher Thrasher, and Bo-June Paul Hsu. Web scale nlp: a case study on url word breaking. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 357–366, New York, NY, USA, 2011. ACM.

[94] Kuansan Wang, Christopher Thrasher, Evelyne Viegas, Xiaolong Li, and Bo-june (Paul) Hsu. An overview of microsoft web n-gram corpus and applications. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, HLT-DEMO '10, pages 45–48, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[95] Xuanhui Wang and ChengXiang Zhai. Mining term association patterns from search logs for effective query reformulation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 479–488. ACM, 2008.

[96] Ryen W White and Gary Marchionini. Examining the effectiveness of real-time query expansion. *Information Processing & Management*, 43(3):685–704, 2007.

[97] Casey Whitelaw, Ben Hutchinson, Grace Chung, and Ged Ellis. Using the web for language independent spellchecking and autocorrection. In *EMNLP*, pages 890–899. ACL, 2009.

[98] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, volume 15, pages 505–512, 2002.

[99] Jinxi Xu and W Bruce 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*, pages 4–11. ACM, 1996.

[100] Xiaoxin Yin and Sarthak Shah. Building taxonomy of web search intents for name entity queries. In *Proceedings of the 19th international conference on World wide web*, pages 1001–1010. ACM, 2010.

[101] Xiaohui Yu and Huxia Shi. Query segmentation using conditional random fields. In *Proceedings of the First International Workshop on Keyword Search on Structured Data*, KEYS '09, pages 21–26, New York, NY, USA, 2009. ACM.

[102] Chengxiang Zhai. Fast statistical parsing of noun phrases for document indexing. In *Proceedings of the fifth conference on Applied natural language processing*, pages 312–319, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[103] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 334–342, New York, NY, USA, 2001. ACM.

[104] Yuchen Zhang, Weizhu Chen, Dong Wang, and Qiang Yang. User-click modeling for understanding and predicting search-behavior. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1388–1396. ACM, 2011.

[105] Zeyuan Allen Zhu, Weizhu Chen, Tom Minka, Chenguang Zhu, and Zheng Chen. A novel click model and its applications to online advertising. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 321–330, New York, NY, USA, 2010. ACM.