ABSTRACT

Title of dissertation: Temporal Context Modeling for Text Streams

Jinfeng Rao
Doctor of Philosophy, 2018

Dissertation directed by: Professor Jimmy Lin
Department of Computer Science

There is increasing recognition that time plays an essential role in many information seeking tasks. This dissertation explores temporal models on evolving streams of text and the role that such models play in improving information access. I consider two cases: a stream of social media posts by many users for tweet search and a stream of queries by an individual user for voice search. My work explores the relationship between temporal models and context models: for tweet search, the evolution of an event serves as the context of clustering relevant tweets; for voice search, the user's history of queries provides the context for helping understand her true information need.

First, I tackle the tweet search problem by modeling the temporal contexts of the underlying collection. The intuition is that an information need in Twitter usually correlates with a breaking news event, thus tweets posted during that event are more likely to be relevant. I explore techniques to model two different types of temporal signals: *pseudo trend* and *query trend*. The pseudo trend is estimated through the distribution of timestamps from an initial list of retrieved documents given a query, which I model through continuous hidden Markov approach as well as neural network-based methods for relevance ranking and sequence modeling. As an alternative, the query trend, is directly estimated from the temporal statistics of query terms, obviating the need for an initial retrieval. I propose two different approaches to exploit query trends: a linear feature-based ranking model and a regression-based model that recover the distribution of relevant documents directly from query trends. Extensive experiments on standard Twitter collections demonstrate the superior effectivenesses of my proposed techniques.

Second, I introduce the novel problem of voice search on an entertainment platform, where users interact with a voice-enabled remote controller through voice requests to search for TV programs. Such queries range from specific program navigation (i.e., watch a movie) to requests with vague intents and even queries that have nothing to do with watching TV. I present successively richer neural network architectures to tackle this challenge based on two key insights: The first is that session context can be exploited to disambiguate queries and recover from ASR errors, which I operationalize with hierarchical recurrent neural networks. The second insight is that query understanding requires evidence integration across multiple related tasks, which I identify as program prediction, intent classification, and query tagging. I present a novel multi-task neural architecture that jointly learns to accomplish all three tasks. The first model, already deployed in production, serves millions of queries daily with an improved customer experience. The multi-task learning model is evaluated on carefully-controlled laboratory experiments, which demonstrates further gains in effectiveness and increased system capabilities. This work now serves as the core technology in Comcast Xfinity X1 entertainment platform, which won an Emmy award in 2017 for the technical contribution in advancing television technologies.

This dissertation presents families of techniques for modeling temporal information as contexts to assist applications with streaming inputs, such as tweet search and voice search. My models not only establish the state-of-the-art effectivenesses on many related tasks, but also reveal insights of how various temporal patterns could impact real information-seeking processes.

Temporal Context Modeling for Text Streams

by

Jinfeng Rao

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Dedication

to my parents and my family...

Acknowledgments

Taking a look back to the past five years of my Ph.D., it has been a unforgettable journey in my life which I will cherish forever. It marks the end of my formal education, also the beginning of my career in industry. At this moment, my feeling is complicated: I am excited that I eventually conquer all the challenges and become a qualified independent researcher; meanwhile, those hard times in the beginning years – loneliness, disappointment and self-suspicion – always remind me to be strong and grateful. There are so many people who kindly offered their warm-hearted help throughout my PhD, and I would like to express my gratitude briefly.

First and foremost, I'd like to thank my advisor, Jimmy Lin, without whom this dissertation will never be possible. Jimmy is not only a honorable advisor to me, but also a trustworthy friend. He is always there to offer his advice at anytime I have questions about my research or career. In my early stage, Jimmy taught me hand by hand on how to start with interesting research questions, investigate potential solutions and how to write high-standard papers. These have been crucial training to prepare me as an independent researcher from an undergraduate who has little research experience before. In the later years, Jimmy offered me great freedom as well as his insightful thoughts to let me explore new research topics and excel my inspiration. Most of all, Jimmy taught me to love research with my true enthusiasm and perseverance, which I could be benefited for my entire life. Additionally, I have been tremendously benefited from Jimmy for his outstanding collaboration and mentoring ability.

I would also like to thank Ferhan Ture, who provides me the opportunity to work on voice search that have been part of this dissertation. Ferhan is a great mentor and friend during my internship in Comcast. He not only arranged everything well for me to let me focus on my research, also guided me with his great insights and breath of knowledge in machine learning.

Hua He is the person who introduces me to neural networks. He offered tremendous help during my learning of NN technologies, which is the major modeling methodology used in this dissertation. I was also greatly beneficial from many discussions with him and sincerely appreciate for his encouragements during those challenging times.

I also want to thank Prof. Hanan Samet for his guidance on my first paper when I took his course on geographic information system. His kindness helped me establish my confidence in conducting research independently.

I would also like to thank Amol Deshpande for serving on my proposal committee, and also Marine Carpuat, Jordan Boyd-Graber, Alan Sussman and John Dickerson for agreeing to be my dissertation committee members. I appreciate their invaluable time on reviewing the manuscript and many feedbacks. Specially, I want to thank Amol for sending me the UMD offer – without this, I might not be able to join UMD and spend such wonderful five years.

I am also grateful to many professors I have learnt a lot from their courses: Computational Linguistics by Philips Resnik, Algorithm by Aravind Srinivasan, Optimization by Tom Goldstein and Distributed System by Peter Keleher.

Without the support from my undergrad advisors, Jianwei Yin and Nagiza Samatova, this dissertation would hardly come true. I would like to thank for their trusts and support in the early years of my career.

The past five years would never be such fun without my lovely friends. First of all, I want to thank all my CLIP colleagues for the wonderful collaboration and daily discussions: Xing, Yulu, He, Weiwei, Thang, Jiaul, Chen, Shi and Fenfei. Also I'd like to thank my Comcast friends for those warm memories: Raul, Hongcheng, Shahin, Hasan, Oliver, Jan, Robert, Anupam, Long, Bing, Shuying and Fa. My friends outside the lab deserve my special thanks. Many thanks to Xingfeng, Zebao, Ruichi, Zichao, Yuhao, Joy, Yuwei, Jing, Yao, Xintong, Zhiyuan, Zhe, Manjie, Victor, Peng, Haotian, Rongjian, Zhengzheng, Haoyu, Zikai, Xi, Qingyi, Yingqi, Chunfeng, Fan, Ruofei, Xiyang, Hao Zhou, Hao Li, Hui, Chenxi, Shangfu, Ang, Zhenpeng, Zheng, Xi, Hong, Chang, Teng. I also want to thank my old friends who provided moral support despite the physical distance: Tao, Kaihua, Jiawei, Wen, Yu, Yabing, Chao, Qiurong, Yiwen, Siying and Zhejing.

Last but not least, I want to thank is my dear girl friend, Wanyi. She always has an idea to make me feel happy and peace when I was depressed. Reminding those warm memories with her never ceases to melt my heart. I appreciate all her accompany in the past and future.

I owe my deepest appreciation to my family who always accompany with me and support me without any hesitation. I left my parents for education at the age of ten. Although there have been some harsh times mixed with homesick and pains, those experiences eventually become an invaluable treasure that taught me to be independent and confident. I appreciate my parents for all their sacrifice and support on my education and pursuit of my dream. I also want to thank my dear sister, Xinqing, who brought me so much fun in the past years. This dissertation is dedicated to them with my utmost respect and gratitude.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In the recent decades, modern search engines have achieved great successes and provided highly-efficient ways for users to find their intended information from a huge collection of information resources. A prominent example is Google Search, which processes over 100 billion searches from 1.2 billion users across the world in each month.[1] The core technology behind search engines is *information retrieval*, an academic field formally defined by Manning et al. [3]:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

The input to an information retrieval (IR) system is a query issued from a user reflecting her information need, and the output will be a list of information resources ordered by their degrees of relevance to the query. In this dissertation, the query is assumed as a short text string either typed through a keyboard or transcribed from a voice request, and the underlying collection is composed of documents which are written in natural languages. A fundamental challenge in IR is how to match relevant documents to queries in an efficient manner. A classical approach is to

[1]`http://goo.gl/ebSvJE`

represent the query as a bag of words with each word weighted by its frequency, then calculate the matching score with respect to the document for each individual word, and finally aggregate all scores as the degree of relevance between the query and document. Such a procedure has been formulated in a variety of ways, including the vector space model with TF-IDF weights, probabilistic model like BM25 [4] and language models like the query likelihood model [5].

Being coupled with inverted lists to accelerate efficiency, traditional approaches represent reasonable solutions for providing both effective and efficient information access. However, their relevance scores are purely computed from exact term matches, which means they can not handle the vocabulary mismatch issue and capture semantic-level similarities. In many cases, descriptions of a same event have different vocabulary usages in queries and documents. For example, given a query “Obama’s trip to China”, an exact-term match approach is not likely to find the relevant document “U.S. president stays in Beijing for a one-week state visit”. To this end, methods like query expansion [6], pseudo-relevance feedback [7], latent semantic indexing [8], or the recent neural network approaches [9–11] based on word2vec [12] have attracted broad attention to solve the semantic matching problem. Their key insight is to capture word similarities through co-occurrences – two words appear frequently between each other are more likely to be relevant, which is ultimately exploited to overcome the vocabulary mismatch challenge.

In addition to modeling textual similarity from lexical and semantic views, time is another important dimension in many search tasks. Indeed, the temporal dynamics and its impact on various components of information retrieval systems, such as temporal indexing [13], temporal query analysis [14,15], time-aware retrieval and ranking [16,17], have received much attention in the last decade, and such study has been reframed as temporal information retrieval (*temporal IR*).

The main focus of temporal IR divides into two categories: 1) changes in collection contents and structures; 2) changes in user behaviors. First, the contents and structures of the Web constantly change over time, e.g., documents are created, deleted, or updated continuously. Many researchers and institutions have recognized the need to build infrastructure to preserve part of the Web [18]. For example, the well-known Internet Archive project has collected over 456 billion web pages from 1996 to 2015. Dai and Davison et al. [19] also point out the link structures of the Web evolves. Such evolutions affect not only the basic IR components like crawling and indexing, also the computation of graph-based authority measures for document ranking.

Meanwhile, how users search Web content changes over time. On one hand, search traffic looking for particular events varies over time and might exhibit certain temporal patterns, like spikes, periodicity, seasonality, and trends. The occurrence of a breaking news event (i.e., earthquakes, murders, the results of president election) would attract substantial amount of attention and user searches in a short period of time, and such popularity would gradually decrease as the news event "dies down". Also identifying the periodicity pattern of certain events (i.e., annual April Fools' day) would be beneficial for predicting when and how users would search for them. On the other hand, the search process can not be viewed as a static process. Users interact with the search system in an incremental manner – they will continue to rewrite and issue their queries as long as they are not satisfactory with returned search results yet. Such kinds of interaction behaviors usually happen in seconds or minutes, which requires a rapid and interactive modeling approach to gradually refine users' real intents.

1.1 Modeling Temporal Contexts for Tweet Search

In the literature on temporal IR, there has been much work on modeling the temporal characteristics of document collections and exploiting temporal features to improve document ranking. Recently, we have witnessed a resurgence of interest in temporal ranking models due to the ubiquity of real-time social media streams [15, 16,20–24]. Social media platforms encourage users to comment, share, and broadcast their opinions about certain breaking events in a real-time manner, which creates a large dynamic collection with strong temporalities. This dissertation selects Twitter data as a representative example for study purposes.

1.1.1 Pseudo Trend Modeling

The tweet search scenario is formally defined in the recent Microblog tracks at the Text Retrieval Conferences (TRECs) [25]: at time t, a user expresses an information need in the form of a query q. The system's task is to return topically-relevant documents (tweets) posted before the query time. Due to the time-sensitive properties of queries in Twitter (which usually corresponds to some real-world events), modeling the temporal patterns of collection content changes will provide additional

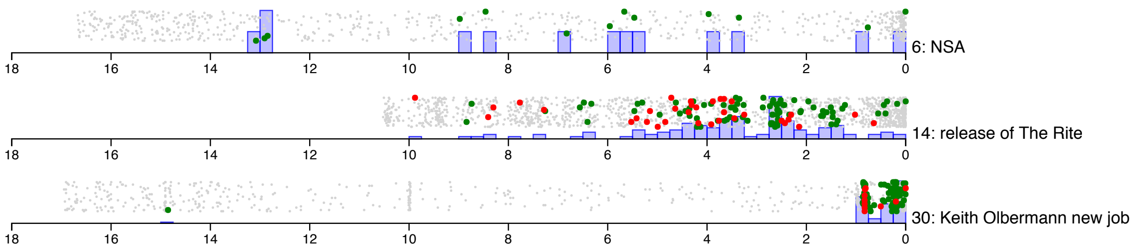


Figure 1.1: Temporal distribution of relevant (green) and highly-relevant (red) tweets for three queries from the TREC 2011 Microblog track.

relevance signals to the content-based search algorithms.

Efron et al. [16] illustrate this intuition with three example queries from the TREC 2011 Microblog Track in Figure 1.1. In each timeline, the query time (the time at which the query was issued) is anchored to the right edge; the x-axis shows time prior to the query time, in days. Dots show tweets that were retrieved by participating teams and evaluated by assessors (i.e., the pools): green dots are relevant, red dots are highly relevant, and gray dots are not relevant. The underlying blue bars show the distribution of relevant and highly-relevant tweets as a histogram. As we can see, relevant tweets for query 14 and 30 tend to cluster together in time, while relevant tweets for query 6 are more evenly distributed. Across all queries from the TREC test collections, we observe many timelines that exhibit a temporal clustering pattern (like query 14 and 30).

These visualizations demonstrate the temporal distribution of relevant tweets are often non-uniform, which suggests any retrieval model that doesn't take temporality into consideration is potentially missing out some important relevance information. Furthermore, the temporal patterns tend to be query-specific, suggesting that a one-size-fits-all strategy would not be able satisfy the needs for all queries.

For example, if we assume a recency prior [26] to select more recent tweets for query 30, it's unclear whether the same method will work for other queries like query 6 and 14. Modeling these temporal trends can provide us benefits on identifying relevant documents and improving ranking effectiveness of a IR system.

My initial work, presented in Chapter 3, aims to estimate the distribution of relevant documents in Figure 1.1 through the distribution of timestamps from the retrieved documents of an initial query. Since idea shares similarity to the pseudo-relevance feedback methods that also rely on the results of an initial query to refine estimates of term distributions in relevant documents, which we call *pseudo trend* methods. Inspired by the recent successes of neural networks in many natural language processing problems [27,28], I first propose an approach for temporal modeling of pseudo trends using recurrent neural networks [29]. Such models have been successfully applied to many sequence learning tasks in natural language processing where the modeling units are temporally dependent (e.g., tagging and parsing). I draw a connection between the temporal clustering of documents, where the relevance of one document may affect its neighbors, to a sequence learning task, and explore the hypothesis that recurrent neural networks provide a rich, expressive modeling framework to capture such temporal signals.

To this end, I propose an end-to-end neural framework [29] to incorporate lexical and temporal evidence. It consists of a lexical modeling component for converting query–document pairs into vector representations denoting their semantic similarities, and another temporal modeling component for capturing temporal relevance signals. Starting with a few existing neural lexical models [11, 30, 31] that achieves competitive performance on tasks like web search or textual similarity measurements, I wondered how they would fare in the context of noisy social media posts and how the pseudo trend signal could help in these cases.

In addition, I consider another user case of how pseudo trends can help us identifying more informative terms for query expansion. The idea of query expansion is to reduce the classical vocabulary mismatch issue by augmenting the initial query with terms that are more likely to appear in relevant documents. To this end, I propose a continuous hidden Markov model (cHMM) [32] to estimate the relevance distribution, through which we can identify the bursty state and select frequent terms in those states for query expansion. These refined query terms establish a better connection of the reformulated query to relevant documents, since we know relevant documents are more likely to occur in bursty states (as bursty states usually correspond to the occurrence of breaking news and events). This intuition is confirmed by effectiveness gains against traditional query expansion techniques without temporal information on standard Twitter collections, demonstrating that my cHMM technique is able to capture the relevance distributions well.

These work have been published in a full ECIR paper [33] for reproduction study, a ICTIR short paper [32] and a NeuIR paper [29].

1.1.2 Query Trend Modeling

In addition to the above pseudo trend methods, I also explore another family of techniques in Chapter 4 to estimate the distribution of relevant documents: instead

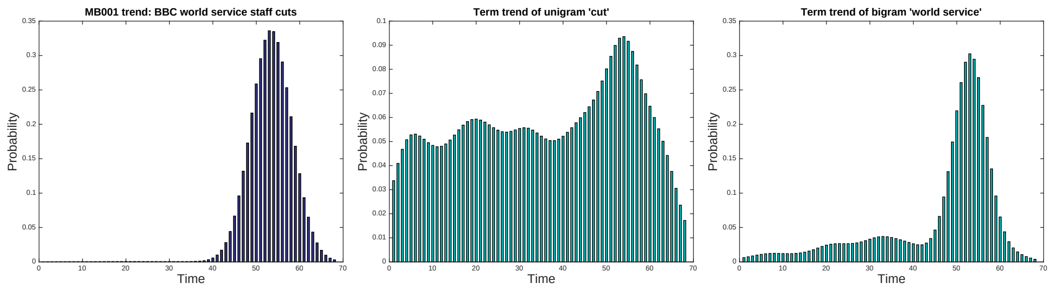


Figure 1.2: distribution of relevant documents vs. query trends on topic MB001 "BBC world service stuff cuts"

of relying on the results of an initial query, I attempt to exploit temporal signals embedded in the distribution of the query terms themselves. I call these *query trends*, which are generalizations of collection statistics of query terms (unigrams and bigrams) in the temporal dimension. In the simplest form, we might keep track of the number of occurrences of query terms across a moving window over the document collection. This intuition is illustrated in Figure 1.2 with an example: the distribution of relevant documents of topic MB001 ("BBC World Service staff cuts") from the TREC 2011 Microblog Track is shown on the left in Figure 1.2. At query time, of course, this distribution is not known—it is the target of our prediction. In the middle and right of Figure 1.2 are the term trends of the unigram *cuts* and bigram *service stuff*, respectively. These are the temporal statistics of query terms, and are known at query time, which can be utilized as features to estimate the distribution of relevant documents. In this case, the query trend of bigram *service stuff* provides a basis from which we can reconstruct the true relevance distribution.

Therefore, I explore two different approaches to exploit query trends:

- A linear ranking model that combines features based on temporal collection statistics of query unigrams and bigrams, their entropies, other related signals.

- A regression-based method that attempts to directly predict the distribution of relevant documents from unigram and bigram query trends.

These two approaches are further combined in an ensemble model, which additionally includes features derived from previous pseudo trend methods. Experimental evaluations show that my proposed methods are significantly more effective than competitive baselines, and detailed studies of feature combinations show the extent to which different types of temporal signals impact retrieval effectiveness. These pieces of work led to a short paper in ECIR [34] and a full paper in ICTIR [35]. The details of these approaches are presented in Chapter 4.

1.2 Multi-Perspective Lexical Modeling for Tweet Search

Though existing neural ranking models [10, 36] have achieved state-of-the-art effectiveness on many web search and NLP tasks. However, their effectiveness remain unknown on the Twitter data, whose setting is quite different as traditional webpages and newswire documents. I identify several important differences:

- **Document length**. Social media posts are much shorter than web or newswire documents. For example, tweets are limited to 280 characters. Thus, *ad hoc* retrieval in this domain contains elements of semantic matching because queries and posts are much closer in length. In particular, neural models that rely on sentence-level or paragraph-level interactions and global matching mechanisms [37] are unlikely to be effective.

- **Informality**. Idiosyncratic conventions (e.g., hashtags), abbreviations ("Happy Birthday" as "HBD"), typos, intentional misspellings, and emojis are prevalent in social media posts. An effective ranking model should account for such language variations and term mismatches due to the informality of posts.

- **Heterogeneous relevance signals**. The nature of social media platforms drives users to be actively engaged in many real-world news and events; users frequently take advantage of URLs or hashtags to gain exposure to their posts. Such heterogeneous signals are not well exploited by existing models, which can potentially boost ranking effectiveness when modeled together with the textual content.

To this end, I present a novel neural ranking model for *ad hoc* retrieval over short social media posts that is specifically designed with the above characteristics in mind. My model aims to represent the relevance of a social media post to a query in a multi-perspective manner, and has three key features:

- To cope with the informality of social media and to support more robust matching, I apply word-level as well as character-level modeling, with URL-specific matching. This allows us to exploit noisy relevance signal at different granularities.

- My model consists of hierarchical convolutional neural network layers to capture latent semantic soft-match signals between queries and tweets from multiple levels, starting from character-level and word-level to phrase-level, and finally to sentence-level.

- I match the learning representations between queries and tweets as well as URLs with pooling-based similarity measurement layers, where term importance weights are injected at each convolutional layer as priors.

Finally, all relevance signals are then integrated using a fully-connected layers to yield the final relevance ranking. Optionally, the neural matching score can be integrated with lexical matching via linear interpolation to further enhance effectiveness. The model is compared to the state-of-the-art feature-based and neural-based baselines on four standard twitter datasets. Detailed ablation study is also performed to examine the effectiveness contribution of each carefully-designed module. This work leads to a full paper (preprint available [9]) which is currently under review.

1.3 Modeling Temporal Contexts for Voice Search

In recent years, voice-based interactions with computing devices are becoming increasingly prevalent, driven by several convergent trends. The ubiquity of smartphones and other mobile devices with restrictive input methods makes voice an attractive modality for interaction: Apple's Siri, Microsoft's Cortana, and the Google Assistant are prominent examples. Google observed that there are more searches taking place from mobile devices than from traditional desktops [38], and that 20% of mobile searches are voice queries [39]. The success of these products has been enabled by advances in automatic speech recognition (ASR), thanks mostly to deep learning.

Increasing comfort with voice-based interactions, especially with AI agents, feeds into the emerging market on "smart homes". Products such as Amazon Echo and Google Home allow users to control a variety of devices via voice (e.g., “turn on the TV”, “play music by Adele”), and to issue voice queries (e.g., “what’s the weather tomorrow?”). The market success of these products demonstrates that people do indeed want to control smart devices via voice.

The context of my work is voice search on the Xfinity X1 entertainment platform by Comcast, one of the largest cable companies in the United States with approximately 22 million subscribers in 40 states. X1 refers to a software package distributed on top of Xfinity’s most recent cable box, which has been deployed to 17 million customers since around 2015. X1 can be controlled via the “voice remote”, which is a remote controller that has an integrated microphone to receive voice queries from viewers. The current deployed system is based on a combination of hand-crafted rules and machine-learned models to arrive at a final response. The system has a diverse set of capabilities, which increases query ambiguity and magnifies the overall challenge of understanding user intent. These capabilities range from channel change to entity search (e.g., sports team, person, movie, etc.). In addition, voice queries may involve general questions, from home security control to troubleshooting the wifi network, or may be ultimately directed to external apps such as Pandora.

1.3.1 Session Context Modeling

In the first part, I tackle the problem of navigational voice queries posed against an entertainment system, where viewers interact with a voice-enabled remote controller to specify the program (TV shows, movies, sports games) they wish to watch. If a viewer wishes to watch the popular series "Game of Thrones", saying the name of the program should switch the television to the proper channel. This is simpler and more intuitive than scrolling through channel guides or awkwardly trying to type in the name of the show on the remote controller. Even if the viewer knows that Game of Thrones is on HBO, finding the right channel may still be challenging, since entertainment packages may have hundreds of channels.

My work leverages a user's query session as contexts. The session concept in search engine originates from Web search and was formally defined in the early 2000s by Arlitt et al. [40] as "a sequence of requests made by a single end-user during a visit to a particular site". The basic assumption of a search session is that users will continuously modify and resend their queries until they have found their intended information or decided to give up. The behaviors users participate in a session include submitting a query, clicking on returned URLs, reading the returned documents and making decision of whether to reformulate a query or not.

User behaviors in voice sessions in entertainment domain are slightly different from that in Web search, since we mainly consider a user's reformulation behaviors as the context. This simplification is because of that searching for programs on an entertainment platform is not optimized for browsing webpages or clicking documents. Query reformulation is a more intuitive way to interact with TV when a user has not found her intended program yet. Indeed, such kinds of explicit reformulations are a gradual refinement and exposure of the user's true information need. For example, compare two sessions issued to the Xfinity entertainment platform for searching TV programs: [“tv shows”, “ncis”, “cargo fire”, “chicago fire”] and [“espn”, “chicago sports”, “chicago fire”]. Although both end in the same query, it is fairly clear that in the first case, the viewer is interested in the TV drama series “Chicago Fire” (since the previous queries all mention other drama series), whereas in the second session, it is clear that the viewer is interested in the sports team with the same name.

The idea of modeling sessions as contexts is operationalized through a hierarchical recurrent neural network (HRNN) model, in which a query is first represented by character and word sequences and converted to a semantic embedding representation through a RNN module, then another RNN module is stacked on top to combine the session context and query representation for user intent prediction. On a carefully-controlled laboratory experimental setting, the HRNN model outperforms competitive neural and non-neural baselines by more than 7.5 absolute points. Following the above promising laboratory experiments, the HRNN model was packaged as a standalone software module that was deployed into production to serve the live traffic. At present, the model serves millions of queries daily on the Comcast Xfinity X1 platform [41] for which the existing system provides no response (in other words, the most difficult queries). The model has substantially increased end-to-end coverage, reducing the number of unhandled queries by three quarters. On these queries, the HRNN definitively improved the customer experience two thirds of the time and arguably did not hurt in the other third. This work has been published as a full CIKM paper [42], and more details can be found in Chapter 6.

1.3.2 Multi-Task Learning

Despite the success of the HRNN model in production, I noticed two main shortcomings. First, the model adopts a classification-based approach, which is unable to predict unseen programs (e.g., newly-added content). Furthermore, its formulation has difficulty handling the long tail of rarely-watched programs. Second, my analysis of millions of queries [43] reveals that they span the gamut from program navigation to vague entertainment intents (e.g., looking for kids cartoons) to direct commands (e.g., turning on closed captioning) to queries that have nothing to do with entertainment (e.g., checking the weather). In fact, I find that around 40% of queries are either ambiguous viewing intents or not related to viewing a program at all. Obviously, a model based on program prediction cannot handle such queries. These two main shortcomings motivated us to explore a different design.

To this end, I propose a novel multi-task neural architecture [41] in Chapter 7 for query understanding that jointly performs three distinct tasks:

(1) **Program prediction** to directly identify the program or channel referenced in a viewer's utterance, out of a catalog of tens of thousands of programs and hundreds of channels.

(2) **Intent classification** to understand what the viewer wishes to do. The system recognizes around one hundred intents, ranging from TV commands (record a particular show) to entertainment intents that vary in specificity to non-entertainment intents (e.g., how to troubleshoot the wifi connection).

(3) **Query tagging** of each token in a viewer's utterance with domain-specific labels such as "entity", "channel", "modifier", etc., drawn from a tag set of roughly a dozen.

Program prediction, intent classification, and query tagging work together in a complementary way. In cases where the decision overlaps—for example, the system detects that the viewer's intent is to switch channels, which is confirmed by the tagging and program prediction modules—multiple sources of evidence reinforce the system's confidence in the decision. In cases where program prediction fails, tagged tokens in the query can serve as keywords for searching the program catalog. For example, given the query "watch Tom Hanks movies on HBO", program prediction may fail since the viewer is not looking for a specific program. The system, however, can parse the query into a logical form via the query tags as follows and return a list of options to the viewer. :

$$[\textit{person}=\text{“Tom Hanks”} \wedge \textit{category}=\text{“movies”} \wedge \textit{channel}=\text{“HBO”}]$$

My multi-task model was evaluated on a large-scale user log, showing further effectiveness gains against HRNN and other competitive baselines. More importantly, it provides a unified framework for understanding voice queries that can express a multitude of intents, shedding light on the design of other voice-enabled applications, such as Google Home, Amazon Alexa, etc. This part of work has been published as a short paper in SIGIR [43] and a full paper in KDD [41]. This work now serves as the core technology in Comcast Xfinity X1 entertainment platform, which won an Emmy award[2] in 2017 for the technical contribution in advancing television technologies.

1.4 Research Contributions

This dissertation makes five major contributions, which are successively introduced from Chapter 3 to 7.

1. Starting with estimating the distributions of relevant documents from the timestamps of an initial list of retrieved documents (pseudo trends), I propose an end-to-end neural framework for temporal modeling of pseudo trends. It consists of a lexical modeling component for producing query–document similarity vectors, and another temporal modeling component for capturing temporal relevance signals. In addition, I propose a novel continuous hidden Markov model for selecting more expressive terms for query expansion, which aims to establish connections to the relevant documents in bursty states. Extensive experiments on TREC Microblog collections verify the superior effectiveness of the proposed temporal techniques.

2. In addition to the pseudo trend methods, I also explore a new source of temporal signal based on term statistics evolutions in collections (query trends). I propose two different approaches to exploiting query trends: a linear ranking model based engineered features and a regression-based model that aims to recover the distribution of relevant documents directly from the query trends.

[2]`https://corporate.comcast.com/news-information/news-feed/comcast-wins-emmy-award-for-x1-voice-remote-technology`

In addition, I present a comprehensive study of combining query trend and pseudo trend signals in a linear ranking model. Experimental results show query trend methods alone are as competitive as the state-of-the-art pseudo trend methods, albeit substantially faster, while combining them together yields significant better results.

3. I highlight three important characteristics of social media posts that make lexical modeling over such collections different from searching web pages and newswire documents. Starting from these insights, I developed MP-HCNN, a novel neural ranking model specifically designed to address these characteristics. Extensive experiments on Twitter collections show that my proposed model significantly outperforms competitive learning-to-rank approaches and many recent state-of-the-art neural ranking models. To my best knowledge, the multi-perspective model is also the first neural ranking model developed specifically for ad hoc retrieval over social media posts.

4. I present novel deep neural network models to efficiently search TV programs with voice requests. This work serves as the first study on the voice search problem in the entertainment domain, in which users are sitting in front of TV and looking for programs to watch. I introduce several unique challenges in this domain, including the short lengths of voice queries, underlying speech recognition errors, and query ambiguities, which are solved in a novel probabilistic framework in which recurrent and feedforward neural network modules are organized in a hierarchical manner. Evaluations on a large real-world

dataset demonstrate the effectiveness of context-aware models, significantly outperforming strong baselines as well as the current deployed system. The best model has been launched into production, serving millions of voice queries per day.

5. I propose a novel multi-task problem formulation and neural architecture for general voice query understanding on an entertainment platform. This study is motivated by the fact that the intents of voice queries span the gamut from program navigation to vague entertainment intents (e.g., looking for kids cartoons) to queries that have nothing to do with entertainment. To this end, I decompose the task of query understanding into jointly performing three related tasks: program prediction, intent classification, and query tagging. The novel multi-task learning model, is evaluated through carefully-controlled laboratory experiments, which demonstrates further gains in effectiveness and increased system capabilities.

1.5 Outline

This dissertation is structured as follows: in Chapter 2, I discuss some related work on information retrieval, neural networks, voice search and multi-task learning. Then I present my temporal modeling techniques on pseudo trend in Chapter 3, and the query trend methods in Chapter 4. The multi-perspective approach for lexical modeling is introduced in Chapter 5. Next, I articulate the voice search problem on an entertainment platform and present a hierarchical recurrent neural network based model for session modeling in Chapter 6. The multi-task model is introduced in Chapter 7. Finally I conclude this dissertation and discuss some future work in Chapter 8.

Chapter 2: Related Work

2.1 Information Retrieval

Manning et al. [3] provide a broad definition of *information retrieval* as below:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

An information retrieval (IR) system takes a query q issued from a user reflecting his information need as input, ranging from a few words to an entire document. In some cases, the query q can also be a multimedia source, such as an image or a video, which is out of scope in this dissertation. The IR systems then process the indexes of collections stored in computers and return a list of information resources (usually documents d) to the user ordered by their degrees of relevance to the query. The main task for IR system is to compute the relevance score $P(d|q)$ for each document in the collection, given a query q.

2.1.1 Classical Ranking Models

Ranking is the core problem in many information retrieval and natural language processing tasks, i.e, *ad hoc* retrieval [35, 44, 45] and question answering [31, 46–49] and textual similarity join [50]. Developed decades ago, the three major categories of widely-used ranking models in IR are: vector space model [51], probabilistic model [4], and language model [52]. The vector space model represents queries and documents as vectors:

$$q = (t_{1,q}, t_{2,q}, ..., t_{m,q})$$
$$d_j = (t_{1,j}, t_{2,j}, ..., t_{n,j})$$

Each dimension represents a term. If a term occurs in a query q or a document d_j, its value is non-zero. The relevance of a document to a query is measured as the cosine similarity of the two vectors. There have been numerous mechanism for term weightings, and *tf-idf* is one of the best known schemas. In the tf-idf weighting, the weight of a term t in a document d is calculated by the product of *term frequency* tf and *inverse document frequency* idf, which are computed as below:

$$tf(t_i, d) = \text{counts of term } t_i \text{ in document } d$$
$$idf(t_i) = \frac{\text{number of documents contain term } t_i}{\text{number of documents in the collection}}$$

where term frequency refers to the relevance of the term within the document context, whereas the inverse document frequency corresponds to the specificity and "rareness" of the term.

Given the query and document representations, the IR system has an uncertain guess of how likely the document contain relevant content to the query. Probabilistic models aim to provide a more principled reasoning behind such uncertainty with probability theory. The difference of probabilistic models and vector space model is not that big as many probabilistic models also adopt tf-idf weighting. For a probabilistic model, it's just that, the relevance is not computed through a cosine similarity but by a slightly different formula motivated by probabilistic theory. BM25 [4] is a canonical example of probabilistic model, with its formula shown below:

$$P(d|q) = \sum_{t_i \in q} \log idf(t_i) \frac{(k_1 + 1) tf(t_i, d)}{k_1((1-b) + b \times (L_d / L_{ave})) + tf(t_i, d)}$$

where L_d and L_{ave} are the length of document d and the average length of all documents in the collection, k_1 and b are tunable parameters.

The language modeling approach models the likelihood of generating a query from the relevant documents. It first builds a probabilistic language model M_d from each document d, then ranks the documents by the possibility of the model generating the query. The basic and most commonly-used language model is Query Likelihood model [52]:

$$\begin{aligned} &P(d|q) \sim P(q|d)P(d) \\ &P(q|d) = \prod_{t_i \in q} P(t_i|M_d) = \prod_{t_i \in q} \frac{tf(t_i, d)}{L_d} \end{aligned}$$

where the above equation uses the simple maximum likelihood estimate to model the term generation probability $P(t_i|M_d)$. Note that some smoothing functions are also commonly used to incorporate collection frequency in $P(t_i|M_d)$, such as bayesian smoothing and Dirichlet smoothing.

2.1.2 Retrieval

In addition to ranking, retrieval is another major component of information retrieval systems. Due to the large size of document collection, it's infeasible to enumerate all the documents to compute the ranking score given a query. Therefore, the *inverted index* files are built to accelerate the computation by ignoring the documents that don't contain any query terms. Basically, inverted indexes are mappings from term ids to postings lists, which are lists of unique document identifiers that contain the particular term, along with some other information, such as term frequencies and positions of the term in each document.

In retrieval, there are two main approaches to traverse inverted indexes and compute query-document scores: document-at-a-time (DAAT) and term-at-a-time (TAAT). In DAAT, the inverted indexes are sorted by document identifiers and term frequencies are stored separately. When a query arrives, DAAT processes document one by one, computing the query-document score until moving to the next document. As a result, we only need to process documents that appear in the union of query terms' postings lists. Some additional optimization strategies are applied in DAAT for minimizing the scoring computation when traversing the postings lists for all of the query terms simultaneously. This is achieved by precomputing a value v_t that represents the maximum contribution term t can have for the scoring model. When the sum of the v_t scores is less than the minimum score of top k documents, we can safely skip to the next postings list to save computation. Such idea is implemented in WAND [53] and its successor BMW [54], which are the two most popular DAAT strategies today.

In TAAT evaluation, as the name suggests, we process the postings lists of each query term in turn. The postings lists are stored in a separate manner as DAAT, where document identifiers are grouped by term frequency. Within each grouping, document ids are sorted from small to large, while the groupings are arranged in a decreasing order of term frequency. This enables further optimization such as early termination, in which we can terminate the processing of a postings list when the term frequency falls behind a threshold. Another useful optimization trick is to process terms with higher idf values first, since higher idf values can imply higher contribution to the scoring model. Such ideas have been implemented by Moffat et al. [55] and Anh et al. [56].

In addition to DAAT and TAAT, another thread of query evaluation strategy is called score-at-a-time (SAAT), where postings lists are ordered by a decreasing order of impact score. Lin and Trotman propose JASS [57], a modern implementation of SAAT, followed by an empirical comparison between DAAT and SAAT by Crane et al. [58].

2.1.3 Learning to Rank

Learning to rank (L2R) is a field that takes advantage of recent advances in machine learning to improve ranking effectiveness. Existing work on L2R can be summarized into three main categories: pointwise, pairwise, and listwise. The main difference lies in the problem formulations with different assumptions, input/output spaces and loss functions. Pointwise methods, such as logistic regression [59], focus on learning a relevance score for each query-document pair represented in a feature space, while pairwise approaches, such as LambdaMART [60] and RankSVM [44], aim to learn the preference between a pair of documents to a query. Listwise approaches, such as ListNet [45], directly optimize the input list of documents to a query to find the best-ranked list. The major drawback of L2R is that it requires effective hand-crafted feature engineering, which can be time-consuming, incomplete, and difficult to generalize to other problems.

2.1.4 Temporal Information Retrieval

There is a long thread of research exploring the role of temporal signals in search [16, 17, 21, 22, 26, 29, 34, 61], and it is well established that for certain tasks, better modeling of the temporal characteristics of queries and documents can lead to higher retrieval effectiveness.

For example, Jones and Diaz [15] study the temporal profiles of queries, classifying queries as atemporal, temporally ambiguous, or temporally unambiguous. They showed that the temporal distribution of retrieved documents can provide an additional source of evidence to improve rankings. Building on this, Li and Croft [26] introduce recency priors that favor more-recent documents. Dakka et al. [61] propose an approach to temporal modeling based on moving windows to integrate query-specific temporal evidence with lexical evidence. Efron et al. [22] present several language modeling variants that incorporate query-specific temporal evidence. The most direct point of comparison to our work (as discussed in Chapter 3.2) is the use of non-parametric density estimation to infer the temporal distribution of relevant documents from an initial list of retrieved documents [16,33].

There have been several other studies of time-based pseudo relevance feedback. Keikha et al. [62] represent queries and documents with their normalized term frequencies in the time dimension and used a time-based similarity metric to measure relevance. Craveiro et al. [63] exploit the temporal relationship between words for query expansion. Choi and Croft [64] present a method to select time periods for expansion based on users' behaviors (i.e., retweets).

In addition to ranking, modeling temporal signals has also been shown to benefit related tasks such as behavior prediction [65], time-sensitive query auto-completion [66], and real-time event detection [67,68]. For example, Radinsky et al. [65] build predictive models to learn query dynamics from historical user data.

2.1.5 Evaluation

The two fundamental measures for evaluating the effectiveness of an information retrieval system are *precision* and *recall*. The precision measures the ratio of correctness of identified relevant documents, whereas the recall measures the ratio of correct documents we have identified from the true candidate pool. Applications need to balance precision and recall to obtain satisfactory user experiences. For example, web search emphasizes more on precision and hopes the top ranked documents to be more relevant, while professional searchers, such as paralegals, may try to get as high recall as possible. *F-score* (or F_1 score) provides a way to combine precision and recall without any prior preference:

$$\mathrm{F}_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

However, precision, recall and F_1 are both set-based measures computed on unordered sets of documents, which makes them inappropriate for evaluating ranked retrieval results. As a result, precision-recall curve plots the precision value as a function of recall, reflecting the changes of system's effectiveness (precision) as the number of retrieved documents (recall) increases. An approximation to the precision-recall curve is the *average precision* (AP) metric. It starts from the highest ranked document, and computes the precision at every relevant document in the ranked list until the list is exhausted, then it divides the sum of these precision values by the total number of retrieved documents. The mean of AP values across a set of queries is called *mean average precision* (MAP), which has become the most standard effectiveness measure in the TREC community.

Other measures are also commonly seen in the IR community. For example, Precision at K measures the precision of the top K retrieved documents. DCG at

K is used in cases where the relevance assessments are not made in binary scale, which refers to *discounted cumulative gain* and computes the total gain of the top K retrieved documents. The gain of the i-th document d_i in the ranked list is measured as $g_i = relevance(d_i)/max(1, \log(i))$. The NDCG at K normalizes the DCG score into [0, 1] range by dividing the upper bound.

2.2 Deep Learning

2.2.1 Convolutional Neural Networks

The core building block of convolutional neural networks [2] is the convolutional layer. A convolutional layer comprises of a number of convolutional filters (or kernels), with each filter having a small receptive window with learnable parameters. In most natural language processing applications, the inputs to the convolution layer is a two-dimension matrix, which denotes an embedding representation of a text sequence. Assume the input is a matrix $\mathbf{D}$ with shape $\mathbb{R}^{n\times l}$, where n is the number of elements in the input and l is the size of embedding. The small receptive window of a kernel can be parameterized by a weight term $W \in \mathbb{R}^{w\times h}$ and a bias term $b \in \mathbb{R}$. We move this filter through the input text gradually, and at each position (i, j), we compute the dot product the entries of the filter weight and input matrix:

$$O_{i,j} = \sum_{p=i}^{i+w-1} \sum_{q=j}^{j+h-1} W_{p,q} D_{p,q} + b$$

.

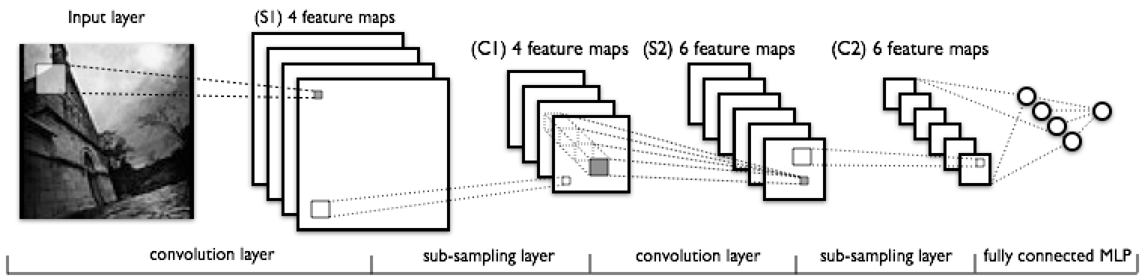


Figure 2.1: The well-known LeNet architecture [2]

The output O has a shape $\mathbb{R}^{(n-w+1)\times(l-h+1)}$. It's also a common practice to set the filter height h as the same as the embedding size l, from which we will obtain a output vector $O \in \mathbb{R}^{n-w+1}$. The major insight of a convolutional operation is to capture spatially local correlation via parameter-sharing receptive filters. A convolutional operation is often coupled with pooling mechanism, such as max pooling and min pooling, to extract the discriminant features and filter noises from raw convolutional outputs. Figure 2.1 shows the well-known LeNet architecture [2], which comprises of multiple convolution layers in a sequential manner. Convolutional neural networks have achieved huge successes in many applications, like image recognition [69], object detection [70], language modeling [30], etc.

2.2.2 Recurrent Neural Networks

Recurrent neural networks (RNN) is a classical category of neural networks where connections between nodes form a directed graph to exploit dynamic temporal information along a sequence. It has many applications in sequence modeling tasks, such as speech recognition [71], hand writing recognition [72], etc. Long Short-Term Memory (LSTM) [73] networks is a type of RNN. It's well-known for being able to

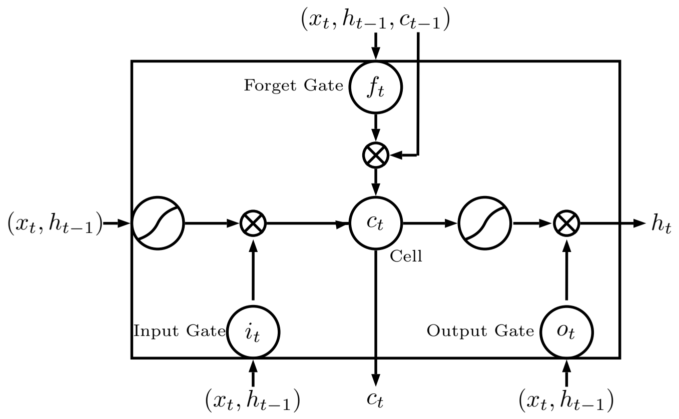


Figure 2.2: LSTM Cell

capture long-range context dependences over input sequences. This is accomplished by using a sequence of memory cells to store and memorize historical information, where each memory cell (shown in Figure 2.2) contains three gates (input gate, forget gate, and output gate) to control the information flow. The gating mechanism enables the LSTM to handle the gradient vanishing/explosion problem for long sequences of inputs.

Given an input sequence $\mathbf{x} = (x_1, ..., x_T)$, an LSTM model outputs a sequence of hidden vectors $\mathbf{h} = (h_1, ..., h_T)$. A memory cell at position t digests the input element x_t and previous state information h_{t-1} to produce the more recent state h_t as follows:

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \sigma(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
h_t &= o_t \cdot \tanh(c_t)
\end{aligned}$$

where W terms are weight matrices, b terms represent bias vectors, σ is the sigmoid activation function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell vectors, with each having the same size as the output vector h.

In many application scenarios, the input sequence $\mathbf{x}$ can vary in length for different instances (i.e., queries can have different number of words and characters in our task). There are two standard ways to handle this variable length issue. One way is to do an initial scan over a single batch or the whole dataset to obtain the maximum sequence length, then create an array of memory cells with the maximum length. Whenever a sequence element x_t arrives, the memory cell at index t will digest the input element and produce the hidden state h_t. The other way is to dynamically allocate space for storing new memory cells only when the arriving instance $\mathbf{x}$ has larger length than all previous instances. The created LSTM memory cells all share the same parameters.

A natural extension to RNN/LSTM is to model the sequence from both positive time direction and negative time direction, which is proposed as bidirectional RNN [74]. Another popular category of RNN is called Gated Recurrent Units (GRU), introduced by Cho et al. [75] in 2014. It achieves on-par performance as LSTM in tasks like machine translation, polyphonic music modeling, and speech signal modeling, but has fewer parameters than LSTM as it lacks an output gate.

2.2.3 Semantic Similarity Measurement

Recent years have witnessed many successes of deep learning in natural language processing tasks [27, 76–79], such as question answering [31, 46], paraphrase detection [80], machine comprehension [81,82], and textual semantic similarity modeling [76]. Many of these tasks can be treated as variants of a *semantic matching* problem, where two pieces of texts are jointly modeled through distributed representations of sentences for similarity learning. Various neural network architectures have been proposed for modeling semantic matching. For example, the classical Siamese architectures [83] have been applied to many neural network models [30,31,46] nowadays, where two pieces of texts share the same module and parameters for representation learning. For machine translation, the sequence-to-sequence model [27, 28] has been widely used, which aims to produce the next token given the source sentence and tokens have been generated. The attention mechanism, first introduced by Bahdanau et al. [84] to enhance the sequence-to-sequence model for token prediction based on semantic closeness to past tokens in machine translation, has become popular in many other tasks, such as question answering [85], machine comprehension [82], relation extraction [86], sentiment analysis [87] and recommendation [88].

2.2.4 Neural Information Retrieval

The current neural approaches for IR can be divided into representation-based [11,31,36] and interaction-based [9,10,89–91] approaches. The early attempts on neural IR mainly focus on representation-based modeling between query and document, such as DSSM [11], C-DSSM [36], and SM-CNN [31]. DSSM [11] is a classical NN architecture for Web search that maps word sequence to character-level trigrams by using a word hashing layer, and then feeds the dense hashed features to a multi-layer perceptron (MLP) for similarity learning. C-DSSM [36] extends this idea by replacing the MLP in DSSM with a convolutional neural network-based (CNN) layer to capture local contextual signals from neighboring character trigrams. SM-CNN can be viewed as a hybrid approach with a main component of a convolutional layer for learning discriminative representations of query and document and a feature layer that exploits hand-crafted features.

Interaction-based approaches [10, 89–91] model on the similarity matrix of word pairs from the query and document. The preparation of similarity matrix is usually computed through word embeddings, such as word2vec [92], which solves the sparsity issue of count-based approaches. The DRMM approach [10] introduces a pyramid pooling technique to convert the similarity matrix to histogram representations, on top of which a term gating network aggregates weighted matching signals from different query terms. Inspired by DRMM, Xiong et al. [89] propose K-NRM that introduces a differentiable kernel-based pooling technique to capture matching signals at different strength levels. Dai et al. [90] extends this idea to model soft-match signals for n-grams with an additional convolutional layer. The DUET model [91] combines the representation-based and interaction-based idea with a global component for the semantic match and a local component for the exact match.

There are also some work that combine neural networks with learning to rank. Previously, I have introduced a noise-contrastive estimation technique [46] with pairwise sampling strategies on pointwise neural network models, which achieves state-of-art performance on a popular benchmark of answer sentence selection. Ai et al. [93] propose a listwise context model with attention-based loss function, which outperforms many learning-to-rank and neural methods by a large margin.

2.3 Voice Search

Along with rapid improvements in speech recognition technologies, there has been work on tackling voice search [94–98] in different applications. However, to our knowledge we are the first to focus on voice queries directed at an entertainment system. How is this particular domain different? The setting is obviously different—in our case, viewers are clearly sitting in front a television with an entertainment intent. To compare and contrast viewers' actual utterances, we can turn to previously-published work that studied the characteristics of voice search logs, especially in comparison to text search data [99–102]. Schalkwyk et al. [102] report statistics of queries collected from Google Voice's search logs which found short queries, in particular 1-word and 2-word queries, were more common in voice search setting, while long queries were much rarer. In contrast, in a more recent study, Guy et al. [101] report that voice queries tend to be longer than text queries, based on a half-million query dataset from the Yahoo! mobile search application. The average length across 32M voice queries was 2.04 in our dataset, much shorter than the reported average of 4.2 for Yahoo voice search[1] [101].

There is also research on voice query reformulations which is relevant to our work on modeling sessions [103–106]. For example, Jiang et al. [103] analyze different types of voice recognition errors and users' corresponding reformulation strategies. Hassan et al. [104] build classifiers to differentiate between reformulated and non-reformulated query pairs. The study by Shokouhi et al. [105] suggests that users don't prefer to switch between voice and text when reformulating a new query. A more recent paper [106] propose an automatic way to label voice queries by examining the post-click and reformulation behaviors, which produced a large amount of "free" training data to reduce ASR errors.

2.4 Multi-Task Learning

Multi-task learning (MTL) is a machine learning paradigm where objectives for multiple related tasks are optimized together. The main intuition is that when multiple tasks are not independent, joint training reinforces individual tasks and results in better generalization across shared parameters. Since its introduction [107], MTL has been studied for many different problems, including computer vision [108, 109] as well as text and web applications [110–112]. Collobert and Weston use MTL to jointly learn six different NLP tasks [111]. For web search ranking, Chapelle et al. [112] claim that MTL yields improvements by allowing implicit data sharing and regularization across different tasks using different datasets. Deep learning

[1]Similar conclusions follow for other length-based statistics: median was 2 (vs. 4), maximum was 69 (vs. 109), and standard deviation was 1.23 (vs. 2.96).

has recently started to receive more attention from MTL: for example, multi-task encoder–decoder architectures are proposed to improve accuracy in machine translation by jointly training for parsing and caption generation [108].

Chapter 3: Temporal Modeling of Pseudo Trends for Tweet Search

3.1 Introduction

There is a large body of literature in information retrieval that has established the importance of understanding and modeling the temporal distribution of documents as well as queries for various information seeking tasks [16,17,21,22,26,61,65]. This is particularly important when searching rapidly-evolving, real-time social media streams such as Twitter, which is the focus of this work. Given an information need expressed as a query, we wish to develop ranking models that incorporate temporal information and return relevant tweets. I refer this problem as temporal ranking to emphasize the need to model temporal aspects of the information need as well as the document collection.

One successful approach to temporal ranking is to estimate the distribution of relevant documents using the distribution of document timestamps from the results of an initial query [16]. This approach is motivated by Efron et al.'s temporal cluster hypothesis [16], which stipulates that in search tasks where time plays an important role (such as tweet search), relevant documents tend to cluster together in time, and that this property can be exploited to improve search effectiveness. Just as van Rijsbergen's "classic" cluster hypothesis suggests that documents relevant to a query form clusters in term space, Efron et al. suggest that documents relevant to a query form clusters along a timeline.

The temporal cluster hypothesis is illustrated by the visualizations in Figure 1.1, similar to those presented by Efron et al. [16], which help illustrate the intuition behind my techniques. These visualizations show three queries (topics) from the TREC 2011 Microblog Track. In each timeline, the query time (the time at which the query was issued) is anchored to the right edge; the x-axis shows time prior to the query time, in days. Dots show tweets that were retrieved by participating teams and evaluated by assessors (i.e., the pools): green dots are relevant, red dots are highly relevant, and gray dots are not relevant. The underlying blue bars show the distribution of relevant and highly-relevant tweets as a histogram. As we can see, relevant tweets for query 14 and 30 tend to cluster together in time, while relevant tweets for query 6 are more evenly distributed. Across all queries from the TREC test collections, we can observe many timelines that exhibit temporal clustering (like query 14 and 30).

Efron et al. [16] proposes an approach based on kernel density estimation to estimate the temporal distribution of relevant documents, where each document's timestamp is viewed as a Gaussian kernel and the estimation process takes a weighted average of all kernels. This approach has demonstrated state-of-the-art effectiveness on modeling pseudo trends for ranking. Inspired by the recent success of neural networks [27,84], I explore an alternative approach for temporal modeling of pseudo trends using recurrent neural networks. Such models have been successfully applied to many sequence learning tasks in natural language processing where the modeling units are temporally dependent (e.g., tagging and parsing). I draw a connection between the temporal clustering of documents, where the relevance of one document may affect its neighbors, to a sequence learning task, and explore the hypothesis that recurrent neural networks provide a rich, expressive modeling framework to capture such temporal signals.

To this end, I propose a unified neural framework to integrate lexical and temporal relevance signals. The framework consists of a lexical modeling component for producing query–document similarity vectors and another temporal modeling component for capturing temporal relevance signals. I start with a few state-of-the-art neural ranking models [11,30] as the lexical component, where the temporal model is stacked on top to explore the temporal interactions between neighboring documents.

In addition to directly modeling pseudo trend for reranking, I also consider to using pseudo trend for query expansion. Query expansion techniques, especially those based on pseudo-relevance feedback, aim to solve the classical vocabulary mismatch issue by augmenting the initial query with teams that are more likely to appear in relevant documents. In standard formulations of pseudo-relevance feedback, the timestamp of a document is not considered in identifying expansion terms—yet we know from Figure 1.1 that relevant documents are bursty and usually occur in temporal clusters, and that this signal should be incorporated into the relevance feedback model. The main insight of my work is that term expansions should be biased to draw from documents that occur in the bursty temporal clusters. This is formally captured by a continuous hidden Markov model (cHMM), in which the temporal distribution of documents (not necessarily relevant) is represented by a sequence of hidden states; the probability of generating a particular number of documents from each state follows a Gaussian distribution. When identifying term expansions, we only select documents from bursty states. Experimental evaluations on test collections from the TREC 2011 and 2012 Microblog Tracks show that this approach is significantly more effective for selecting informative expansion terms than standard query expansion techniques without temporal information.

I make the following contribution in this chapter:

- I present, to my knowledge, the first end-to-end neural network architecture that integrates lexical and temporal signals. Using the best lexical modeling component, my model is able to obtain significant improvements over competitive temporal baselines on standard tweet test collections.

- I introduce a novel query expansion technique that incorporate time information to select the most informative expansion terms with continuous hidden Markov model. Experiments on standard TREC collections demonstrate the state-of-the-art effectiveness of my approach.

This chapter is organized as follows: I first discuss some related work on temporal modeling of pseudo trends as a background in Chapter 3.2, then I introduce the end-to-end neural framework for integrating lexical and temporal evidence in Chapter 3.3. Next, I present my approach of utilizing pseudo trends for temporal query expansion in Chapter 3.4. The Twitter datasets are introduced in Chapter 3.5, which we evaluate the proposed models across this chapter and the next two chapters. Evaluations are presented in Chapter 3.6 and the conclusion follows in Chapter 3.7.

3.2 Background and Related Work

As a starting point, I introduce several existing methods for modeling pseudo trends, which are used as baselines for comparison in the experiments in this chapter and the next chapter. Let's first consider the simple query-likelihood approach in the language modeling framework [52].[1] Documents are ranked by $P(D|Q) \propto P(Q|D)P(D)$, where $P(Q|D)$ is the likelihood that the language model that generated document D would also generate query Q, and $P(D)$ is the prior distribution.

One of the simplest way to let time influence ranking was proposed by Li and Croft [26], in the form of a document prior that favors recently published documents. If T_D is the timestamp associated with document D, $P(D)$ could take the form of an exponential distribution (with rate parameter $\lambda \geq 0$):

$$P(D) = \lambda e^{-\lambda T_D} \tag{3.1}$$

Though previous studies have shown that recency priors increase overall effectiveness, by definition they are query-independent. This, however, is problematic because we know that the dependencies between time and relevance vary from query to query [15]. Figure 1.1 clearly shows that this is the case: we would expect recency

[1]Note that this section, up through Chapter 3.2, reuses some of the text from Efron et al. [16] (with the permission of those authors).

priors to be effective for query 30, but such techniques are not likely to be effective for information needs represented by query 14, where the relevant documents are not clustered close to the query time.

To address this issue, Dakka et al. [23] proposed a query-specific way to combine lexical and temporal evidence in the language modeling framework by separating the two components: W_D, the words in the document and T_D, the document's timestamp. This leads to the following derivation:

$$\begin{aligned} P(D|Q) &= P(W_D, T_D|Q) && (3.2)\\ &= P(T_D|W_D, Q)P(W_D|Q) && (3.3)\\ &\sim P(W_D|Q)P(T_D|Q) && (3.4) \end{aligned}$$

where the last step follows from Eq. (3.3) if we assume independence between content and temporal evidence. More generally, we take the view that there are two sources of evidence that we need to integrate in document ranking: $P(R|W_D, Q)$, based on document content, and $P(R|T_D, Q)$, based on temporal evidence.

For content relevance, we adopt a standard query-likelihood model, that is,

$$P(R|W_D, Q) \overset{def}{=} P(Q|D). \tag{3.5}$$

By assuming term independence and a multinomial language model, we have:

$$P(Q|D) = \prod_{i=1}^{c(Q)} P(q_i|\theta_D) \tag{3.6}$$

for the language model θ_D, where $c(Q)$ is the number of terms in the query. Using Bayesian updating with a Dirichlet prior parameterized by the real vector $\mu P(w|C)$, we have the estimator:

$$\hat{P}(w|D) = \frac{c(w, D) + \mu P(w|C)}{c(w, D) + \mu} \tag{3.7}$$

where $P(w|C)$ is the term probability given the language model of the entire corpus, and $c(w, D)$ is the count of term w in document D.

Now consider $P(R|T_D, Q)$, the probability of relevance of document D to Q given temporal information. To combine content and temporal evidence, we can use a log-linear model, as Efron et al. [16] have done. For a parameter $\alpha \in [0, 1]$, we can rank documents as follows:

$$\begin{aligned} \log P_\alpha(R|D, Q) = Z_\alpha + (1 - \alpha) \log P(R|W_D, Q) \\ + \alpha \log P(R|T_D, Q) \end{aligned} \tag{3.8}$$

where Z_α is a normalization constant. Since Z_α does not depend on D for ranking, we can ignore it. The parameter α can be estimated from a set of training topics.

In essence, we can think of Eq. (3.8) as a very simple linear feature-based ranking model [113], albeit with only two features. The more general form of such a model is:

$$S_d = \sum_i \alpha_i \cdot F_i(d, q) \quad \text{s.t.} \sum_i \alpha_i = 1. \tag{3.9}$$

Accepting this view, $P(R|T_D, Q)$ no longer needs to be a probability distribution, but can be any arbitrary feature (e.g., of a document's timestamp). In fact, it doesn't need to be just one single feature, which allows us the flexibility to combine multiple sources of temporal evidence in a well-established document ranking framework.

One natural source of temporal evidence for document ranking is the temporal distribution of documents retrieved by an initial bag-of-words query. This thread of work was explored by Efron et al. [16] and later expanded by me in [33] for reproduction. Here, I summarize these work.

The theoretical motivation for modeling the distribution of initial retrieved documents is what Efron et al. [16] call the *temporal cluster hypothesis*: that relevant documents tend to cluster together in time. We assume that there is a density f_Q over the time span of the document collection, such that f_Q is large for times where relevant documents are likely to appear and small during times where we are unlikely to find relevant documents. Intuitively, we want to promote documents whose timestamps coincide with large values of f_Q, i.e., temporal regions where relevant documents "cluster together".

To estimate f_Q, Efron et al. take advantage of kernel density estimation (KDE), which is a non-parametric method to approximate a density by analyzing data generated from that density, applied to the distribution of document timestamps from an initial bag-of-words query.

Let $\{x_1, x_2, \ldots, x_n\}$ be an i.i.d. sample drawn from some distribution with an unknown density f. We are interested in estimating the shape of this function f.

Its kernel density estimator is:

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=0}^{n} K\left(\frac{x - x_i}{h}\right) \tag{3.10}$$

where $K(\cdot)$ is the kernel—a symmetric but not necessarily positive function that integrates to one—and $h > 0$ is a smoothing parameter called the bandwidth. Though many kernel functions are viable, Efron et al. use the common Gaussian distribution, such that:

$$K\left(\frac{x - y}{h}\right) = \mathcal{N}\left(\frac{x - y}{h}, 0, h\right) \tag{3.11}$$

where $\mathcal{N}$ is the normal density. Efron et al. chose the Gaussian kernel for two reasons. First, as shown below, it gives a ready plug-in value for the optimal bandwidth h. Second, experimentally Efron et al. found that the choice of kernels has almost no effect on the effectiveness of the methods.

A kernel density estimate is very similar to a histogram. However, KDE requires no binning of data, offloading the bias/variance tradeoff to the choice of bandwidth, which has well-defined methods of selection. One key advantage in using KDE versus histograms for estimating f is KDE's ability to handle weighted observations naturally. If we have $\{\omega_1, \omega_2, \ldots, \omega_n\}$, a vector of non-negative weights on our observed X's such that $\sum \omega_i = 1$, then

$$\hat{f}_\omega(x) = \frac{1}{nh}\sum_{i=0}^{n} \omega_i K\left(\frac{x - x_i}{h}\right) \tag{3.12}$$

is also a proper density: $\hat{f}_\omega$ is similar to $\hat{f}$, except that we allocate different weights to the kernels. As noted by Hall and Turlach [114], ω_i can be interpreted as the probability associated with x_i. Unless otherwise specified, in this paper, the phrase *kernel density estimate* refers to Eq. (3.12).

KDE, via Eq. (3.12), presents a simple framework for weighting observations (document timestamps) during density estimation. The intuition behind the weight ω_i for document D_i is that this quantity corresponds to our prior belief that the corresponding timestamp T_i was truly generated by f_Q. Efron et al. proposed three weighting schemes:

- *Uniform weights.* The simplest approach is to give all documents in the initial results equal weights.

- *Score-based weights.* We can weight each document based on its query-likelihood, i.e.,

$$\omega_i^s = \frac{P(Q|D_i)}{\sum_{j=1}^n P(Q|D_j)}. \tag{3.13}$$

- *Rank-based weights.* We can adopt a rank-based scheme that preserves the ordering in the initial results, but not the actual scores, via an exponential distribution:

$$\omega_i^r = \frac{\lambda e^{-\lambda r_i}}{\sum_{j=1}^n \lambda e^{-\lambda r_i}} \tag{3.14}$$

  where $\lambda > 0$ is the rate parameter of the exponential and r_i is the rank of document D_i in R. Though we could leave λ as a tuneable parameter, a simpler approach is to use the maximum likelihood estimate. If R contains

n documents, the MLE of λ is simply $\frac{1}{\bar{r}}$, where $\bar{r}$ is the mean of the ranks $1, 2, \ldots, n$.

All the KDE techniques proposed above are applied over an initial ranked list of documents retrieved using a bag-of-words query, and thus require no manual intervention. However, as an upper bound oracle condition, we can perform KDE directly on the known relevant documents (from assessor judgments). This quantifies the effectiveness upper bound of models that take the form of the log-linear model in Eq. (3.8) and provides a point of reference for comparing other models. The KDE method introduced above is also considered as the state-of-the-art on modeling pseudo trends.

3.3 Neural Framework to Integrate Lexical and Temporal Signals

By now I have introduced a number of existing methods to model the pseudo trends. In this section, I present a neural network-based approach to model pseudo trends, which integrates lexical and temporal signals in an end-to-end manner, as shown in Figure 3.1. The overall architecture consists of distinct components for lexical modeling, to capture query–document similarity, and temporal modeling, to capture relevance signals contained in the temporal sequencing of documents. The two components are independent and in particular we can view the lexical modeling component as a black box, allowing us to explore different architectures. However, the entire model is trained end-to-end in a two-stage process, which is explained in Chapter 3.3.3.

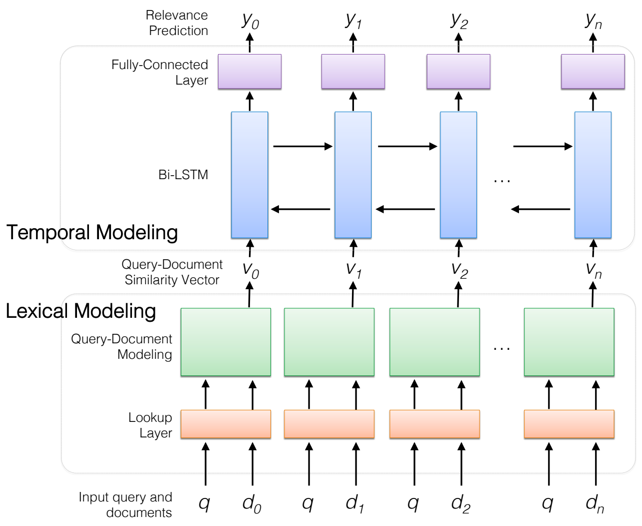


Figure 3.1: My neural network architecture that integrates lexical and temporal signals. The lexical modeling component can be viewed as a black box for producing query–document similarity vectors. A temporally-ordered sequence of these vectors feed into our bidirectional LSTM for temporal modeling.

Lexical Modeling. The architecture for the lexical modeling component is shown in the lower half of Figure 3.1, where each "slice" of the network is identical (i.e., with shared parameters). Each instance of the model takes as input a query and a document to generate a query–document similarity vector v. This is accomplished by translating an input sequence of tokens (either the query or the document) into a sequence of distributional vectors $[w_1, w_2, ...w_{|S|}]$, where $|S|$ is the length of the token sequence, from a word embedding lookup layer. The resulting matrix then feeds into a neural network. At a high level, this similarity model can be viewed as a black box, but I describe several instantiations below.

Temporal Modeling. The architecture of the temporal modeling component is shown in the upper half of Figure 3.1. I use a bidirectional LSTM where the inputs are the query–document similarity vectors from the lexical modeling component, sorted in time order. That is, documents from the training set are temporally ordered, and the lexical modeling component is applied to the query paired with each individual document to yield a collection of query–document similarity vectors $\{v_0, v_1, \ldots, v_n\}$. The output of the bidirectional LSTM feeds into a fully-connected layer plus softmax to yield a prediction of document relevance y. Note that each instance of the fully-connected layer and softmax share parameters. In what follows, I describe each of the components in detail.

3.3.1 Lexical Modeling Component

In this part, I considered three existing approaches to generating query–document similarity vectors. All three adopt what is commonly known as a "Siamese" structure [83], with two subnetworks processing the query and document in parallel, yielding a "joined" representation that feeds into a relevance modeling component:

DSSM [11]: The Deep Structured Semantic Model (DSSM) is an early application of neural networks to web search. One of its key features is a word hashing layer that converts all tokens into trigrams, which greatly reduces the size of the vocabulary space to help handle misspellings and other noisy text input. In parallel, the dense hashed features from either the query or the document feed into a multi-layer perceptron with a softmax on top to make the final relevance prediction. I take the intermediate semantic representation of the query and document, just before the softmax, as the query–document similarity vector.

SM-CNN [31]: The convolutional neural network (CNN) proposed by Severyn and Moschitti has been previously applied to question answering as well as tweet reranking. In both the query and document subnetworks, convolutional feature maps are applied to the input embedding matrix, followed by ReLU activation and simple max-pooling, to arrive at a representation vector x_q for the query and x_d for the document. Intermediate representations are concatenated into a single vector at the join layer:

$$\mathrm{x}_{\mathrm{join}} = [\mathrm{x}_q^T; x_{\mathrm{sim}}; \mathrm{x}_d^T; \mathrm{x}_{\mathrm{feat}}^T] \tag{3.15}$$

where x_{sim} defines the *bilinear* similarity between x_q and x_d. The final component consists of "extra features" $\mathrm{x}_{\mathrm{feat}}$ derived from four word overlap measures between the query and the document.

In the original SM-CNN model, the join vector feeds into a fully-connected layer and softmax for final relevance prediction, but in my approach I use the join vector $\mathrm{x}_{\mathrm{join}}$ as the query–document similarity vector.

Multi-Perspective CNN [30]: This approach was developed at roughly the same time as the SM-CNN model and can be described as an ensemble of convolutional neural networks. The "multi-perspective" idea refers to different types of convolutional feature maps, pooling methods, and window sizes to capture semantic similarity between textual inputs. Another key feature is a similarity measurement layer to explore the interactions between the learned convolutional feature maps at different levels of granularity. At the time the work was published, it achieved state-of-the-art effectiveness on several semantic modeling tasks such as paraphrase detection and question answering (although other models have improved upon it since).

As with the SM-CNN model, I take the joined representation just before the fully-connected layer and softmax as the query–document similarity vector.

3.3.2 Temporal Modeling Component

On top of a sequence of temporally ordered query–document similarity vectors (the output of the lexical modeling component), I layer a recurrent neural network to capture the temporal clustering of relevant documents (see Figure 3.1). Compared to kernel density estimation, I hypothesized that recurrent neural networks provide a richer, more expressive modeling framework to capture temporal signals that can yield more effective results.

I used a variant of recurrent neural networks, bidirectional LSTM (BiLSTM) [73], which have been successfully applied to text similarity tasks [76, 115]. One key feature of LSTMs is their ability to capture long-range dependencies, and a bidirectional LSTM consists of two LSTMs that run in parallel in opposite directions: one (forward LSTM^f) on the input sequence and the other (backward LSTM^b) on the reverse of the sequence. At time step t, the BiLSTM hidden state h_t^{bi} is a concatenation of the hidden state h_t^{for} of LSTM^f and the hidden state h_t^{back} of LSTM^b, representing the neighboring contexts of input v_t in the temporal sequence.

Given BiLSTM output h_t^{bi}, the prediction output y_t of my temporal ranking model at time step t is obtained by passing the BiLSTM output through a fully-connected layer and softmax as follows:

$$g_t = \sigma(W^m \cdot h_t^{\text{bi}} + b^m) \tag{3.16}$$

$$y_t = \text{softmax}(W^p \cdot g_t + b^p) \tag{3.17}$$

where the output y_t indicates the relevance of the document at time step t. W^* and b^* are learned weight matrices and biases.

3.3.3 Model Training

Although this neural network architecture breaks down into two distinct components, I train the entire model end-to-end in a two-stage manner, with stochastic gradient descent to minimize negative log-likelihood loss of the entire model. In each epoch, I first train the lexical modeling component independently, and then use the results to generate inputs to the temporal modeling layer. The losses from all documents are summed together to train the BiLSTM and the top layers, while the underlying lexical component is held constant. The reason for this two-stage approach is to restrict the search space during model optimization, since we have limited labeled data for training.

At inference time, I first retrieve candidate documents from the collection using a standard ranking function. These documents are then ordered chronologically and fed into the model. The classification scores outputted by each step of the BiLSTM (corresponding to the processing of that document) are used to resort the ranked list, which I take as final output for evaluation. The evaluations of the above model is presented later, in Chapter 3.6.

3.4 Temporal Query Expansion with Pseudo Trends

A longstanding challenge in information retrieval is the issue of vocabulary mismatch, where query terms are not present in relevant documents. This problem is especially severe in searching social media posts such as tweets due to their short lengths and frequent use of informal language. Query expansion techniques, especially those based on pseudo-relevance feedback, are effective in addressing this problem. The main idea is to augment the user's query with terms that appear in the initial top k retrieved documents. In this section, we extend this idea to consider the temporal dimension in the term expansion process.

In standard formulations of pseudo-relevance feedback, the timestamp of a document is not considered in identifying expansion terms—yet we know from Figure 1.1 that relevant documents are bursty and usually occur in temporal clusters, and that this signal should be incorporated into the relevance feedback model. The main insight of this work is that term expansions should be biased to draw from documents that occur in the bursty temporal clusters. This is formally captured by a continuous hidden Markov model (cHMM), in which the temporal distribution of documents (not necessarily relevant) is represented by a sequence of hidden states; the probability of generating a particular number of documents from each state follows a Gaussian distribution. I present the derivation of an EM algorithm to estimate the parameters of such a cHMM. Given a query, I first perform an initial retrieval, estimate the parameters for a cHMM that best explains the observed distribution of retrieved documents, and then use Viterbi decoding to compute the most likely state sequence. In identifying term expansions, only documents from bursty states are selected.

3.4.1 Temporal Modeling via Continuous Hidden Markov Model

Let's begin with the standard definition of an HMM for modeling a discrete observation sequence O of length T with a fixed number of hidden states. An HMM is parameterized by (A, B, π), where A is the transition matrix with A_{ij} denoting the transition probability from state i to state j at each time step, B is the emission matrix with each $B_i(O)$ denoting the probability of generating observation O from state i, and π is the initial state distribution vector.

My approach is a variant of classic HMMs. In classic HMMs each observation is a discrete symbol drawn from a finite alphabet, while in my case the observation is an integer that denotes the document count at time interval t. That is, I assume the probability of generating an observation count O_t in state i follows a Gaussian distribution:

$$B_i(O_t) = P(O_t|q_t = i) \sim N(u_i, \sigma_i)$$

The underlying states in the cHMM capture the burstiness of tweets during a particular time interval. A bursty state might correspond to a time when there are lots of users postings tweets (for example, when something newsworthy is taking place). A quiet state would correspond to times when nothing interesting is happening. In

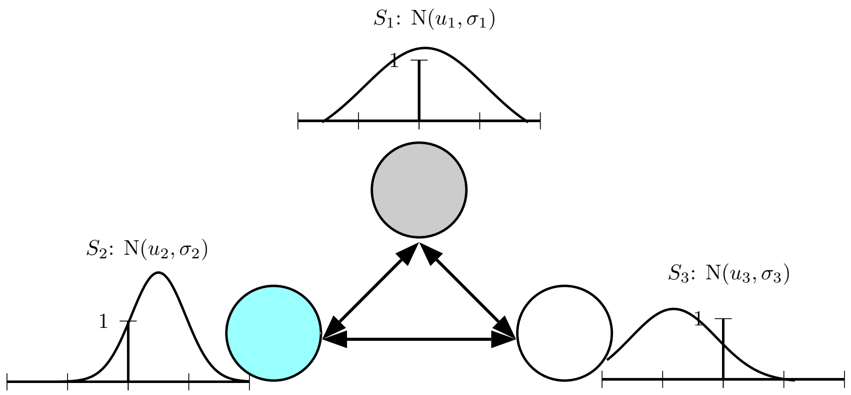

S_1: N(u_1, σ_1)
1
S_2: N(u_2, σ_2)
1
S_3: N(u_3, σ_3)
1

Figure 3.2: An illustration of a three-state cHMM. Each circle represents a state and arrows represent transitions. The Gaussians represent emissions (count of documents) from each state.

my current implementation, the cHMM uses three hidden states, but the model can be extended to capture arbitrarily many gradations of burstiness. The state transitions in the cHMM model sequential dependencies in these states—for example, a burst "dies down" when a newsworthy event passes. In each state, the mean u controls the "intensity" of the burst (i.e., how many documents are generated), and σ controls variations in different instances of the same state.

Figure 3.2 shows the three-state cHMM in our current implementation: circles represent states and arrows represent transitions. The blue circle denotes a "bursty" state as it has the largest mean, while the white circle can be interpreted as an "inactive" state since it has the smallest mean; the gray circle might be interpreted as an intermediate state.

Thus, the cHMM model is parameterized as $\lambda = (A, u, \sigma, \pi)$. Given a sequence of observations (document counts within a fixed time window), we can derive an EM algorithm to estimate the parameters iteratively.

In the E-step, the expectation of the complete-data log-likelihood $\log P(O, q|\lambda)$, namely the Q *function*, is:

$$Q(\lambda, \lambda') \propto \sum_{q} \log P(O, q|\lambda) P(O, q|\lambda') \tag{3.18}$$

where λ' represents estimates of parameters in the previous iteration that are known in the calculation and λ represents unknown parameters that we are trying to estimate for maximizing the Q function.

From the independence assumptions of HMMs (namely, that the observation O_t is only dependent on state q_t; state q_t is only dependent on the previous state q_{t-1}), we can compute the joint probability $P(O, q|\lambda)$ as follows:

$$\begin{aligned} P(O, q|\lambda) &= P(q|\lambda) P(O|q, \lambda) \\ &= \pi_{q_1} \prod_{t=2}^{T} A_{q_{t-1} q_t} \prod_{t=1}^{T} B_{q_t}(O_t) \end{aligned} \tag{3.19}$$

Substituting $P(O, q|\lambda)$ in Equation (3.19) into Equation (3.18):

$$\begin{aligned} Q(\lambda, \lambda') = &\sum_{q} \log \pi_{q_1} P(O, q|\lambda') \\ &+ \sum_{q} \left(\sum_{t=2}^{T} \log A_{q_{t-1} q_t} \right) P(O, q|\lambda') \\ &+ \sum_{q} \left(\sum_{t=1}^{T} \log B_{q_t}(O_t) \right) P(O, q|\lambda') \end{aligned} \tag{3.20}$$

We have broken the overall objective into three independent parts that we can optimize individually in the M-step. Since the optimization shares similarities with discrete HMMs (I recommend the tutorial by Bilmes [116] for more details), I skip the detailed derivations here and provide the final solutions as follows:

$$\pi_i = \frac{P(O, q_1 = i|\lambda')}{P(O|\lambda')} \tag{3.21}$$

$$A_{ij} = \frac{\sum_{t=2}^{T} P(O, q_{t-1} = i, q_t = j)}{\sum_{t=2}^{T} P(O, q_{t-1} = i|\lambda')} \tag{3.22}$$

$$u_i = \frac{\sum_{t=1}^{T} O_t \cdot P(O, q_t = i|\lambda')}{\sum_{t=1}^{T} P(O, q_t = i|\lambda')} \tag{3.23}$$

$$\sigma_i^2 = \frac{\sum_{t=1}^{T} (O_t - u_i)^2 \cdot P(O, q_t = i|\lambda')}{\sum_{t=1}^{T} P(O, q_t = i|\lambda')} \tag{3.24}$$

As with any EM algorithm, we iteratively update the parameters using above derivations until convergence. After arriving at the final parameter estimates $\lambda_f(A, u, \sigma, \pi)$, we can then use the Viterbi algorithm to find the sequence of states q_{opt} that maximizes $P(O|\lambda_f)$. Expansion terms are then computed from this state sequence, explained next.

3.4.2 Temporal Query Expansion

Given a query Q consisting of n query terms $\{t_1, t_2, ...t_n\}$, I first use the above continuous hidden Markov model to find the state sequence that best describes the temporal distribution of the top k documents collected by an initial retrieval. I consider the state with the largest mean as the bursty state, and only select documents whose timestamps fall in the bursty state for query expansion. For convenience, I call these documents *bursty documents*. I then estimate a relevance model $P(w|R)$ [6] as follows:

$$P(w|R) = \sum_{D \in C} P(D)P(w|D) \prod_{i=1}^{n} P(t_i|D) \tag{3.25}$$

where C is the set of bursty documents. I assume uniform priors $P(D)$, so the relevance model is simply a weighted average of the terms in the documents, where the weights are the query likelihood scores.

Finally, just as in RM3 [6], I interpolate the estimated relevance model with the original query model:

$$P'(w|R) = \alpha \cdot P(w|R) + (1 - \alpha) \cdot P(w|Q) \tag{3.26}$$

The interpolation parameter α is set to 0.5 by default. Following common parameter settings, I estimated the relevance models from $k = 50$ pseudo-relevant documents and selected $m = 20$ feedback terms. The evaluation of cHMM model is presented in Chapter 3.6.2.

3.5 Twitter Datasets

In this section, I describe the four Twitter test collections from the TREC Microblog Tracks in 2011, 2012, 2013, and 2014, which are used across Chapter 3, Chapter 4 and Chapter 5 for all the experiments. Note that some experiments only use a subset of the complete dataset. The statistics of the four datasets are shown in Table 3.1. Each dataset contains about 50 queries. I use the open-source implementations of tweet search provided by the TREC Microblog API[2] to retrieve up to 1000 tweets per query using query likelihood (QL) method. This helps us rule out the effects of different preprocessing strategies in collection preparation (i.e., tokenization, stemming).

The underlying document collection for the TREC Microblog 2011 and 2012 topic sets is the Tweets2011 collection, which consists of an approximately 1% sample of tweets from January 23, 2011 to February 7, 2011 (inclusive), totaling approximately 16M tweets. The underlying collection for TREC Microblog 2013 and 2014 topic sets is the Tweets2013 collection, which consists of approximately 243M tweets crawled from Twitter's public sample stream between February 1 and March 31, 2013 (inclusive).

Following standard experimental procedures, the proposed models across Chapter 3 to Chapter 5 are evaluated in a reranking task, using as input the top 1000 retrieved documents (tweets) from a bag-of-words retrieval QL ranking. I use the Stanford Tokenizer tool[3] to divide the retrieved tweets into token sequences to serve as model input. Non-ASCII characters are removed and no stemming is performed. The relevance judgments are made on a three-point scale ("not relevant", "relevant", "highly relevant"), and I treat both higher grades as relevant, also per Ounis et al. [117]. Retweets are removed in the final results, as the track guidelines consider

[2] `https://github.com/lintool/twitter-tools`

[3] `https://nlp.stanford.edu/software/tokenizer.shtml`

Test Set	2011	2012	2013	2014
# of query topics	49	60	60	55
# of query-doc pairs	39,780	49,879	46,192	41,579
# of relevant docs	1,940	4,298	3,405	6,812
# of unique words	21649	27470	24,546	22099
# of unique OOV words	13067	17190	15724	14331
# of URLs	20351	25405	23100	20885
# of hashtags	6784	8019	7869	7346

Table 3.1: Statistics of the TREC Microblog Track datasets

them not relevant.

In addition, I count the number of words in each dataset that doesn't appear in the vocabulary of the well-known word2vec embeddings [92]. This can be considered as a measure of language informality for the datasets. As we can see, more than 50% of words (OOV words) are not found in the word2vec vocabulary across all datasets, suggesting tweets are much more informal than web documents and other language tasks. I also collect the hashtags and URLs contained in tweets for future reference. Since most URLs in tweet contents are masked and shortened, for example, `http://zdxabf`, I recover the original URL addresses from redirection. The recovered URLs are truncated to a maximum of 120 characters. The four datasets used in this dissertation are publicly available.[4]

[4]`https://github.com/Jeffyrao/TREC-Microblog-Datasets`

3.6 Evaluation

3.6.1 Evaluations of Neural Temporal Framework

In this section, I present the evaluation of my proposed neural framework in Chapter 3.3 for integrating lexical and temporal evidences. The model is compared to competitive lexical and temporal baselines. The lexical baselines include query likelihood (QL), DSSM [11], SM-CNN [31], and MP-CNN [30]. The temporal baselines include the kernel density estimation method [16] with four weighting schemes: uniform-based, score-based, rank-based, and oracle. The first three are based on pseudo-feedback because they do not rely on user relevance judgments in the initial retrieved hits, while the oracle method requires explicit relevance judgements. The oracle, naturally, is not realistic, but is nevertheless useful to illustrate upper bound effectiveness. The experiments are trained on TREC Microblog 2011 topic set, and evaluated on the 2012 topic set in terms of mean average precision (MAP), precision at 15, 30, and 100, denoted as P15, P30, P100, respectively.

3.6.1.1 Implementation Details

I used existing 300-dimensional GloVe [118] word embeddings to encode each word, which was trained on 840 billion tokens and freely available. The vocabulary size of the datasets is 90.3K, with around 37% words not found in the GloVe word embeddings. Unknown words were randomly initialized with values uniformly sampled from $[-0.05, 0.05]$. During training, I used stochastic gradient descent together with RMS-PROP to iteratively update the model. The output size of the BiLSTM layer is 400 and the hidden layer size is 150. The learning rate was initially set to 0.001, and then decreased by a factor of three when the development set loss stopped decreasing for three epochs. The maximum number of training epochs was 25.

3.6.1.2 Experimental Results

Table 3.2 shows the experimental results, with each row representing an experimental condition (numbered for convenience). For each method, I performed significance testing against the lexical baseline (QL) and the best-performing temporal KDE model (rank-based). In addition, I tested the significance of differences between each pair of lexical-only model vs. lexical + temporal model. In all cases, I used Fisher's two-sided, paired randomization test [1]. Superscripts indicate the row indexes for which the metric difference is statistically significant ($p < 0.05$).

From the block in Table 3.2 labeled "Temporal Baselines", we see that the KDE approaches (with the exception of the oracle condition) yield limited improvements over the QL baseline.[5] Looking at the block of Table 3.2 labeled "Neural Ranking Approaches", we find that the SM-CNN model and DSSM do not appear to be as effective as the multi-perspective CNN; in particular, the first two models actually perform worse than the simple QL baseline.

In Table 3.2, under "Neural Ranking + Temporal Modeling", I report results

[5]These results are consistent with my previous results reported in [33]; although those experiments affirmed the overall effectiveness of the KDE techniques, results from individual configurations (such as a particular train/test split) may not yield significant improvements.

ID	Method		P15	P30	P100	MAP
1	Query Likelihood (QL) [52]		0.381	0.329	0.234	0.200
	Temporal Baselines					
2	KDE [16]	(uniform)	0.366	0.326	0.243	0.203
3		(score-based)	0.383	0.334	0.244	0.203
4		(rank-based)	0.387	0.337	0.244	0.202
5		(oracle)	**0.409**^{1,4}	**0.383**^{1,4}	**0.262**^{1,4}	**0.228**^{1,4}
	Neural Embedding Approaches					
6	L2R + Embedding		0.358	0.323	0.249	**0.219**^{1,4}
7	SM-CNN [31]		0.203	0.188	0.170	0.116
8	DSSM [11]		0.187	0.168	0.153	0.102
9	Multi-Perspective CNN [30]		**0.401**^{1}	**0.356**^{1}	**0.252**^{1}	0.197
	Neural Embedding + Temporal Ranking					
10	L2R + Embedding + Temporal		0.337	0.307	0.238	**0.211**^{1,4}
11	SM-CNN [31] + Temporal		0.222	0.196	0.169	0.116
12	Multi-Perspective CNN [30] + Temporal		**0.418**^{1,4,9}	**0.366**^{1,4}	**0.257**^{1,4,9}	0.203^{1,9}

Table 3.2: Results from TREC Microblog 2011/12 topic sets. The TREC 2011 topic set was used to train the models, and 2012 topic set was used for evaluation. Superscripts indicate the row indexes from which the metric difference is statistically significant ($p < 0.05$) using Fisher's two-sided, paired randomization test [1].

from combining the SM-CNN model and the multi-perspective CNN with the BiLSTM temporal model. In the first case, the improvement is minor over the SM-CNN model alone, but with the multi-perspective CNN, the addition of a temporal layer yields significant improvements over the multi-perspective CNN alone (condition 8) and also rank-based KDE (condition 4). It's also worth noting that the multi-perspective CNN + BiLSTM model approaches the effectiveness of the oracle KDE condition (and in the case of P15, exceeds it, albeit not significantly). This suggests that neural networks offer an expressive framework for integrating lexical and temporal signals, potentially beyond what is available to non-parametric density estimation techniques alone, even with oracle input.

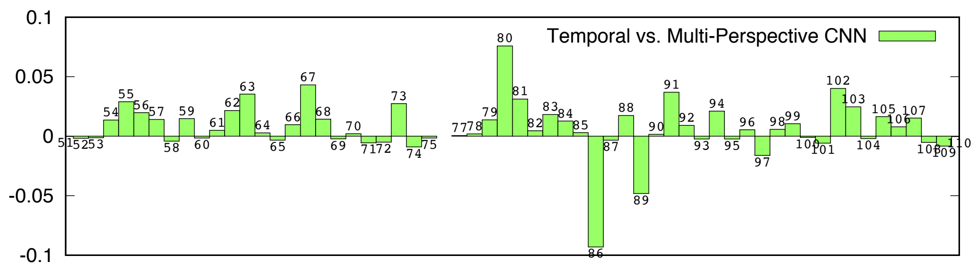


Figure 3.3: Per-topic improvement on MAP metric of the temporal ranking model vs. the multi-perspective CNN model.

Label	QL Score	MP-CNN Pred	Temporal Pred	Tweet Content
R	5.14	0.03	0.16	immigration probe of chipotle widens (reuters): reuters - upscale burrito chain chipotl @url source : yahoo news
R	5.14	0.10	0.34	now they are messing with chipotle i hate the gop mt @breakingnews: chipotle told to expect new immigration inspections
I	4.46	0.01	0.01	about to get my chipotle #siceeee
I	5.29	0.01	0.04	wowtip raid rx: delivering and receiving healer feedback: every week raid rx will help you quarterback your he @url
R	5.14	0.05	0.19	chipotle faces ice inspections in two more states (reuters): reuters - chipotle mexican grill inc has re @url
I	4.46	0.00	0.00	chipotle then home

Table 3.3: Relevance scores computed by QL method, the multi-perspective CNN and its temporal variant for sample tweets of query MB080 "chipotle raid". **R** stands for relevant and **I** stands for irrelevant. The sample tweets are ordered by their posted timestamps.

3.6.1.3 Error Analysis

To further gain insights of how the temporal modeling helps, I drew a figure (Figure 3.3) showing per-topic improvements on metric MAP of my temporal method versus its base model (the multi-perspective model). From the figure, we can see the temporal ranking model wins against its base model in the majority of topics. Except topic 86 and 89, all other bad-performing topics have very few degradation in performance.

To understand the inner workings of the temporal model, I selected some

Label	QL score	MP-CNN Pred	Temporal Pred	Tweet Content
R	13.6	0.46	0.65	mp calls for change in the law after 'intrusive' coverage of joanna yeates case @url #bristol
I	3.28	0.02	0.15	#nw murder was the case - snoop dogg (mtv jams)
I	3.28	0.00	0.02	@wandfc can i get a rt for my 41st birthday from my favourite axe murderer
I	3.28	0.17	0.55	"death proof" campy murder retro styli tarintino film rose mcgowen five stars dig in
I	3.28	0.15	0.48	the pain is brutally murdering me
I	3.28	0.01	0.06	sisters and brothers in solidarity - memorial march for missing and murdered native women @url

Table 3.4: Relevance scores computed by the QL method, the multi-perspective CNN and its temporal variant for sample tweets of query MB086 "joanna yeates murder". **R** stands for relevant and **I** stands for irrelevant.

sample tweets for the best-performing topic MB080 "chipotle raid" and the worst-performing topic MB086 "joanna yeates murder". In Table 3.3, I show 6 sample tweets for topic MB080 "chipotle raid" and the prediction scores generated by the QL method, the base multi-perspective CNN, and the temporal model. These tweets are ordered by their posted timestamps. Basically, topic MB080 is looking for tweets discussing about the news that the Mexican chain company, Chipotle, fired hundreds of employees because of an immigration raid. First, we observe that the basic multi-perspective CNN model is able to give relatively higher scores to the relevant tweets, reflecting a more reliable ranked list compared to the QL baseline. In comparison, the temporal model is able to exploit sequential dependences between neighboring tweets to better discriminate those relevant and irrelevant tweets by having a much larger divergence between the prediction scores of different classes.

In contrast, from Table 3.4, we can see the predictions of the basic multi-perspective CNN model is not that accurate, with the fourth and fifth irrelevant tweets both assigned high similarity scores. But taking a closer look at the contents of the fourth and fifth tweets, I find the main reason of their high similarity scores is because they contain many semantic-related terms to the query term “murder”, like “death”, “proof” and “pain”. However, without explicit judgements, the temporal ranking model is confused by those high scores generated by its base model, eventually boosting the similarity scores of irrelevant documents and hurting effectiveness. This confirms my previous finding that the temporal ranking model requires a high-quality neural model for generating robust document embeddings.

3.6.2 Evaluation of Temporal Query Expansion

In this section, I evaluate the cHMM model proposed in Chapter 3.4 on TREC 2011 and 2012 topic sets. The experimental procedure is as follows: I first performed initial retrieval using query-likelihood to gather ranked lists of tweets from the corpus. I then trained the cHMM model on the top 50 tweets for each topic, with three states and the number of time intervals T set to 30. After the cHMM parameters have been estimated via EM, I apply Viterbi decoding to extract the most likely state sequence, which is then used for temporal query expansion. Note that this experimental procedure does not require a training/test split of the topics.

The cHMM temporal pseudo-feedback technique is compared against the RM3 pseudo-feedback technique [6,119] as a baseline. I also implemented the KDE variant of RM3 [16,33], which includes four different ways to estimate feedback parameters: uniform, score-based, rank-based, and oracle. The first three are based on pseudo-feedback and they do not rely on user relevance judgments in the initial retrieved hits

Method	P5	P15	P30	MAP
QL	0.465	0.411	0.354	0.268
RM3	0.500	0.433	0.378	0.302
RM3 + KDE (score)	0.494	0.436	0.379	0.300
RM3 + KDE (rank)	0.490	0.425	0.376	0.292
RM3 + KDE (oracle)	$0.548^{\bullet}$	$0.492^{\bullet}$	$0.422^{\bullet}$	$0.319^{\bullet}$
cHMM	$\mathbf{0.528}^{\bullet}$	$\mathbf{0.444}^{\bullet}$	$\mathbf{0.391}^{\circ}$	$\mathbf{0.310}^{\circ}$

Table 3.5: Experimental results comparing the effectiveness of cHMMs against RM3 and KDE variants.

(which is the same as with RM3 and cHMM), while the oracle method demonstrates the upper bound as it requires explicit relevance judgments. Here, I include results for the score-based, rank-based, and oracle conditions. I follow the same parameter tuning procedure in Rao et al. [33], where the parameters were learned using test data from TREC 2013 and 2014 topics. For completeness, I show the results of the initial query-likelihood (QL) retrieval without any feedback (this, of course, is a weak condition to compare against).

Experimental results are reported in Table 3.5. The symbols $\circ$ and $\bullet$ indicate that differences with respect to the RM3 baseline are statistically significant at $p < 0.10$ and $p < 0.05$ based on Fisher's two-sided, paired randomization test [120], respectively. We observe that QL is relatively ineffective as all other models outperform it by a large margin (all differences are statistically significant at $p < 0.01$). This replicates the robust finding that query expansion is effective for searching tweets.

Consistent with the findings in Rao et al. [33], the KDE (score) and KDE (rank) approaches do not improve upon the effectiveness of RM3 by itself. However, the cHMM approach significantly outperforms RM3, confirming my initial intuitions—we obtain higher-quality expansion terms from bursty documents, and that bursty states can be captured with my cHMM. The results of the KDE (oracle) condition are not surprising, since it exploits users' explicit relevance feedback. This condition can be viewed as an upper bound on how much temporal signal can be extracted to improve relevance ranking (at least with this broad class of techniques)—and results show that my cHMM achieves effectiveness that is pretty close to this upper bound.

As a specific example of how the cHMM helps, I took a closer look at topic 14 "release of The Rite", which achieves an improvement of 0.22 (MAP) and 0.57 (P30) against the RM3 baseline. I visualized the estimated cHMM state sequence from day 6 to day 1 in Figure 3.4. As there are too many states to show if we follow the setting of $T = 30$ in the experiments above, I reduced the number of states to one per day for illustrative purposes. The blue circle denotes a bursty state, the gray circles denote an intermediate (less bursty) state, and the white circle denotes an inactive state. As we can see, the bursty state reflects the cluster of documents at day 3 in the distribution of relevant documents (topic 14 in Figure 1.1). From day 6 to day 1, the inferred states reflect the density of the documents along the timeline. Overall, this example suggests that the cHMM is able to capture sequential dependencies in the temporal distribution of relevance, which is essential for identifying those bursty and expressive terms for expansion.

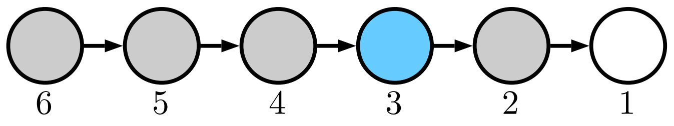


Figure 3.4: State evolution of topic 14 "release of The Rite" from day 6 to day 1 (each circle = one day). The blue circle represents a bursty state, the gray circles represent an intermediate state, and the white circle represents an inactive state.

3.7 Conclusion

To conclude, I describe two ways to model the pseudo trend for improving relevance ranking. I first introduce a unified neural framework to model the temporal distribution of relevant documents for reranking (in Chapter 3.3), Next I present a continuous hidden Markov model for selecting the more informative terms in bursty states for query expansion (in Chapter 3.4). Extensive experiments on TREC Microblog collections demonstrate the state-of-the-art effectivenesses of my approaches. Further ablation studies and error analysis show the inner workings of my temporal modeling approaches.

Chapter 4: Temporal Modeling of Query Trends for Tweet Search

4.1 Introduction

In the previous chapter, I introduce pseudo trend techniques that estimate the distribution of relevant documents using distribution of document timestamps from the results of an initial query [16]. In this chapter, I take a different approach to estimate the distribution of relevant documents: instead of relying on the results of an initial query, I attempt to exploit temporal signals embedded in the distribution of the query terms themselves. I call these *query trends*, which are generalizations of collection term statistics (of query unigrams and bigrams) in the temporal dimension. Specifically, we can keep track of the number of occurrences of query terms across a moving window over the document collection.

Consider an example that illustrates this intuition: the distribution of relevant documents (i.e., from human judgments) for topic MB127 ("hagel nomination filibustered") from the TREC 2013 Microblog Track is shown on the top in Figure 4.1. The x axis denotes a timeline, with units in days anchored at the query time on the right edge. Of course, this distribution is not known at query time—it is the target of our prediction. The remaining rows in Figure 4.1 show *query trends*, the distribution of query terms in the collection across time, for the unigrams "filibus-

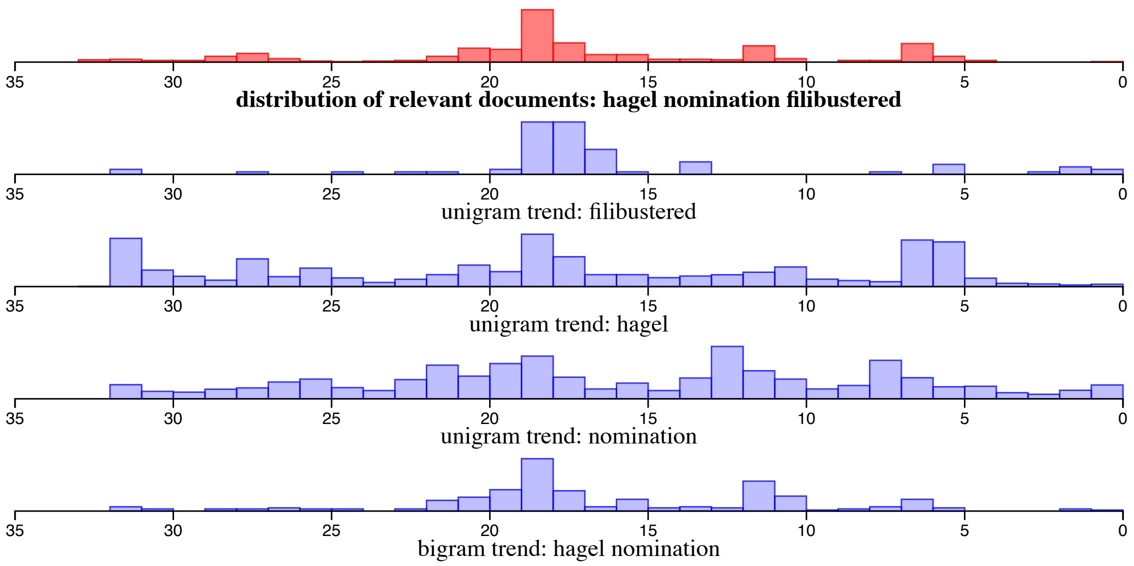


Figure 4.1: The temporal distribution of relevant documents (top row, in red) and unigram/bigram query trends (remaining rows, in blue) for MB127 ("hagel nomination filibustered") from the TREC 2013 Microblog Track. Informally, the problem can be characterized as using the blue distributions to predict the red distribution.

tered", "hagel", "nomination", and the bigram "hagel nomination". Informally, the problem can be characterized as using query trends to predict the distribution of relevant documents (i.e., the top row in Figure 4.1).

From this example, it is apparent that there are correlations between query trends and the distribution of relevant documents. Furthermore, a key advantage of my approach over previous pseudo trend methods is that it eliminates the need for an initial retrieval, since temporal term statistics can be preprocessed and stored offline. This means query trend approaches can be substantially faster than pseudo trend methods. However, the estimation of query trends requires fast access to the term statistics within a particular time window, what I reframe as *term statistics time series*. Such data could be huge in a large document collection—essentially the cross product of the vocabulary and the number of time intervals—but are also sparse, which makes them amenable to compression. Naturally, we would like to achieve as much compression as possible to minimize the storage requirements, but this needs to be balanced with decoding latencies, as the two desiderata are often intension.[1]

I first explore different algorithms for compressing and decoding term statistics time series. I begin with a number of well-known integer compression techniques and propose a novel approach based on Huffman codes over blocks of term counts. My Huffman-based techniques are able to substantially reduce storage requirements compared to state-of-the-art compression techniques while still maintaining good decoding performance. This provides us an opportunity to model query trends in a real time manner for estimating the distribution of relevant documents, which can be substantially faster than previous approaches that require an initial retrieval.

Then I explore two different approaches to exploiting query trends:

- A linear ranking model that combines features based on the temporal collection statistics of query unigrams and bigrams, their entropies, other related signals.
- A regression-based method that attempts to directly predict the distribution of relevant documents from unigram and bigram query trends.

These two approaches are further combined in an ensemble model, which additionally includes features derived from previous pseudo trend work based on kernel density estimation.

My contributions can be summarized as below:

[1]We set aside compression speed since we are working with retrospective collections.

- I perform an empirical comparison of compression techniques for term statistics time series. I begin with a number of well-known integer compression techniques and build toward a novel approach based on Huffman codes over blocks of term counts. I show that Huffman-based techniques are able to substantially reduce storage requirements compared to state-of-the-art compression techniques while still maintaining good decoding performance. My contribution enables retrieval systems to load large amounts of time series data into memory and access term statistics with low latency.

- I explore the temporal collection statistics of query terms (what I call query trends) for temporal ranking. To my knowledge, my focus on such query term statistics is novel. Experimental evaluations on standard tweet test collections show that my proposed methods are significantly more effective than competitive baselines. Furthermore, detailed studies of different feature combinations show the extent to which different types of temporal signals impact retrieval effectiveness.

This chapter is organized as follows: I first introduce my compression techniques for term statistics time series and their evaluations in Chapter 4.2, and present my approaches on temporal modeling of query trends in Chapter 4.3, followed by the experiments in Chapter 4.4. I conclude this chapter in Chapter 4.5.

4.2 Compressing and Decoding Term Statistics Time Series

Unlike the pseudo trend techniques that requires an initial retrieval stage – after a list of documents has been returned and gathered for a particular query, my approaches for estimating the query trends require real-time access to the temporal distribution of query term statistics, what I reframe as *term statistics time series*. However, such data could be huge in a large document collection, which makes them amenable to compression.

4.2.1 Compression Methods

As a start, I adopt the standard definition of a time series as a finite sequence of n real numbers, typically generated by some underlying process for a duration of n time units: $x = \{x_0, x_1, x_2, ..., x_n\}$, where each x_n corresponds to the value of some attribute at a point in time. In my case, these time series data correspond to counts on a stream of timestamped documents (tweets in this case) at fixed intervals (e.g., hourly). To be precise, these term statistics represent collection frequencies of unigrams and bigrams from a "temporal slice" of the document collection consisting of documents whose timestamps fall within the interval.

In this work, I assume that counts are aggregated at five minute intervals, so each unigram or bigram is associated with $24 \times 60/5 = 288$ values per day. Previous work [121] suggests that smaller windows are not necessary for most applications, and coarser-grained statistics can always be derived via aggregation.

I compared five basic integer compression techniques: variable-byte encoding (VB) [122], Simple16 [123], PForDelta (P4D) [124], discrete wavelet transform (DWT) with Haar wavelets, and variants of Huffman codes [125]. The first three are commonly used in IR applications, and therefore I simply refer readers to previous papers for more details. I discuss the last two in more detail.

Discrete Wavelet Transform (DWT): The discrete wavelet transform enables time-frequency localization to capture both frequency information and when (in time) those frequencies are observed. In this work, I use Haar wavelets. To illustrate how DWT with Haar wavelets work, let's start with a simple example. Suppose we have a time series with four values: $X = \{7, 9, 5, 3\}$. We first perform pairwise averaging to obtain a lower resolution signal with the values: $\{8, 4\}$. The first value is obtained by averaging $\{7, 9\}$ and the second by averaging $\{5, 3\}$. To account for information lost in the averaging, we store detail coefficients equal to pairwise differences of $\{7, 9\}$ and $\{5, 3\}$, divided by two. This yields $\{-1, 1\}$, which allows us to reconstruct the original signal perfectly. Assuming a signal with 2^n values, we can recursively apply this transformation until we end up with an average of all values. The final representation of the signal is the final average and all the detail coefficients. This transformation potentially yields a more compact representation since the detail coefficients are often smaller than the original values. I further compress the coefficients using either variable-byte encoding or PForDelta. Since the coefficients may be negative, we need to store the signs (in a separate bit array).

Huffman Coding: A nice property of Huffman coding [125] is that it can find the optimal prefix code for each symbol when the frequency information of all symbols are given. In my case, given a list of counts, we first partition the list into several *blocks*, with each block consisting of eight consecutive integers. After we calculate the frequency counts of all blocks, we are able to construct a Huffman tree over the blocks and obtain a code for each block. We then concatenate the binary Huffman codes of all blocks and convert this long binary representation into a sequence of 32-bit integers. Finally, we can apply any compression method on top of these integer sequences. To decode, we first decompress the integer array into its binary representation. Then, this binary code is checked bit by bit to determine the boundaries of the original Huffman codes. Once the boundary positions are obtained, we can recover the original integer counts by looking up the Huffman code mapping. The decoding time is linear with respect to the length of Huffman codes after concatenation.

Beyond integer compression techniques, we can exploit the sparseness of unigram counts to reduce storage for bigram counts. There is no need to store the bigram count if any unigram of that bigram has a count of zero at that specific interval. For example, suppose we have count arrays for unigram A, B and bigram AB below: A: 00300523, B: 45200103, and AB: 00100002. In this case, we only need to store the 3rd, 6th, and 8th counts for bigram AB (that is, 102), while the other counts can be dropped since at least one of its unigrams has count zero in those intervals. To keep track of these positions we allocate a bit vector 288 bits long (per day) and store this bit vector alongside the compressed data. This truncation technique saves space but at the cost of an additional step during decoding. When recovering the bigram counts, we need to consult the bit vector, which is used to pad zeros in the truncated count array accordingly.

In terms of physical storage, we maintain a global array by concatenating the compressed representations for all terms across all days. To access the compressed array for a term on a specific day, we need its offset and length in the global array. Thus, we keep a separate table of the mapping from (term id, day) to this information. Although in the experiments I assume that all data are held in main memory, my approach can be easily extended to disk-based storage.

As an alternative, instead of placing data for all unigrams and bigrams for all days together, we could partition the global array into several shards with each shard containing term statistics for a particular day. The advantage of this design is apparent: we can select which data to load into memory when the global array is larger than the amount of memory available.

4.2.2 Evaluation of Compression Methods

I evaluate the above compression techniques in terms of two metrics: size of the compressed representation and decoding latency. For the decoding latency experiments, I iterate over all unigrams or bigrams in the vocabulary, over all days, and report the average time it takes to decode counts for a single day (i.e., 288 integers). All my algorithms were implemented in Java and available open source.[2] Experiments were conducted on a server with dual Intel Xeon 4-core processors (E5620 2.4 GHz) and 128 GB RAM.

My algorithms were evaluated over the Tweets2011 and Tweets2013 collections, described in Chapter 3.5. All non-ASCII characters were removed in the

[2]`https://github.com/Jeffyrao/time-series-compression`

preprocessing phase. I set a threshold (by default, greater than one per day) to filter out all low frequency terms (including unigrams and bigrams). I extracted a total of 0.7M unigrams and 7.3M bigrams from the Tweets2011 collection; 2.3M unigrams and 23.1M bigrams from the Tweets2013 collection.

Results are shown in Table 4.1. Each row denotes a compression method. The first row "Raw" is the collection without any compression (i.e., each count is represented by a 32-bit integer). The row "VB" denotes variable-byte encoding; row "P4D" denotes PForDelta. Next comes the wavelet and Huffman-based techniques. The last row "Optimal" shows the optimal storage space with the lowest entropy to represent all Huffman blocks. Given the frequency information of all blocks, the optimal space can be computed by summing over the entropy bits consumed by each block (which is also the minimum bits to represent a block). The column "size" represents the compressed size of all data (in base two). To make comparisons fair, instead of comparing with the (uncompressed) raw data, I compared each approach against PForDelta, which is considered state of the art in information retrieval for coding sequences such as postings lists [124]. The column "percentage" shows relative size differences with respect to PForDelta. The column "time" denotes the decompression time for each count array (the integer list for one term in one day).

Results show that both Simple16 and PForDelta are effective in compressing the data. Simple16 achieves better compression, but for unigrams is slightly slower to decode. Variable-byte encoding, on the other hand, does not work particularly well: the reason is that the count arrays are aggregated over a relative small temporal window (five minutes) and therefore term counts are generally small. This enables

Tweets2011	Unigrams			Bigrams		
Method	size (MB)	percentage	time (μs)	size (MB)	percentage	time (μs)
Raw	4760			12800		
VB	1200	+442%	1.9	3200	+318%	1.1
Simple16	200	−9.50%	1.1	653	−14.6%	0.7
P4D	221	-	1.0	764	-	1.2
Wavelet+VB	1300	+488%	2.3	3700	+384%	2.3
Wavelet+P4D	352	+59.3%	2.7	978	+28.0%	2.3
Huffman	65	−70.6%	7.8	396	−48.2%	2.9
Huffman+VB	46	−79.2%	8	180	−76.4%	3.2
Optimal	32	−85.5%	-	108	−85.9%	-
Tweets2013	Unigrams			Bigrams		
Method	size (GB)	percentage	time (μs)	size (GB)	percentage	time (μs)
Raw	52.5			171.8		
VB	13.1	+446%	3.8	43.0	+347%	1.3
Simple16	2.2	−8.33%	2.2	8.3	−13.5%	0.8
P4D	2.4	-	1.9	9.6	-	1.2
Wavelet+VB	14.8	+517%	6.4	49.0	+410%	2.6
Wavelet+P4D	3.8	+58.3%	4.7	12.9	+34.4%	6.2
Huffman	0.71	−70.4%	14.7	4.9	−49.0%	6.2
Huffman+VB	0.48	−80.0%	15.6	3.0	−68.7%	6.3
Optimal	0.33	−86.2%	-	0.95	−90.1%	-

Table 4.1: Results on the Tweets2011 (top) and Tweets2013 (bottom) collections.

Simple16 and PForDelta to represent the values using very few bits. In contrast, VB cannot represent an integer using fewer than eight bits. I also noticed that the Wavelet+VB and Wavelet+P4D techniques require more space than just VB and PForDelta alone, which suggests that the wavelet transform is not effective. I believe this increase comes from: (1) DWT requires an additional array to store the sign bits of the coefficients, and (2) since the original counts are already sparse, DWT does not additionally help.

The decoding times for VB, Simple16, PForDelta, and the wavelet methods are all quite small, and it is interesting to note that decoding bigrams can be actually *faster* than decoding unigrams, which suggests that my masking mechanism is effective in reducing the length of the bigram count arrays.

Experiments show that we are able to achieve substantial compression with the Huffman-based techniques, up to 80% reduction over PForDelta. Overall, the findings hold consistently over both the Tweets2011 and Tweets2013 collections. In fact, Huffman+VB is pretty close to the entropy lower bound. Entropy coding techniques like Huffman coding prefer highly non-uniform frequency distributions, and thus are perfectly suited to the time series data. Although my Huffman+VB technique also increases decoding time, I believe that this tradeoff is worthwhile, but of course, this is application dependent. I did not try to combine Huffman coding with Simple16 or PForDelta as I found that the integer lists transformed from Huffman codes were generally composed of large values, which are not suitable for word-aligned compression methods.

4.3 Temporal Modeling of Query Trends

By now we have introduced how to compress and access term statistics time series in an efficient manner, which enables us to explore modeling the query trends for improving the ranking effectiveness. In the following, I first introduce a feature engineering method to model query trends in Chapter 4.3.1, then I present a regression way to estimate the distribution of documents directly from all query trends in Chapter 4.3.2. Finally, I combine query trend with pseudo trend features in Chapter 4.3.3.

4.3.1 Feature Engineering on Query Trends

Intuitively, we would expect to find more relevant documents in temporal intervals where the query terms are bursty. I illustrate this in Figure 4.1 for topic MB127 ("hagel nomination filibustered") from the TREC 2013 Microblog Track, as described in the introduction. The top row shows the actual distribution of relevant documents, which is the target of our prediction and of course not known at query time. The remaining rows show the query trends of the unigrams "filibustered", "hagel", "nomination", and the bigram "hagel nomination".[3] As we might expect, there are correspondences between peaks in the query trends and the actual distribution of relevant documents—for example, the few days when the unigram "filibustered" occurs most frequently are also when most of the relevant documents are clustered.

Of course, not all query trends are created equal. In the example in Figure 4.1, we see that the distribution of the unigram "nomination" is less predictive of the distribution of relevant documents. Overall, we find that less bursty terms are less useful, a notion we can formally capture by computing the entropy of the distribution. Given the counts of a particular unigram or bigram $t = \{c_1, c_2, ..., c_n\}$ across various time intervals (e.g., days), its entropy can be computed as follows:

$$\text{Entropy}(t) = -\sum_i \frac{c_i}{C} \log \frac{c_i}{C} \tag{4.1}$$

[3]The other query bigram "nomination filibustered" is ignored in this analysis because it does not occur with sufficient frequency (based on a simple threshold).

where $C = \sum_i c_i$. Lower entropy indicates a less uniform distribution and thus more bursty behavior.

From the query trends we can derive a family of features for a learning-to-rank model. There is, however, one additional complication we need to address: queries vary in length, which means that different queries have different numbers of unigram and bigram query trends. This is problematic since the linear feature-based model we use assumes a fixed number of features. I address this issue in a more principled manner in the next section, but here I introduce features based on the unigram and bigram with the lowest entropy (thus, the largest burstiness). I call these the *representative* unigram and bigram query trend, respectively.

From the basic concepts introduced above, I propose the following features:

- The relative entropy of the representative unigram. The relative entropy reflects the burstiness of a unigram query trend, computed as the absolute difference between the unigram entropy and the maximum entropy. The maximum entropy is computed by assuming a uniform distribution over term counts. Note that queries can have different timespans (because each is associated with a different query time), and thus the maximum entropy is query-dependent; computing relative entropy normalizes for the effects of different query timespans.

- The relative entropy of the representative bigram. This feature is computed in exactly the same manner as described above, except on bigram query trends.

- Estimated density at the document's timestamp from the query trend of the representative unigram. This feature is document-dependent. First, I perform kernel density estimation over the representative query unigram. Then, for the particular document that we are scoring, we compute the estimated density at the document's timestamp.

- Estimated density at the document's timestamp from the query trend of the representative bigram. This is similar to above, except with bigrams.

In Chapter 4.3.3, I detail how these features are integrated into the final ranking model.

4.3.2 Regression on Query Trends

The above feature engineering approach tries to predict the distribution of relevant documents via a single representative unigram or bigram query trend. An alternative is to integrate evidence from *all* unigram and bigram query trends. Such an approach, however, can be a double-edged sword. On the one hand, I observe that for many topics, the distribution of relevant documents has many peaks. In these cases, it is unlikely that a single unigram or bigram query trend is sufficient to reconstruct the reference distribution. Such cases would seemingly benefit from integrating multiple sources of evidence to overcome the limited signal from any individual query trend. On the other hand, I see that some query trends have low or even negative correlations with the actual distribution of relevant documents (e.g., query terms that aren't important to the information need). In these cases, the query trends merely introduce noise into the prediction. How to balance these two factors is a question I explore.

The basic idea behind my regression-based method is to predict the actual query distribution by integrating all unigram and bigram query trends. When a query arrives, we can apply the entropy computations and kernel density estimations on all query terms. Suppose we have computed an entropy of e_t and a kernel density function of f_t for each term t. We can then attempt to fit the actual density of relevant documents Y (which is obtained by KDE [16] on the distribution of relevant documents) as follows:

$$Y \approx \sum_t w_t f_t \tag{4.2}$$

where weight w_t is a function of entropy e_t and our goal is to learn this mapping function.

Note that approximating a continuous function from multiple kernel density functions is difficult, so instead I sample the distributions at fixed intervals. Now this model transforms into a non-linear regression problem. Given the unigram entropies E_u, bigram entropies E_b, unigram densities U at the sample points, and bigram densities B at the sample points, our task is to predict the densities Y at the same points. For more details about symbols used in this section, please refer to Table 4.2.

Two questions need to be answered in this non-linear regression problem. First, how to determine the importance of each term in contributing to the estimated density? Based on my observations, I find that terms with larger normalized entropies, i.e., a larger difference between its absolute entropy and the entropy of a uniform distribution, are more likely to reflect the true distribution of relevant documents.

	Description
N_q	number of queries
N_p	number of sample points per query
N	$N_q \cdot N_p$, number of sample points across all queries
N_u	max. number of unigrams per query (default 10)
N_b	max. number of bigrams per query (default 10)
Y_i	$N_p \times 1$, densities computed from relevant docs for query i
Y	$N \times 1$, concatenation of densities $(Y_1, ..., Y_i, ..., Y_{N_q})$
U	$N \times N_u$, densities computed from unigram trends
B	$N \times N_b$, densities computed from bigram trends
E_u	$N_q \times N_u$, normalized relative unigram entropies
E_b	$N_q \times N_b$, normalized relative bigram entropies
R	$N_q \times 1$, ratio of max unigram to bigram entropy
w_i^u	$N_u \times 1$, weight vector for unigrams of query i
w_i^b	$N_b \times 1$, weight vector for bigrams of query i

Table 4.2: Notation Table.

Therefore, I formulate the mapping from entropy to weights via an exponential increasing function, $w_t = \exp(\theta \cdot e_t) - 1$, where e_t is the normalized entropy of term t with its value ranging from zero to one. A term with zero normalized entropy would have zero weight, and thus can be ignored. The parameter θ controls the exponential rate. I use α for unigrams and β for bigrams as θ below.

The second question is how to differentiate contributions of unigrams from those of bigrams. For some queries, unigram query trends are more predictive, while for others, bigram trends are more predictive. How to evaluate their contributions for different queries is one key aspect of my model. To this end, for each query, I assign a weight $u_i \in [0, 1]$ to denote its unigram contribution; the corresponding bigram weight would be $1 - u_i$. I link the normalized unigram weight u_i to the entropy ratio R_i (which is the ratio of the maximum normalized unigram to bigram entropy for query i) by observing correlations between these two factors in training data. This mapping is normalized by a logistic function:

$$u_i = \text{logistic}(R_i, \gamma) = \frac{1}{1 + \exp(-\gamma R_i)} \tag{4.3}$$

where

$$R_i = \frac{\max_u E_i^u}{\max_b E_i^b} - 1 \tag{4.4}$$

and γ is a parameter to be estimated.

Intuitively, R_i greater than zero implies that the maximum normalized unigram entropy is larger than the maximum normalized bigram entropy. In this case, the logistic function would assign a unigram weight $u_i > 0.5$, and so unigrams would contribute more to the density estimate than bigrams. Finally, I desire that the integrated densities approximate the actual query density Y for each query i:

$$Y_i \approx u_i U_i w_i^u + (1 - u_i) B_i w_i^b \tag{4.5}$$

where w_i^u and w_i^b are weight vectors of unigrams and bigrams of query i, respectively. Overall, I sum up the square loss between ground truth densities Y_i and the estimated densities $\hat{Y}_i$ over all queries, plus some regularization terms. The final loss function L is formulated as follows:

$$\begin{aligned} L = & \sum_{i=1}^{N_q} \| Y_i - (u_i U_i (e^{\alpha E_i^u} - 1)^T + (1 - u_i) B_i (e^{\beta E_i^b} - 1)^T) \|^2 \\ & + \lambda(\alpha^2 + \beta^2 + \gamma^2) \end{aligned} \tag{4.6}$$

where R_i and u_i are defined above.

Note that this model has three parameters (α, β, and γ) to be estimated, which are the weights of the entropy mapping function and the logistic function. Since the loss L is differentiable with respect to the three parameters, we can optimize the parameters using gradient-based methods. By constituting the logistic function into the overall loss function L, the gradients with respect to the parameters are computed as follows:

$$
\begin{aligned}
\text{term} &= 2 \cdot \Big(Y_i - (u_i U_i (e^{\alpha E_i^u} - 1)^T + (1 - u_i) B_i (e^{\beta E_i^b} - 1)^T)\Big) \\
\frac{\partial L}{\partial \alpha} &= -\sum_{i=1}^{N_q} \big(u_i \cdot \text{term}^T \cdot U_i \cdot (E_i^u \cdot e^{\alpha E_i^u})^T\big) + 2\lambda\alpha \\
\frac{\partial L}{\partial \beta} &= -\sum_{i=1}^{N_q} \Big((1 - u_i) \cdot \text{term}^T \cdot B_i \cdot (E_i^b \cdot e^{\beta E_i^b})^T\Big) + 2\lambda\beta \\
\frac{\partial L}{\partial \gamma} &= \sum_{i=1}^{N_q} \Bigg(\text{term}^T \cdot \Big(-U_i \cdot (e^{\alpha E_i^u} - 1)^T + B_i \cdot (e^{\beta E_i^b} - 1)^T\Big) \\
&\quad \cdot R_i \text{logistic}(R_i, \gamma) \cdot (1 - \text{logistic}(R_i, \gamma))\Bigg) + 2\lambda\gamma
\end{aligned}
$$

After solving the objective, we learn two mappings: an exponential mapping from entropy to term weight $w^t = \exp(\theta e) - 1$, and a logistic mapping from ratio to unigram weight $u = \text{logistic}(R, \gamma)$. We are then able to estimate densities for queries in the test data:

$$\hat{Y}_i = u_i U_i w_i^u + (1 - u_i) B_i w_i^b \tag{4.7}$$

Finally, the estimated density $\hat{Y}_i$ serves as a feature in the final evidence combination

	Description
1	QL score
	Density estimate from:
2	KDE over initial retrieved docs (uniform)
3	KDE over initial retrieved docs (score-based)
4	KDE over initial retrieved docs (rank-based)
5	KDE over relevant docs (oracle)
Chapter 4.3.1	
6	Relative entropy of representative unigram
7	Relative entropy of representative bigram
	Density estimate from:
8	KDE of representative unigram distribution
9	KDE of representative bigram distribution
Chapter 4.3.2	
10	Density estimate from query trend regression model

Table 4.3: Summary of all features.

approach (more details below).

4.3.3 Pull Everything Together

To recap, I have introduced three families of features for modeling temporal evidence: KDE applied to initial retrieved documents [16] (Chapter 3.2), features derived from query trends (Chapter 4.3.1), and density estimates from a query trend regression model (Chapter 4.3.2). In total, we have ten features, including query-likelihood for capturing content relevance, which are summarized in Table 4.3.

Then I integrate all these features in a linear feature-based ranking model [113]. The general form of such a model, extended from Eq. (3.8), is as follows:

$$S_d = \sum_i \alpha_i \cdot F_i(d, q) \quad \text{s.t.} \sum_i \alpha_i = 1. \tag{4.8}$$

Naturally, we would like to understand the relative contributions of each type of

Method	Features
QL	1
IRD_u	1, 2
IRD_s	1, 3
IRD_r	1, 4
QT	1, 6–9
QT + IRD_r	1, 4, 6–9
Reg	1, 10
Reg + IRD_r	1, 4, 10
Oracle	1, 5

Table 4.4: Summary of different feature combinations.

feature, but it does not make sense to exhaustively explore all possible combinations. Thus, I take the middle road and explore a number of interesting feature set combinations, summarized in Table 4.4:

- Different weighting schemes for KDE applied to the initial retrieved documents. These are the same experimental conditions in Efron et al. [16] and my previous results [33]. For convenience, these models are referred to as IRD_u (uniform weights), IRD_s (score-based weights), and IRD_r (rank-based weights). My previous experiments [33] show that rank-based weights are the most effective overall, and thus for subsequent configurations I only use rank-based weights.

- Query trend features as a group (QT) and query trend features combined with KDE on the initial retrieved documents with rank-based weights (QT + IRD_r).

- Query trend regression (Reg) and query trend regression combined with KDE on the initial retrieved documents with rank-based weights (Reg + IRD_r).

Note that use of the IRD_r features requires an initial retrieval, and thus we lose the efficiency advantage of feature combinations that use only query trends.

4.4 Evaluation of Query Trend Methods

I evaluate the proposed methods on TREC Microblog 2013 and 2014 topic sets (datasets are described in Chapter 3.5), which uses the Tweets2013 collection as the underlying document corpus. In my experiments, I examined four different ways of splitting the test collections into training and test sets:

- First, I trained on odd-numbered topics from the TREC 2013 and 2014 Microblog Tracks (57 topics) and evaluated on even-numbered topics (58 topics).
- Second, I swapped the training/test splits: training on even-numbered topics and testing on odd-numbered topics.
- Third, I performed four-fold cross validation across all topics.
- Finally, I performed a series of trials in which I randomly selected half the topics for training and used the remaining for testing. Results across multiple trials are aggregated.

I used coordinate ascent in RankLib[4] to learn the parameters in Eq. (4.8), optimizing and evaluating on the same metric.

Several baselines are used as points of comparison to my proposed methods. Query likelihood (QL) [52] is used as a lexical baseline. Temporal baselines include:

- Li and Croft's recency prior method [26].
- The moving window method of Dakka et al. [61].

[4] `https://sourceforge.net/p/lemur/wiki/RankLib/`

- The kernel density estimation (KDE) methods of Efron et al. [16] with uniform weights (IRD_u), score-based weights (IRD_s), and rank-based weights (IRD_r).

In addition, I also include the KDE oracle as a reference upper bound. In this condition, I apply kernel density estimation over the distribution of the relevant documents based on human assessor judgments. This characterizes how much temporal signal can be extracted to improve relevance ranking, at least with this class of density estimation techniques.

To build the query trend features, we need to precompute collection frequencies across time windows for the entire vocabulary. I aggregated term statistics and worked with query trends at the day granularity—that is, each term's trend is represented by an integer array of size 59, where each integer denotes the collection frequency for a single day. By discarding terms with a collection frequency lower than five, I extracted a total of 2.3 million unigrams and 23.1 million bigrams from the Tweets2013 collection. I used the PForDelta encoding technique to compress the term statistics time series (in Chapter 4.2), down to a size of 0.26 GB for unigrams and 2.2 GB for bigrams. The average decoding time of the compressed term statistics is 5.1 μs per unigram and 5.8 μs per bigram on a commodity server. Due to the efficient compression, we are able to load the term statistics into memory to estimate query trend features very quickly.

ID	Method		Odd-Even AP	Odd-Even P30	Even-Odd AP	Even-Odd P30	Cross AP	Cross P30
1	Query Likelihood (QL) [52]		0.271	0.475	0.357	0.564	0.315	0.520
2	Recency prior [26]		0.277	0.499^{1}	0.359	0.574	0.313	$0.534^{1,4}$
3	Moving Window (WIN) [61]		0.283^{1}	0.487^{1}	0.358	0.567	0.319	0.527
4	KDE [16]	IRD_u	0.273	0.481	0.350	0.566	0.308	0.515
5		IRD_s	0.274	0.487^{1}	0.353	0.577^{1}	0.314	$0.530^{1,4}$
6		IRD_r	$0.288^{1,4,5}$	$0.517^{1,3\text{-}5}$	0.360	$0.588^{1\text{-}4}$	$0.327^{1,2,4,5}$	$0.552^{1\text{-}5}$
7	This work	**QT**	0.278	$0.492^{1,4}$	$0.367^{1,4,5}$	$0.587^{1\text{-}4}$	0.320	$0.530^{1,4}$
8		**Reg**	0.276	0.488^{1}	$0.366^{1,4,5}$	0.576^{1}	$0.329^{1,2,4,5}$	$0.535^{1,4}$
9		**QT-IRD$_r$**	$0.290^{1,2,4,5}$	$0.522^{1\text{-}5}$	**$0.370^{1\text{-}5}$**	**$0.598^{1\text{-}5}$**	$0.328^{1,2,4,5}$	$0.565^{1\text{-}5}$
10		**Reg-IRD$_r$**	**$0.302^{1\text{-}6}$**	**$0.535^{1\text{-}5}$**	$0.368^{1\text{-}5}$	$0.596^{1\text{-}5}$	**$0.332^{1\text{-}5}$**	**$0.566^{1\text{-}5}$**
11	Oracle		$0.314^{1\text{-}6}$	$0.536^{1\text{-}6}$	$0.382^{1\text{-}6}$	$0.636^{1\text{-}6}$	$0.349^{1\text{-}6}$	$0.586^{1\text{-}6}$

Table 4.5: Results from the TREC 2013/14 Microblog Track test collections: "Odd-Even" represents training on odd topics and testing on even topics; "Even-Odd" represents the opposite; "Cross" represents four-fold cross validation. Superscripts indicate the row indexes from which the metric differences are statistically significant ($p < 0.05$).

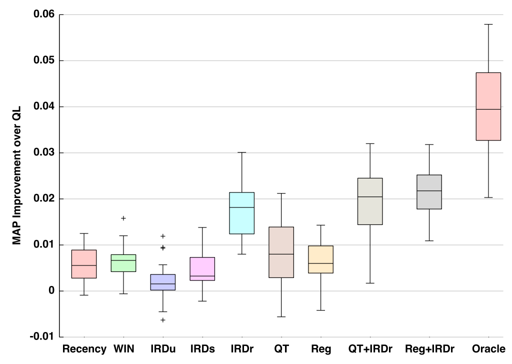
4.4.1 Effectiveness of Temporal Models

The experimental results are summarized in Table 4.5. Each row denotes an experimental condition (numbered for convenience): the third column "Odd-Even" represents training on odd-numbered topics and testing on even-numbered topics; "Even-Odd" represents the opposite; "Cross" represents four-fold cross validation. The best result for each setting is in bold. I compare each method against all lexical and temporal baselines for statistical significance using Fisher's two-sided, paired randomization test [1]. Superscripts indicate the row indexes from which the metric differences are statistically significant ($p < 0.05$).

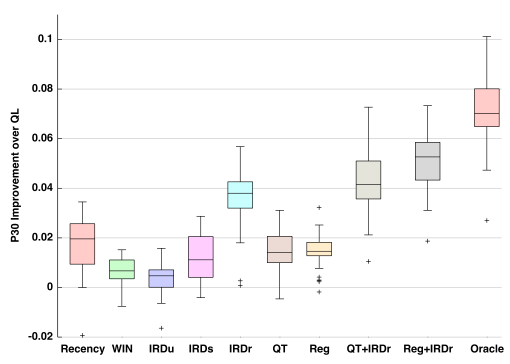
First, we observe that most temporal baselines (Recency, WIN, IRD_s, and IRD_r) outperform the lexical baseline in terms of P30, but generally not in terms of AP, suggesting that they are better suited to improving early precision. Among the temporal baselines, IRD_u performs consistently the worst and IRD_r outperforms the rest. Note that while IRD$_s$ and IRD$_r$ both place more weight on top-ranked documents, the gap in effectiveness comes from the fact that the retrieved scores of the top-ranked documents are generally quite similar. Thus, score normalization does not introduce sufficient bias to help us distinguish the high-ranking documents.

Second, we see that my query trend methods (QT and Reg) significantly outperform the lexical baselines in most conditions, suggesting that signals captured from temporal collection statistics are beneficial to relevance ranking. While these "vanilla" query trend methods alone do not significantly improve over the temporal baselines, combining them with the pseudo trend methods (as in QT+IRD$_r$ and Reg+IRD$_r$) yields a boost in effectiveness. These ensemble methods are consistently more effective than the best-performing temporal baseline IRD$_r$. They also come close to the upper bound (oracle) in some conditions, especially for P30. For the Reg+IRD$_r$ model, features 1, 4, and 10 (query likelihood, IRD$_r$, Reg features) received weights 0.84, 0.10, and 0.06 in the Odd-Even split, respectively, which shows that the different sources of temporal evidence are complementary.

In the above experiments, I noticed variance in effectiveness under different conditions, depending on how the test collections are split into training/test sets. My random split experiments were designed to factor out noise from this issue. In each trial, I trained on half of the topics (randomly selected) and evaluated on the other half. I then computed the effectiveness differences between each technique and the QL baseline. These differences, collected over 30 trials, are summarized in box-and-whiskers plots in Figure 4.2 for all temporal approaches. I show the distribution of effectiveness differences in terms of AP (left) and P30 (right). Each



(a) AP on TREC 2013/14



(b) P30 on TREC 2013/14

Figure 4.2: Box-and-whiskers plots summarizing how much each temporal model outperforms the QL baseline across 30 random trials (half for training, half for testing) on the TREC 2013/14 Microblog Track test collections.

box represents the span between the first and third quartiles, with a horizontal line at the median value. Whiskers extend from the ends of each box to the most distant point whose value lies within 1.5 times the interquartile range. Points that lie outside these limits are drawn individually. These results capture the overall effectiveness of each method, better than metrics from any single arbitrary split.

From Figure 4.2, it is clear that IRD$_r$ outperforms all baselines as well as the raw query trend approaches (QT and Reg). The ensemble approaches (QT+IRD$_r$ and Reg+IRD$_r$) yield further improvement over IRD$_r$, with Reg+IRD$_r$ coming out higher. Although I did not observe a statistically significant difference between the best ensemble method (Reg+IRD$_r$) and the best baseline (IRD$_r$) in the previous experiments, the box plots show that the effectiveness gains of Reg+IRD$_r$ are more consistent, This is especially true for P30 (right side of Figure 4.2): the median of Reg+IRD$_r$ is above 0.05 whereas IRD$_r$ has a median below 0.04. Another observation is that the bottom of the Reg+IRD$_r$ box is still above the top of the IRD$_r$ box, meaning that the top 75% of Reg+IRD$_r$ runs were better than the bottom

75% of IRD_r runs. Although it is difficult to definitively conclude statistical significance from these experiments, quantifying the variance associated with arbitrary training/test splits provides additional evidence supporting the effectiveness of my proposed methods.

4.4.2 Per-Topic Analysis

In order to gain a better understanding of how different temporal features contribute to effectiveness in temporal ranking, we performed a topic-by-topic analysis along with an in-depth examination of the various component distributions. Due to a lack of space, here we present only results comparing the best-performing ensemble model (Reg+IRD_r) against the lexical baseline QL and the temporal baseline IRD_r. In Figure 4.3, I show per-topic differences as a bar chart, measured in terms of P30 on the even topics from the TREC 2013/14 Microblog Track test collections.

From the top bar chart in Figure 4.3, we can see that the ensemble model (Reg+IRD_r) improves over the QL baseline for most of the topics; there are only a few topics where effectiveness decreases (and not by much). From the bottom bar chart, we see that Reg+IRD_r improves over IRD_r alone, which confirms that the regression model contributes additional signal over kernel density estimation alone.

In Figure 4.4, I take a closer look at the best-performing topic, MB144 "downtown abbey actor turnover". The top row shows the distribution of relevant documents (in red), the second row shows the distribution inferred from the pseudo trend using IRD_r (in green), and the remaining rows show the query trends (in

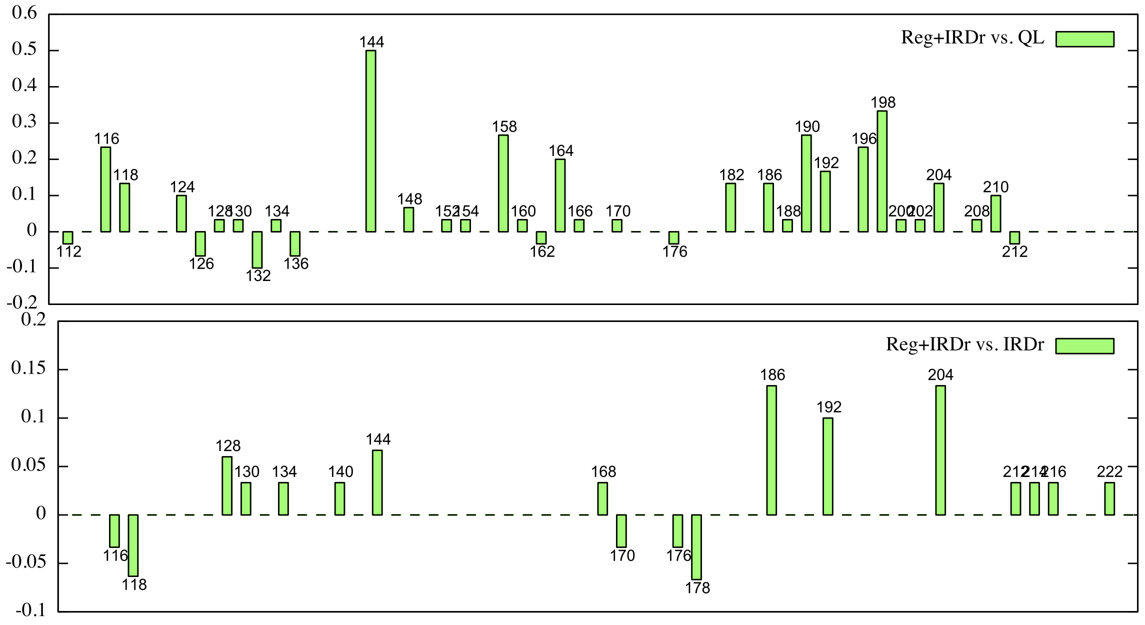


Figure 4.3: Per-topic improvements of the ensemble model Reg+IRD$_r$ compared to the QL baseline and IRD$_r$ method.

blue). Clearly, we can see that the distribution of relevant documents has two peaks, one around days 3–5 and the other around days 14–16. We can observe that the pseudo trend from IRD$_r$ is able to capture the burst of relevant documents at days 3–5. We also see a strong correlation between the query trends (unigrams "downtown", "abbey", and the bigram "downtown abbey") and the ground truth relevance distribution at days 14–16. Thus, the combination of pseudo trend and query trend features allows us to nicely recover this multimodal distribution, which is affirmed by the large improvements for this topic compared to both QL and IRD$_r$. In addition, the regression model is able to smooth out noise from non-important terms "actor" and "turnover". This observation is confirmed in many other topics, like MB192 "whooping cough epidemic" and MB204 "sotomayor, prosecutor, racial comment", where we also observe strong correlations between query trends and the ground truth relevance distributions.

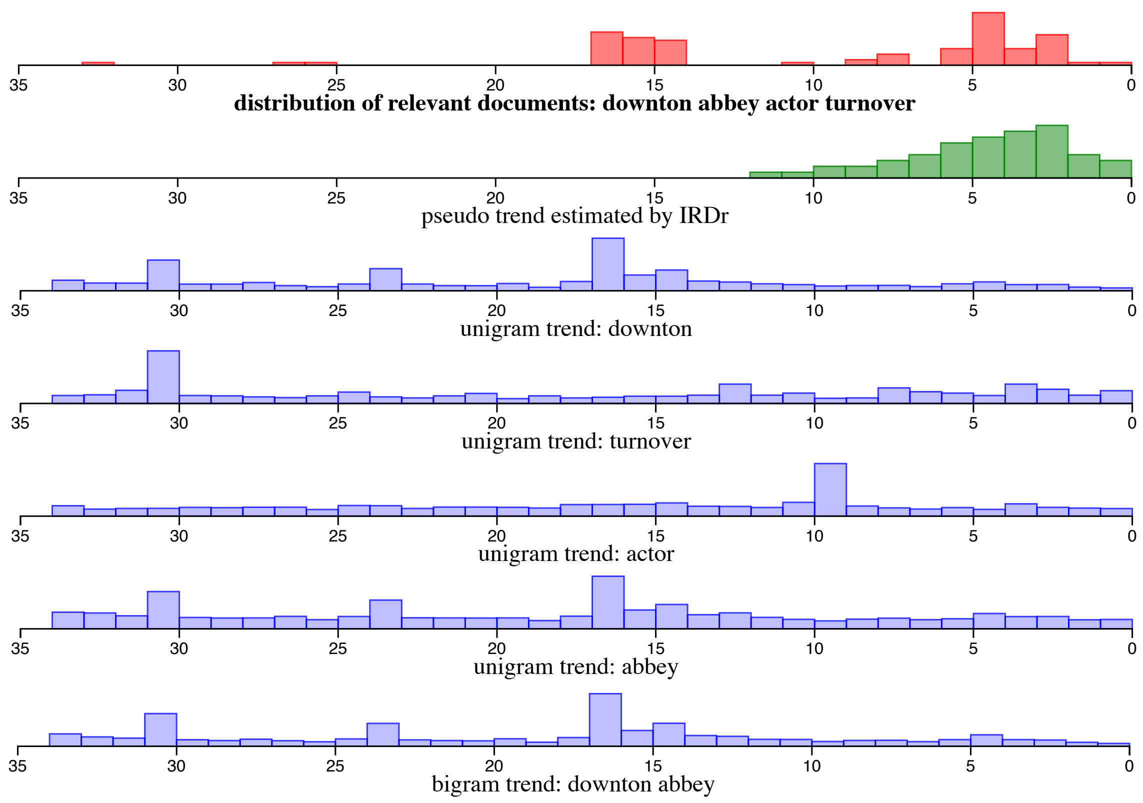


Figure 4.4: Analysis of MB144 ("downtown abbey actor turnover") from the TREC 2013 Microblog Track. Rows show: distribution of relevant documents (red), pseudo trend based on KDE (green), and query trends (blue).

I also examined topics where effectiveness decreased with respect to IRD$_r$, such as MB116 "Chinese computer attacks" and MB118 "Israel and Turkey reconcile". I found that the representative query trends (the unigram "Chinese" for MB116 and the bigram "and Turkey" for MB118) are very different from the distribution of relevant documents, and thus my methods infer an inaccurate distribution. No approach is perfect, but overall the per-topic analysis affirms the effectiveness of my query trend methods.

4.5 Conclusion

Quite obviously, the temporal distribution of relevant documents provides an important signal for temporal ranking. As an alternative to previous pseudo trend methods that analyze the results of an initial query to infer this distribution, I propose query trend methods that attempt to make predictions directly from the temporal collection statistics of query terms. Experiments show that these sources of evidence are complementary, and the regression method appears to be more effective than the feature-based approach. Although query trend methods alone, which do not require an initial retrieval, improve over a lexical baseline, combining query trends with pseudo trends yields the best results. This ensemble approach, however, *does* require an initial retrieval, which negates the performance advantages of query trend methods. Costly approaches that involve actually searching the collection appear to provide temporal signals that we currently cannot obtain from the temporal collection statistics of query terms alone.

Chapter 5: Multi-Perspective Lexical Modeling for Tweet Search

5.1 Introduction

In Chapter 3, I have shown that effective temporal models for tweet search require good representations of query–document similarities. However, existing state-of-the-art neural ranking models are insufficient for this purpose, as the experiments have verified in Chapter 3.6.1. Therefore, I explore a novel lexical modeling technique by taking advantage of recent advances in neural networks, which have achieved great success in many natural language processing (NLP) tasks, such as question answering [31, 46], paraphrase detection [80], and textual semantic similarity modeling [76]. Many of these tasks can be treated as variants of a *semantic matching* problem, where two pieces of texts are jointly modeled through distributed representations of sentences for similarity learning. Various neural network architectures, e.g., Siamese networks [115], sequence-to-sequence models [27], and attention mechanism [77], have been proposed to model the semantic similarity of a text pair using diverse modeling techniques.

On the other hand, techniques based on deep learning and neural networks offer exciting opportunities for the information retrieval community. For example, distributed word representations (e.g., word2vec [92]) provide a promising solution to overcome the vocabulary mismatch problem in ranking [126]. However, there are still fundamental challenges to be solved. Guo et al. [10] pointed out that *relevance matching*, which is the core problem in IR, has different characteristics from the *semantic matching* problem that many NLP models are designed for. In particular, exact match signals still play a critical role in ranking, more than the role of term matching in, for example, paraphrase detection. Furthermore, in document ranking there is an asymmetry between queries and documents in terms of length and the richness of signals that can be extracted; thus, symmetric models such as Siamese architectures may not be entirely appropriate. Nevertheless, significant progress has been made, and many neural ranking models have been recently proposed [11, 36, 37, 89, 91], which have been shown to be effective on *ad hoc* retrieval.

Despite much progress, it remains unclear how neural ranking models designed for "traditional" *ad hoc* retrieval tasks perform on searching social media posts such as tweets on Twitter. I identify several important differences:

- **Document length**. Social media posts are much shorter than web or newswire documents. For example, tweets are limited to 280 characters. Thus, *ad hoc* retrieval in this domain contains elements of semantic matching because queries and posts are much closer in length. In particular, neural models that rely on sentence-level or paragraph-level interactions and global matching mechanisms [37] are unlikely to be effective.

- **Informality**. Idiosyncratic conventions (e.g., hashtags), abbreviations ("Happy Birthday" as "HBD"), typos, intentional misspellings, and emojis are prevalent in social media posts. An effective ranking model should account for such language variations and term mismatches due to the informality of posts.

- **Heterogeneous relevance signals**. The nature of social media platforms drives users to be actively engaged in many real-world news and events; users frequently take advantage of URLs or hashtags to gain exposure to their posts. Such heterogeneous signals are not well exploited by existing models, which can potentially boost ranking effectiveness when modeled together with the textual content.

To this end, I present a novel neural ranking model for *ad hoc* retrieval over short social media posts that is specifically designed with the above characteristics in mind. My model, MP-HCNN (**M**ulti-**P**erspective **H**ierarchical **C**onvolutional **N**eural **N**etwork), aims to model the relevance of a social media post to a query in a multi-perspective manner, and has three key features:

1. To cope with the informality of social media and to support more robust matching, I apply word-level as well as character-level modeling, with URL-specific matching. This allows us to exploit noisy relevance signal at different granularities (Chapter 5.2.1).

2. My model consists of stacked convolutional neural network layers to capture latent semantic soft-match signals between query and post contents (tweets in our case, but we use posts for generic purpose). By gradually expanding the convolutional window in a hierarchical manner, increasingly larger contexts can be leveraged for modeling relevance, starting from character-level and word-level to phrase-level, and finally to sentence-level (Chapter 5.2.2).

3. Matching of learned representations between query and posts as well as URLs is accomplished with a pooling-based similarity measurement layer where term importance weights are injected at each convolutional layer as priors (Chapter 5.2.3).

Finally, all relevance signals are then integrated using a fully-connected layers to yield the final relevance ranking. Optionally, the neural matching score can be integrated with lexical matching via linear interpolation to further enhance effectiveness. I view my contributions in this chapter as below:

- I highlight three important characteristics of social media posts that make *ad hoc* retrieval over such collections different from searching web pages and newswire documents. Starting from these insights, I developed MP-HCNN, a novel neural ranking model specifically designed to address these characteristics. To my best knowledge, this is also the first neural ranking model developed specifically for *ad hoc* retrieval over social media posts.

- I evaluate the effectiveness of my MP-HCNN model on four Twitter benchmark collections from the TREC Microblog Tracks 2011–2014. My model is compared to learning-to-rank approaches as well as many recent state-of-the-art neural ranking models that are designed for web search and "traditional" *ad hoc* retrieval. Extensive experiments show that my model improves the state-of-the-art over previous approaches significantly. Ablation studies further confirm that these improvements come from specific components of my model designed to tackle characteristics of social media posts as identified above.

In the following, I introduce the proposed model architecture in Chapter 5.2, followed by its evaluation in Chapter 5.3 and the conclusion in Chapter 5.4.

5.2 Model Architecture

As discussed in the introduction, the proposed model, MP-HCNN (Multi-Perspective Hierarchical Convolutional Neural Network), has three key features: First, I apply word-level as well as character-level modeling on query, posts, and URLs to cope with the informality of social media posts (Chapter 5.2.1). Second, I exploit stacked convolutional layers to learn soft-match relevance at multiple granularities (Chapter 5.2.2). Finally, matches between the learned representations via pool with injected external weights are learned (Chapter 5.2.3). The overall model architecture is shown in Figure 5.1, and each of the above key features are described in detail below.

5.2.1 Multi-Perspective Input-level Modeling

A standard way for neural text processing is to take advantage of word embeddings (e.g., word2vec [92]) to encode each word. However, in the social media domain, informal post contents produce a large amount of out of vocabulary (OOV) words which can't be found in pre-trained word embeddings. The embeddings of OOV words are randomly initialized by default. In fact, I observe about 50%-60% words are OOV words in the TREC Microblog datasets (details in Table 3.1). This greatly complicates the matching process simply relying on word-level semantics,

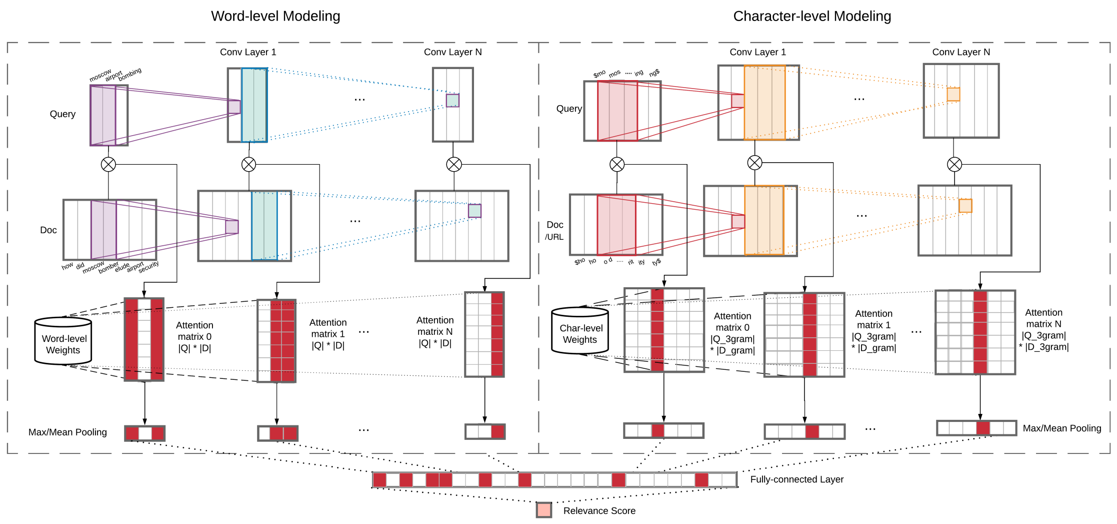

Word-level Modeling
Character-level Modeling
Conv Layer 1
Conv Layer N
Query
Doc
Doc /URL
Word-level Weights
Char-level Weights
Attention matrix 0 |Q| * |D|
Attention matrix 1 |Q| * |D|
Attention matrix N |Q| * |D|
Attention matrix 0 |Q_3gram| * |D_gram|
Attention matrix 1 |Q_3gram| * |D_gram|
Attention matrix N |Q_3gram| * |D_3gram|
Max/Mean Pooling
Fully-connected Layer
Relevance Score

Figure 5.1: Overview of the Multi-Perspective Hierarchical Convolutional Neural Network model, which consists of two parallel components for word-level and character-level modeling between queries, social media posts, and URLs. The two parallel components share the same architecture (with different parameters), which comprises hierarchical convolutional layers for representation learning and a semantic similarity layer for multi-level matching. Finally, all relevance signals are integrated using a fully-connected layer to produce the final relevance score.

motivating the need for character-level input modeling to copy with noisy texts.

To better understand the origin of OOV errors, I randomly select 500 OOV words from the vocabulary and provide a summary of the major sources of OOV occurrences in the social media domain as well as a few examples below:

1. **Compounds** (42.4%): chome-os, actor-director, earlystage

2. **Non-English words** (29.2%): emociones (Spainish, emotions), desgostosa (Portuguese, disgusted), hayatım (Turkish, sweetheart)

3. **Typos** (17.1%): begngen (beggen), yawnn (yawn), tansport (transport), afternoo (afternoon), foreverrrr (forever)

4. **Abbreviations** (5.6%): EASP (European Association of Social Psychology),

Query	MB001: BBC world service cuts
Tweet	BBC news - BBC world service cuts to be outlined to staff #bbcworldservice.
URL	http://bbc-world-service-to-cut-staff.html?spref=tw

Table 5.1: Example query-post pair retrieved by topic MB001 from the TREC Microblog 2011 dataset.

b-day (birthday)

5. **Domain-specific words** (5.7%): utf-8, vlookup

As we can see above, compounds, non-English words and typos are the three major source for OOV words. Character-level modeling is beneficial for both the compounds and typos cases.

In addition, social media posts often comprise many heterogeneous signals which can contain fruitful relevance signals, such as mentions, hashtags, or external URL links. An analysis over the TREC Microblog Track 2011–2014 datasets show around 50% tweet posts contain one or more URL links. More detailed statistics can be found in Table 3.1. In fact, by taking a closer look at the data, I observe many URL links can be fuzzy matched to query texts. I provide one example in Table 5.1. For those posts without URLs, I add a placeholder symbol "<URL>". It's worth noting that we don't model the document texts referenced by the URLs since many URL links are not accessible over time and the HTML formats of many web documents are quite noisy, making it difficult to extract content.

To tackle the above overwhelming language variation issues and utilize the URL information, I consider multiple inputs for relevance modeling: (1) query and post at word-level; (2) query and post at character-level; (3) query and URL at character-level. For character-level modeling, I partition the query and post content as well as the URL link to a sequence of character trigrams (e.g., "hello" to {#he, hel, ell, llo, lo#}), which has shown to obtain good effectiveness in capturing morphological variations and reducing the vocabulary size for efficient learning [11]. Then I adopt the same architecture as the word-level semantic modeling to capture the matching evidences at character-level, which I will discuss in the following section.

5.2.2 Hierarchical Representation Learning

Given a query q and a document d, the textual matching component aims to learn a relevance score $f(q, d)$ using the query terms $\{w_1^q, w_2^q, ..., w_n^q\}$ and document terms $\{w_1^d, w_2^d, ..., w_m^d\}$, where n and m are the number of terms in q and d, respectively. To be clear, "document" can either refer to a social media post or an URL, and "term" refers to either words or character trigrams. One important novel aspect of my model is relevance modeling from multiple perspectives, and my architecture exhibits symmetry in the word- and character-level modeling (see Figure 5.1), and thus for expository convenience, I use "document" and "term" in the generic sense above. I first employ an embedding layer to convert each term into a L-dimensional vector representation, generating a matrix representation for the query $\mathbf{Q}$ and document $\mathbf{D}$, where $\mathbf{Q} \in \mathbb{R}^{n \times L}$ and $\mathbf{D} \in \mathbb{R}^{m \times L}$. In the following, I introduce my representation learning method with hierarchical convolutional neural networks.

A convolutional layer applies convolutional filters to the text which are represented by an embedding matrix $\mathbf{M}$, such as $\mathbf{Q}$ or $\mathbf{D}$. Let $\mathbf{W} \in \mathbb{R}^{k \times L}$ denote a convolutional filter with a window size of k (L is the size of embeddings). We move this filter through the input text gradually, and at each step, we sum up the k term embeddings from the input matrix slice $\mathbf{M}_{i:i+k}$ weighted by the filter parameters $\mathbf{W}$. More formally, we obtain a vector representation $v \in \mathbb{R}^{\|M\|-k+1}$ of the input, with the i-th dimension of v calculated as:

$$v_i = \sum_{j=1}^{k} \sum_{l=1}^{L} \mathbf{W}_{j,l} \cdot \mathbf{M}_{i+j,l} + b,$$

where b is a bias value added to the weighted sum. Intuitively, v_i can be regarded as a weighted average of the i-th k-gram in the input sentence, learned by the filter $\mathbf{W}$. To ensure a fixed-size output vector v, we pad the input matrix $\mathbf{M}$ with $k-1$ zero columns such that v has a size of $\|M\|$, where$\|M\|$ equals to n for $\mathbf{Q}$ and m for $\mathbf{D}$. To increase the modeling capacity, each convolutional layer applies F different filters to the input, and therefore produces F output vectors $\{v_1, v_2, \ldots, v_F\}$. Lastly we concatenate all F output vectors and apply a non-linear activation function ReLU element-wise to obtain the output representation matrix $\mathbf{M}^o \in \mathbb{R}^{\|M\| \times F}$ for this CNN layer:

$$\mathbf{M}^o = \text{CNN}(\mathbf{M}) = \text{ReLU}([v_1; v_2; \ldots; v_F]).$$

This CNN layer with F filters comprises of $F \times (kL+1)$ parameters with $F \times (KL)$ parameters from the filters and F from the bias terms.

I then stack multiple convolutional layers in a hierarchical manner to obtain higher-level k-gram representations. For notation simplicity, I drop the superscript o from all output matrices and add a superscript h to denote the output of the h-th convolutional layer. Stacking N CNN layers therefore corresponds to obtaining the output matrix of the h-th layer $\mathbf{M}^h \in \mathbb{R}^{\|M\| \times F^h}$ via:

$$\mathbf{M}^h = \text{CNN}^h(\mathbf{M}^{h-1}), h = 1, \ldots, N,$$

where $\mathbf{M}^{h-1}$ is the output matrix of the $(h-1)$-th convolutional layer. Note that $\mathbf{M}^0 = \mathbf{M}$ denotes the matrix $\mathbf{Q}$ and $\mathbf{P}$ obtained directly from the word embedding layer, and the parameters of each CNN layer are shared by the query and document inputs.

Intuitively, consecutive convolutional layers allow us to obtain higher-level abstractions of the texts, starting from character-level or word-level to phrase-level and eventually to sentence-level. A single CNN layer is able to capture the k-gram semantics from the input embeddings, and two CNN layers together would allow us to expand the context window to up to $2k - 1$ terms. Generally speaking, the deeper the convolutional layers, the wider the context considered for relevance matching. Empirically, I found the filter size k of 2 for word-level inputs and 4 for character-level inputs worked well. The number of convolution layers N was set to 4. This setting is reasonable as it enables us to gradually learn the representations of word-level and character-level n-gram of up to $O(N * k)$ length. Since most queries and documents in the social media domain are shorter or closer to this length, we can think the outputs from the last convolutional layer as an approximation of the sentence representations.

An alternative to my deep hierarchical design is a *wide* architecture, which reduces the depth but expands the width of the network, by concatenating multiple convolutional layers with different filter sizes k in parallel to learn the variable-sized phrase representations. However, such design will require quadratically more parameters and be less efficient than my approach. More specifically, my deep model comprises of $O(N \times F \times kL)$ parameters with N CNN layers, while a wide architecture with the same representation window will need $O(F \times (kL + 2kL + ... + NkL)) = O(N^2 \times F \times kL)$ parameters. The saved parameters mainly come from the representation reusing at each CNN layer, which also generalizes the learning process by sharing representations between successive layers.

5.2.3 Similarity Measurement and Weighting

To measure the similarity between the query and the document, I match the query with the document at each convolutional layer by taking the dot product between the query representation matrix $\mathbf{M}_q$ and the document representation matrix $\mathbf{M}_d$:

$$\mathbf{S} = \mathbf{M}_q {\mathbf{M}_d}^T, \mathbf{S} \in \mathbb{R}^{n \times m},$$

where $\mathbf{S}_{i,j}$ can be considered the similarity score by matching the query phrase vector $\mathbf{M}_q[i]$ with the document phrase vector $\mathbf{M}_d[j]$. Since the query and document share the same convolutional layers, similar phrases will be placed closer in a high-dimensional embedding space and their product will produce larger scores. Next we obtain a normalized similarity matrix $\tilde{\mathbf{S}}$ by applying a *softmax* function over $\mathbf{S}$ to normalize the similarity scores into $[0, 1]$ range:

$$\tilde{\mathbf{S}}_{i,j} = \mathit{softmax}(\mathbf{S}_{i,j}) = \frac{e^{\mathbf{S}_{i,j}}}{\sum_{k=1}^{m} e^{\mathbf{S}_{i,k}}}.$$

For each query phrase i, the above *softmax* function normalizes its matching scores to all phrases in the document, and helps discriminate those matches with significant higher scores. An exact match will dominate others and contribute a similarity score close to 1.0. I then apply *max* and *mean* pooling to the similarity matrix to obtain discriminative feature vectors:

$$\mathit{Max}(\mathbf{S}) = [\max(\tilde{\mathbf{S}}_{1,:}), \max(\tilde{\mathbf{S}}_{2,:}), ..., \max(\tilde{\mathbf{S}}_{n,:})],$$

$$\mathit{Mean}(\mathbf{S}) = [\mathit{mean}(\tilde{\mathbf{S}}_{1,:}), \mathit{mean}(\tilde{\mathbf{S}}_{2,:}), ..., \mathit{mean}(\tilde{\mathbf{S}}_{n,:})],$$

$$\mathit{Max}(\mathbf{S}), \mathit{Mean}(\mathbf{S}) \in \mathbb{R}^n.$$

Each score generated from pooling can be viewed as a matching evidence of a specific query phrase to the document. Its value denotes the significance of relevance signal. Compared to *Max* pooling, *Mean* pooling is beneficial for the cases when a query phrase is matched to multiple relevant terms in the document.

To measure the relative importance of different query terms and phrases, I inject external weights as prior information by multiplying the score after pooling with the weighting of that specific query term/phrase. These are provided as feature inputs to the subsequent learning to rank layer, denoted by Φ:

$$\begin{gathered}\Phi = \{weights(q) \odot Max(\mathbf{S}), weights(q) \odot Mean(\mathbf{S})\}, \\ \Phi \in 2 \cdot \mathbb{R}^n,\end{gathered} \tag{5.1}$$

where $\odot$ is an element-wise product between the weights of query terms/phrases with the pooling scores. $weights(q)^i$ denotes the weight of the i-th term or phrase in the query. Its value changes in the intermediate CNN layers since deeper CNN layer represents longer phrases. Note that the weights of long phrases become sparse as the depth of CNN layers increases. Therefore I only use weights for the first two CNN layers ($N = 1, 2$) for word-level inputs, and $N = 1, 2, 3$ for character-level inputs. The weights of upper layers are assigned a default value of 1.0. I choose the classical inverse document frequency (IDF) as the weighting measure. A higher IDF weight implies a rarer occurrence in the collection thus a larger discrimination power. The weighting method also allows us to reduce the impact of high matching scores from common words like stop words. There can be some other weighting mechanisms, like weights generated from a pseudo-relevance feedback method [127] or from a sequential dependency model [128]. I leave these as future directions.

The similarity measurement layer has two important properties. First, all the layers here, including matching, softmax, pooling, and weights, have no learnable parameters. Second, the parameter-free nature enables my model to be highly interpretable and more robust from overfitting. By matching query phrases with document phrases in a joint manner, we can easily track which phrase matching contributes more relevance signal to the final prediction. This boosts the interpretability of my model greatly as it has become a prevalent concern with the complicated neural models for IR and NLP applications [129].

5.2.4 Evidence Integration

Given the similarity features learned from word-level Φ^w (from Equation 5.1) and character-level Φ^c, I employ a simple fully-connected layer with two linear layers and a non-linear ReLU activation in between as the learning to rank module:

$$o = \mathit{softmax}(\mathbf{W}_{h_2} \cdot \mathrm{ReLU}(\mathbf{W}_{h_1} \cdot \Phi + b_{h_1}) + b_{h_2}),$$
$$o \in \mathbb{R}^{|\mathrm{class}|},$$

where $\Phi = \Phi^w \sqcup \Phi^c$ and $\{\mathbf{W}_{h_1}, \mathbf{W}_{h_2}, b_{h_1}, b_{h_2}\}$ are the weight matrices and bias vectors in the two linear layers, $\sqcup$ is a concatenation operation. The outside *softmax* function normalizes the final prediction to a similarity vector o with its values between 0 and 1. The training goal is to minimize the negative log likelihood loss L summed over all samples (o_i, y_i) below:

$$L = -\sum_{(o_i,y_i)} \log o_i[y_i],$$

where y_i is the annotation label of sample i.

5.2.5 Interpolation with Language Model

Various studies have shown that neural network-based models are good at capturing *soft*-match signals [10, 89]. However, are the exact match signals still effective to neural network-based methods? I examine this hypothesis by adopting a commonly-used linear interpolation method to combine the ranking scores of NN-based model with language model between a (query, document) pair:

$$\text{Score}(q, d) = \lambda \cdot \text{NN}(q, d) + (1 - \lambda) \cdot \text{LM}(q, d). \tag{5.2}$$

The best hyper-parameter λ is tuned on the training and validation set, and the interpolated scores are leveraged for re-ranking. I choose the query-likelihood method (QL) [130] as the language model here. The interpolation technique is applied to my multi-perspective model and other NN-based methods I used as baselines in this paper. I report both effectiveness with and without interpolation in the experimental section.

5.3 Evaluation

5.3.1 Dataset Preprocessing

I use the four TREC Microblog topic sets (2011–2014), described in Chapter 3.5, to evaluate the effectiveness of the MP-HCNN model. I run four-fold cross validation where each of the four datasets is used for evaluation, with the other three used for training (e.g., train on TREC 2011–2013, test on TREC 2014). In each experiment, I sample 10% of the training queries as the validation set. The same padding strategy is used across the four datasets by setting to the largest query/document/URL length, where each query is padded to 10 words and 51 characters, each tweet is padded to 68 words and 140 characters and each URL is padded to 120 characters, respectively. The mentions are removed and hashtags are treated as normal words (i.e., "#bbc" to "bbc"). The IDF weights of word and character k-grams are computed from the Tweets2013 collection.

5.3.2 Baselines

I compare the MP-HCNN model to a number of competitive *non-neural* and *neural* baselines: 1) *non-neural* baselines such as language model and pseudo feedback algorithm; 2) *neural* baselines with recent *neural* ranking models designed for "standard" *ad hoc* retrieval tasks on web and newswire documents. The non-neural baselines are as follows:

1. **Query Likelihood** (QL) [130] is the most widely-used language modeling baseline.

2. **RM3** [127] is an interpolation model combining the QL score with a relevance model using pseudo-relevance feedback.

3. **Learning to Rank (L2R)**. I adopt the LambdaRank [60] as a L2R baseline, which is a competitive ranking algorithm that won the Yahoo! Learning to Rank Challenge [131]. I designed three sets of features: (a) **text-based**: in addition to QL, I compute another four overlap-based measures between each query-tweet pair (word overlap and IDF-weighted word overlap computed between all words and only non-stopwords, from Severyn and Moschitti [31]); (b) **URL-based**: whether the tweet contains URLs and the fraction of query terms that matched

parts of URLs; (c) **hashtag-based**: whether tweets contains hashtags and the fraction of query terms that matched hashtags.

The neural baselines are as follows:

4. **DSSM** (2013) [11] is one of the earliest NN architectures for web search that uses word hashing to model interactions between queries and programs at the level of character 3-grams.

5. **C-DSSM** (2014) [36] is a variant of DSSM that replaces the fully-connected layer in DSSM with a CNN-based model to capture local contextual signals from neighboring n-grams.

6. **MatchPyramid** (2016) [37] uses a CNN-based model to to extract matching patterns from word level to phrase level and sentence level from a similarity matrix.

7. **DRMM** (2016) [10] is an interaction-based approach that converts the similarity matrix of query and document to a histogram representation for relevance prediction.

8. **DUET** (2017) [91] is document ranking model that combines a local component for exact match and a global component for semantic match between query and document.

9. **K-NRM** (2017) [89] introduces a differentiable kernel-based layer to capture multi-level granularities of soft match signals from the input similarity matrix.

5.3.3 Implementation Details

Model Training. To enable fair comparison with the baselines, I adopt the same tuning strategies, such as embeddings, optimizer, and hyper-parameter tuning, in my experiments. I use the word2vec [92] 300-dimension word vectors pre-trained from Google News dataset with 100B tokens. From Table 3.1, more than 50% words (OOV words) are out of the word2vec vocabulary across all datasets. This could have a negative impact on model effectiveness since the embeddings of those OOV words and character trigrams are both initialized from a uniform sampling between $[0, 0.1]$. All the embeddings are updated during training. Stochastic gradient descent (SGD) with a learning rate of 0.05 and a batch size of 256 is used for training. The linear layer size in the learning to rank component is set to 150. The convolutional filter sizes are set to 2 for words and 4 for characters. The maximum number of convolutional layers N is set to 4. The number of convolutional filters is tuned between $\{64, 128, 256\}$, and the dropout rate is tuned between $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ on validation set. At test time, I selected the model that obtained the lowest loss on the validation set for evaluation. The interpolation parameter λ is tuned after the neural network model converges. My model is implemented using the Keras framework, while the other neural baselines are open-sourced in the MatchZoo library.[1]

Model Size. The total number of parameters in the proposed MP-HCNN model is about 71M, where 48% parameters are coming from the learnable word embeddings and another 47% are from the character trigram embeddings. Only about 5%

[1]`https://github.com/faneshion/MatchZoo`

(3.5M) parameters are from the convolutional part and learning to rank layer. It's worth noting that although word-level and character-level inputs share the same architecture, they have different parameters. For character-level inputs, query-post and query-URL modeling share the same parameters. The training process of MP-HCNN consumes about 3 minutes per epoch on a GPU machine (GeForce GTX 1080) with 8 GB memory and usually converges in 10 epochs.

5.3.4 Experimental Results

The experimental results of the MP-HCNN model are shown in Tables 5.2 and 5.3. Rows are numbered in the first column for convenience. I run statistical significance tests using Fisher's two-sided, paired randomization test [1] against the three non-neural baselines: QL, RM3, and L2R (with all features). Superscripts indicate the row indexes for which a metric difference is statistically significant at $p < 0.05$.

From the first block "Non-Neural Baselines" in Table 5.2 and 5.3, we can see that RM3 significantly outperforms QL on all datasets, demonstrating its superior effectiveness. However, RM3 requires an extra round of retrieval to select terms for query expansion, which is substantially slower. L2R achieves effectiveness on par with RM3 when using all the hand-crafted features. From its contrastive variant with only text-based features, we can see that the overlap-based features provide little gain over QL. Comparing the rows "(text+URL)" and "(text+hashtag)" to row "(text)", adding URL-based features leads to a significant improvement over

ID	Model Metric	2011 MAP	 P30	2012 MAP	 P30
Non-Neural Baselines					
1	QL [130]	0.3576	0.4000	0.2091	0.3311
2	RM3 [127]	0.3824^{1}	0.4211^{1}	0.2342^{1}	0.3452
3	L2R [60] (all)	0.3845^{1}	0.4279	0.2291^{1}	0.3559
	(text)	0.3547	0.4027	0.2072	0.3294
	(text+URL)	0.3816	0.4272	0.2317	0.3667
	(text+hashtag)	0.3473	0.4020	0.2039	0.3175
Neural Baselines					
4	DSSM [11] (2013)	0.1742	0.2340	0.1087	0.1791
5	C-DSSM [36] (2014)	0.0887	0.1122	0.0803	0.1525
6	DUET [91] (2017)	0.1533	0.2109	0.1325	0.2356
7	MatchPyramid [37] (2016)	0.1967	0.2259	0.1334	0.2390
8	DRMM [10] (2016)	0.2635	0.3095	0.1777	0.3169
9	K-NRM [89] (2017)	0.2519	0.3034	0.1607	0.2966
Neural Baselines with Interpolation					
10	DRMM+	0.3477	0.4034	0.2213	0.3537
11	DUET+	0.3576	0.4000	0.2243^{1}	0.3644^{1}
12	K-NRM+	0.3576	0.4000	0.2277^{1}	0.3520^{1}
Proposed Model					
13	MP-HCNN	0.3940	0.4306	0.2313^{1}	0.3757^{1}
14	MP-HCNN+	$\mathbf{0.4193}^{1,2,3}$ (+17.2%)	$\mathbf{0.4653}^{1,2,3}$ (+16.3%)	$\mathbf{0.2482}^{1,3}$ (+18.6%)	$\mathbf{0.3915}^{1,2,3}$ (+18.2%)

Table 5.2: Main results on TREC Microblog 2011–2012 datasets. Rows are numbered in the first column for convenience, and each row represents a model or a contrastive condition. Superscripts indicate the row indexes for which a metric difference is statistically significant at $p < 0.05$.

text-based features, while hashtag-based features seem to bring fewer benefits. This confirms our observation in Table 3.1 that URLs appear more frequently in tweets and contain meaningful relevance signals.

Looking at the second block "Neural Baselines", we find all the neural methods perform worse than the QL baseline. In fact, all the character-based approaches (DSSM, C-DSSM, and DUET) are consistently worse than the word-based approaches (MatchPyramid, DRMM, K-NRM). This is likely attributable to the fact

ID	Model Metric	2013 MAP	 P30	2014 MAP	 P30
	Non-Neural Baselines				
1	QL [130]	0.2532	0.4450	0.3924	0.6182
2	RM3 [127]	$0.2766^{1,2}$	0.4733^{1}	$\mathbf{0.4480}^{1,3}$	0.6339
3	L2R [60] (all)	0.2477	0.4617	0.3943	0.6200
	(text)	0.2394	0.4456	0.3824	0.6091
	(text+URL)	0.2489	0.4506	0.3974	0.6206
	(text+hashtag)	0.2447	0.4533	0.3815	0.5939
	Neural Baselines				
4	DSSM [11] (2013)	0.1434	0.2772	0.2566	0.4261
5	C-DSSM [36] (2014)	0.0892	0.1717	0.1884	0.2752
6	DUET [91] (2017)	0.1380	0.2528	0.2680	0.4091
7	MatchPyramid [37] (2016)	0.1378	0.2561	0.2722	0.4491
8	DRMM [10] (2016)	0.2102	0.4061	0.3440	0.5424
9	K-NRM [89] (2017)	0.1750	0.3178	0.3472	0.5388
	Neural Baselines with Interpolation				
10	DRMM+	0.2639	0.4772	0.4042	0.6139
11	DUET+	$0.2779^{1,3}$	0.4878^{1}	$0.4219^{1,3}$	$\mathbf{0.6467}^{1}$
12	K-NRM+	$0.2721^{1,3}$	0.4756	$0.4137^{1,3}$	0.6358^{1}
	Proposed Model				
13	MP-HCNN	$0.2856^{1,3}$	$0.5211^{1,3}$	0.4178	0.6279
14	MP-HCNN+	$\mathbf{0.2937}^{1,3}$	$\mathbf{0.5250}^{1,2,3}$	$0.4403^{1,3}$	0.6455
		(+15.9%)	(+17.9%)	(+12.2%)	(+4.4%)

Table 5.3: Main results on TREC Microblog 2013–2014 datasets. Superscripts indicate the row indexes for which a metric difference is statistically significant at $p < 0.05$.

that all word-based NN models use pre-trained word vectors that encode more semantics than a random initialization of character trigram embeddings, suggesting that the Twitter datasets are not sufficient to support learning character-based representations from scratch. Particularly, C-DSSM suffers more than DSSM, showing that a more complex model leads to lower effectiveness in a data-poor setting. Comparing the three word-based NN models, DRMM seems to be most effective while MatchPyramid is the worst. Considering that the three models share the same embedding-based similarity matrix as input, the large effectiveness differences between DRMM/K-NRM and MatchPyramid suggest that term weighting is crucial for tweet search. In addition, the smaller parameter space of DRMM (161 parameters in total) affirms that the low effectiveness is not simply because due to a shortage of data. As a comparison, my MP-HCNN model achieves high effectiveness on all datasets across both metrics, significantly beating all baselines in most settings.

In the third block "Interpolation Baselines", we observe that simple interpolation with QL boosts the effectiveness of all neural baselines dramatically, showing that the exact match signal is complementary to the soft match signals captured by the NN methods. This observation also holds for my MP-HCNN and only differs in a smaller margin of improvement (due to the effectiveness of MP-HCNN alone). The best results on TREC Microblog 2011–2013 datasets are achieved by MP-HCNN+, with an average of 15% relative improvement against QL (shown in last row). A minor exception is TREC 2014, where we see that the QL baseline already achieves fairly high absolute numbers, limiting the space for potential improvement.

Overall, the findings are consistent in the base model and interpolation setups: (1) existing NN models do not appear to provide effective rankings alone, while some are marginally effective with interpolation, showing that these ranking models fail to adapt to tweet search; (2) the MP-HCNN model is more effective than the neural and non-neural baselines we examined, suggesting that the customized design is necessary to capture domain-specific characteristics and challenges.

Setting	**2011**		**2012**		**2013**		**2014**	
Metric	**MAP**	**P30**	**MAP**	**P30**	**MAP**	**P30**	**MAP**	**P30**
QL	0.3576	0.4000	0.2091⋆	0.3311⋆	0.2532⋆	0.4450⋆	0.3924	0.6182
Full MP-HCNN	**0.3940**	**0.4306**	**0.2313**	**0.3757**	**0.2856**	**0.5211**	**0.4178**	**0.6279**
− mean pooling	0.3687⋆	0.4054⋆	0.2251	0.3480	0.2766	0.5000	0.3907⋆	0.5897⋆
− max pooling	0.0982⋆	0.1320⋆	0.0767⋆	0.1243⋆	0.0920⋆	0.1706⋆	0.1934⋆	0.2176⋆
− IDF weighting	0.3511⋆	0.3714⋆	0.2119⋆	0.3452	0.2717⋆	0.4967⋆	0.3992	0.6097⋆
− word module	0.1651⋆	0.1293⋆	0.0762⋆	0.1119⋆	0.0987⋆	0.1517⋆	0.1849⋆	0.2048⋆
− URL char rep.	0.3594⋆	0.3707⋆	0.2131⋆	0.3333⋆	0.2797⋆	0.4989⋆	0.4037⋆	0.6085⋆
− doc char rep.	0.3603⋆	0.3721⋆	0.2188⋆	0.3537⋆	0.2757⋆	0.5122	0.4012	0.6103
− char module	0.3528⋆	0.3709⋆	0.2087⋆	0.3271⋆	0.2718⋆	0.5011⋆	0.4050⋆	0.6091⋆

Table 5.4: MP-HCNN Ablation Study. ⋆ denotes the score is significantly lower than the base MP-HCNN model at $p < 0.05$.

5.3.5 Ablation Study

To better understand the contribution of each module in the proposed model, I perform an ablation study on the base MN-CNN model, removing each component step by step. Here, I aim to study how the semantic-level, character-level, and weighting modules contribute to model effectiveness. These results are shown in Table 5.4, with each row denoting the removal of a specific module. For example, the row "− URL char rep." represents removing the URL modeling module. The ⋆ symbol denotes that the model's effectiveness in the ablation setting is significantly lower than base MP-HCNN model at $p < 0.05$. I also add QL performance in the table as a reference.

From the first two rows "w/o max/mean pooling", we can see that removing the max pooling leads to a significant performance drop while taking out mean pooling only results in a minor reduction. This matches our observation that most query terms only receive at most one exact or relevant match in the short tweets. Mean pooling on matching features is largely dominated by the max pooling, which selects the largest matching score for each query term. Also, removing the IDF weights makes the results consistently and significantly worse across the four datasets, which confirms that injecting external weights is important for tweet search. It is also no surprise that the complete word-level module is essential to model effectiveness, as shown in the table.

Turning our attention to the last three rows, we observe that removing the character representations of URLs or documents both lead to significant drops across all datasets, with larger drops when URL representations are removed. This suggests that URLs provide more relevance signals than character-level document modeling. Taking away the entire character-level module causes slightly more effectiveness loss. To conclude, the word-level matching module contributes the most effectiveness, but the character-level matching module still provides complementary and significantly useful signals. However, recall the low effectiveness of character-based methods in Table 5.2 and 5.3, we add a caveat: with more training data or pre-trained character trigram embeddings, we could expect the benefits of the character-level matching module to improve.

Additionally, I examine how the depth of hierarchical convolutional layer affects the model effectiveness. Figure 5.2 shows the effectiveness distribution on MAP score with different convolutional depth N on TREC 2011–2014 datasets. A setting of $N = 0$ means there are no convolutional layers on top of the embedding layer, and the prediction is purely based on the matching evidence at word-level. A larger value of N indicates wider ranges of phrases are represented and modeled. We can clearly see there is a consistent climbing pattern with increasing depth on

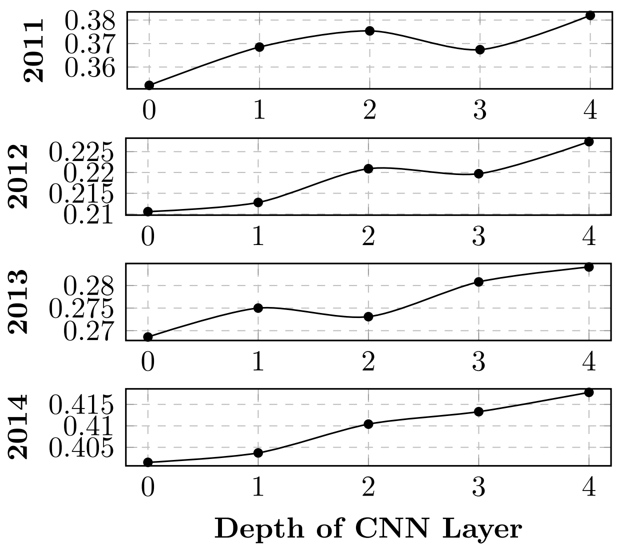


Figure 5.2: MAP score with different convolutional depth N on TREC 2011–2014 datasets.

all datasets, except for $N = 3$ on TREC 2011. For the dataset 2011, 2012 and 2014, the improvements at $N = 2$ are quite close to the upper bound at $N = 4$. This implies modeling of short phrases brings immediate effectiveness gains while the inclusion of longer phrases further boosts the overall effectiveness. I don't explore larger values of N as $N = 4$ already enables us to model a window of $O(N \times k) = 8$ consecutive words, which is longer than than most queries and close to the length of many tweets. Overall, this ablation experiment clearly shows the value of the hierarchical convolutional layers in semantic modeling at the phrasal level.

5.3.6 Error Analysis

So far, we have shown that the weighted similarity measurement component, as well as the URL matching and phrase matching (enabled by the hierarchical architecture), are crucial to the model's effectiveness. However, we still lack knowl-

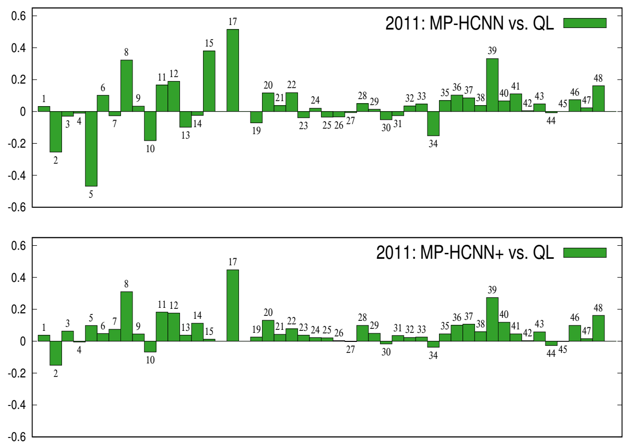


Figure 5.3: Per-query MAP differences of MP-HCNN and MP-HCNN+ vs. QL on TREC 2011.

edge about the following two questions: (1) What are the common characteristics of well-performing queries, and how do the different components contribute their effectiveness? (2) When does the proposed model fail, and how can we further improve the model? Therefore, I provide additional qualitative and quantitative analysis over sample tweets from well-performing and poor-performing queries.

In Figures 5.3 and 5.4, I visualize the per-query improvements on the MAP metric for MP-HCNN and MP-HCNN+ against the QL baseline on the TREC 2011 and 2012 datasets, respectively. Since the TREC 2013 and 2014 datasets exhibit similar trends, I omit their figures here. Overall, we see that the base MP-HCNN model shows improvements for the majority of queries in both the 2011 and 2012 datasets. In 2011, MP-HCNN wins on 26 topics and loses on 13 topics out of 49 topics; in 2012, it wins on 35 topics and loses on 19 topics out of 60 topics. The average margin of improvement is also greater than the losses. With the interpolation technique, MP-HCNN+ is able to smooth out the errors of many

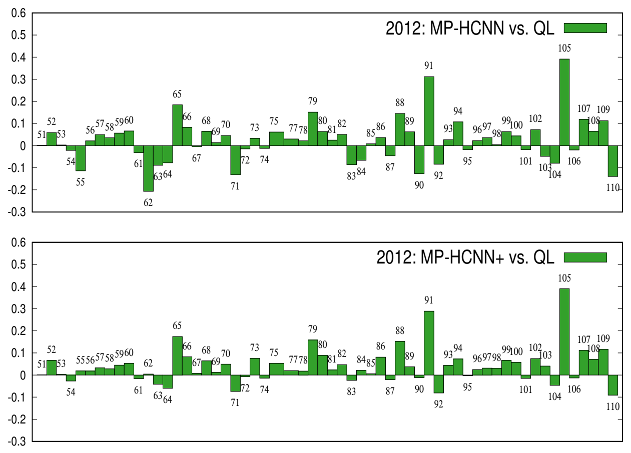


Figure 5.4: Per-query MAP differences of MP-HCNN and MP-HCNN+ vs. QL on TREC 2012.

poor-performing topics, such as topic 5 “nist computer security”, resulting in more stable improvements.

For the five best-performing queries (15, 17, 39, 91, 105), I select the top 20 tweets for each query sorted by the MP-HCNN prediction scores for analysis. I manually classify the matching evidence of the selected 100 tweets into the following categories (a tweet can satisfy multiple categories):

- **Exact word match**: the tweet has exact word matches with the query.
- **Exact phrase match**: the tweet has exact phrase matches with the query.
- **Partial paraphrase match**: the tweet has partial phrase matches with the query. For example, the phrase “the white stripes call it quits” is partially matched to the query 17 (“white stripes breakup”).
- **Partial URL match**: the query is contained in or partially matched to the URL in the tweet.

Category	Percentage (%)
Exact word match	100
Exact phrase match	44
Partial paraphrase match	59
Partial URL match	29

Table 5.5: Matching evidence breakdown by category based on manual analysis of the top 100 tweets for the five best-performing topics.

Table 5.5 provides a breakdown of matching evidence by category. We can see that all tweets have exact word matches to the queries, and partial paraphrase matches occur more frequently than exact phrase matches, suggesting that the hierarchical architecture with embedding inputs is able to capture those soft semantic match signals. In addition, partial URL matches make up another big portion, affirming the need for character-level URL modeling.

To gain additional insights into how my model fails, I select some sample tweets for the worst-performing queries 2 ("2022 fifa soccer") and 5 ("nist computer security"). Some of these sample tweets are shown in Table 5.6. Column "Label" represents whether the tweet is relevant to the query: "R" denotes relevant and "I" denotes irrelevant. Column "Score/Rank" shows the prediction scores and the ranked position of sample tweets produced by each method (QL or MP-HCNN). In addition, I also visualize the matching scores produced by the similarity measurement layer. The scores are normalized to range [0, 1] from the softmax function, and are visualized with the pink color background. The brighter the color, the higher the score. For example, in the second tweet, the word "fifa" has a matching score of 0.99 to the query, while "2022" has a matching score of 0.22.

Looking at the first tweet, it obtains the highest score by MP-HCNN due

ID	Query	Sample Tweet	Label	Score/Rank	
				QL	MP-HCNN
1	2: 2022 fifa soccer	#ps3 best sellers: fifa soccer 11 ps3 #cheaptweet https://www.amazon.com/fifa-soccer-11-playstation-3	I	7.33(#54)	0.85(#1)
2		qatar 's 2022 fifa world cup stadiums: https://wordlesstech.com/qatars-2022-fifa-world-cup-stadiums/	R	10.58(#2)	0.41(#105)
3		2022 world cup could be held at end of year: fifa : lausanne switzerland the 2022 world cup in qatar: http://www.reuters.com/article/us-soccer-world-blatter	R	11.25(#1)	0.31(#127)
4	5: nist computer security	cybersecurity : nist provides advice on securing full virtualization technologies: the national #security #hacker https://www.infosecurity.com/news/nist-provides-advice-on-securing-full/	R	9.79(#6)	0.39(#1)
5		photo: abdul buvar (computer security expert) malware expert and consultant for network security as a http://krr48.tumblr.com/post/abdul-buvar-computer-security-expert-malware	I	5.40(#45)	0.28(#2)
6		new nist guidance tackles public cloud security : 2 other special pubs on cloud defs virtualization http://www.govinfosecurity.com/articles.php?art_id=3321	R	9.79(#5)	0.24(#5)

Table 5.6: Sample analysis of the bad-performing topic 2 ("2022 fifa soccer") and topic 5 ("nist computer security"). **R** stands for relevant and **I** stands for irrelevant.

to the phrase match "fifa soccer" (a matching score of 0.89) from the content and URL. However, the MP-HCNN model fails to understand that "fifa soccer 11" refers to a video game on PS3, showing the limits of a matching-based algorithm for entity disambiguation. In contrast, though the second and third tweets look more relevant to the query, they are assigned much lower scores by MP-HCNN. This is because the query word "2022" is an out-of-vocabulary word, thus the impact of its matching evidence is greatly reduced due to the random initializations of OOV word embeddings. Comparing the second and third tweet, they share similar matching evidence in the content while the third tweet has a higher MP-HCNN score due to the character *n*-gram match "2022-fifa" in its URL. Also, it's worth noting that there are many terms that co-occur with "fifa soccer" in relevant tweets such as "qatar" and "world cup", suggesting that neural networks for term expansion can be promising. Since tweets 4–6 show similar patterns, I omit detailed discussions here.

In summary, the results of these manual analyses confirm the quantitative results from the previous sections. Exact term match remains critical to relevance modeling, while soft matches that incorporate phrases and semantic similarities make substantial contributions as well. Furthermore, although URLs provide a smaller role in matching, they appear to provide complementary signals as well. Though soft-match signals can be led astray, as the above failure analysis shows, overall they help more than they hurt.

5.4 Conclusion

To conclude, this chapter presents, to my knowledge, the first substantial work on neural ranking models for *ad hoc* retrieval on social media. I have identified three main characteristics of social media posts that make the problem different from "standard" document ranking over web and newswire documents. My model is specifically designed to cope with each of these issues, capturing multiple signals from queries, social media posts, as well as URLs contained in the posts – at the character-, word-, and phrase-levels. Extensive experiments demonstrate the effectiveness of my model and ablation studies verify the importance of each model components, suggesting that the customized architecture indeed captures the characteristics of the domain-specific ranking challenge.

Chapter 6: Temporal Modeling of Session Behaviors for Voice Search

6.1 Introduction

Voice-based interactions with computing devices are becoming increasingly prevalent, driven by several convergent trends. The ubiquity of smartphones and other mobile devices with restrictive input methods makes voice an attractive modality for interaction: Apple's Siri, Microsoft's Cortana, and the Google Assistant are prominent examples. Google observed that there are more searches taking place from mobile devices than from traditional desktops [38], and that 20% of mobile searches are voice queries [39]. The success of these products has been enabled by advances in automatic speech recognition (ASR), thanks mostly to deep learning.

Increasing comfort with voice-based interactions, especially with AI agents, feeds into the emerging market on "smart homes". Products such as Amazon Echo and Google Home allow users to control a variety of devices via voice (e.g., "turn on the TV", "play music by Adele"), and to issue voice queries (e.g., "what's the weather tomorrow?"). The market success of these products demonstrates that people do indeed want to control smart devices via voice.

In this paper, I tackle the problem of navigational voice queries posed against an entertainment system, where viewers interact with a voice-enabled remote controller to specify the program (TV shows, movies, sports games) they wish to watch. If a viewer wishes to watch the popular series "Game of Thrones", saying the name of the program should switch the television to the proper channel. This is simpler and more intuitive than scrolling through channel guides or awkwardly trying to type in the name of the show on the remote controller. Even if the viewer knows that Game of Thrones is on HBO, finding the right channel may still be challenging, since entertainment packages may have hundreds of channels.

The problem is challenging for a few reasons. Viewers have access to potentially tens of thousands of programs (including on-demand titles), whose names can be ambiguous. For instance, "Chicago Fire" could refer to either the television series or a soccer team. Even with recent advances, ASR errors can exacerbate ambiguity by transcribing queries like "Caillou" (a Canadian children's education television series) as "you". In addition, I observe that voice queries are very short, which makes the prediction problem more difficult because there is less signal to extract.

I tackle the above challenges using two key ideas to infer user intent: hybrid query representations and modeling search sessions. Specifically, my contributions are as follows:

- To my knowledge, this work serves the first to systematically study voice queries in the entertainment context. I propose a technique to automatically collect ground truth labels for voice query sessions from real-world usage data.

- My approaches model voice search sessions to understand the contextual dependencies in query sequences, which are accomplished with a probabilistic framework in which recurrent and feedforward neural network modules are organized in a hierarchical manner.

- Evaluations demonstrate the effectiveness of my context-aware models, significantly outperforming competitive baselines. Detailed analyses clarify *how* my models are better able to understand user intent.

This chapter is organized as follows: I first introduce the background of this work in Chapter 6.2. Then I describe the model architecture in Chapter 6.3, and present the experimental setup in Chapter 6.4 and evaluation results in Chapter 6.5. Next, I present the deployment practice of my model into the Xfinity entertainment platform in Chapter 6.6. Finally, I conclude this chapter in Chapter 6.7.

6.2 Background

The context of this work is voice search on the Xfinity X1 entertainment platform by Comcast, one of the largest cable companies in the United States with approximately 22 million subscribers in 40 states. X1 refers to a software package distributed on top of Xfinity's most recent cable box, which has been deployed to 17 million customers since around 2015. X1 can be controlled via the "voice remote", which is a remote controller that has an integrated microphone to receive voice queries from viewers. The current deployed system is based on a combination of hand-crafted rules and machine-learned models to arrive at a final response. The system has a diverse set of capabilities, which increases query ambiguity and magnifies the overall challenge of understanding user intent. These capabilities range from channel change to entity search (e.g., sports team, person, movie, etc.). In addition, voice queries may involve general questions, from home security control to troubleshooting the wifi network, or may be ultimately directed to external apps such as Pandora. In this work, I focus on navigational voice queries where viewers specify the TV program they wish to watch.

In our application, we receive as input the one-best transcription from the ASR system. We do not have access to the acoustic signal, as the ASR system is a black box. While it would be ideal if we could build joint models over both the acoustic signal, transcription lattice, and user intent, in many operational settings this is not practical or even possible. In the case of X1, the ASR is outsourced to a third party—a scenario not uncommon in many organizations. Thus, transcription errors compound ambiguity in the queries and make this problem harder.

This work, of course, is not the first to tackle voice search [94–98], although to my knowledge it is the first to focus on voice queries directed at an entertainment system. How is this particular domain different? The setting is obviously different—in our case, viewers are clearly sitting in front of a television with an entertainment intent. To compare and contrast viewers' actual utterances, we can turn to previously-published work that studied the characteristics of voice search logs, especially in comparison to text search data [99–102]. Schalkwyk et al. [102] reported statistics of queries collected from Google Voice search logs, which found that short queries, in particular 1-word and 2-word queries, were more common in the voice search setting, while long queries were much rarer. In contrast, a more recent study by Guy [101] reported that voice queries tend to be longer than text queries, based on a half-million query dataset from the Yahoo mobile search application. The average length across 32M voice queries is 2.04 in our dataset, much shorter than the reported average of 4.2 for Yahoo voice search [101].

We note another important difference between our entertainment context and voice search applications on smartphones: on a mobile device, it is common to back off to a web search if the query intent is not identified with high confidence. Less than half of Yahoo voice queries (43.3%) are handled by a pre-defined card [101]. While we am not aware of a scientific study about such behavior on a TV, our intuition is that a list of search results is less useful to TV viewers than it might be for smartphone users, since subsequent interactions are more awkward: it is difficult for users to scroll and they have limited input methods for follow-up interactions. Of course, this does not mean that ranking is unimportant, and my methods are evaluated using several rank-based metrics.

My proposed approach to tackling ambiguous voice queries is to take advantage of context in voice search sessions. The assumption is that when a viewer is not satisfied with the results of a query, she will issue more queries in rapid succession and continue until the desired program is found or until she gives up. In fact, we've found that across 20M sessions in a week (details in Chapter 6.4.1), more than 30% of sessions have multiple queries, accounting for over 57% of all queries. Figure 6.1 shows that more than half of the users issued at least one multi-query session in that week.

Preliminary explorations suggest that multi-query sessions are at least in part due to an unsatisfied user reformulating the original query. That is, sessions are long

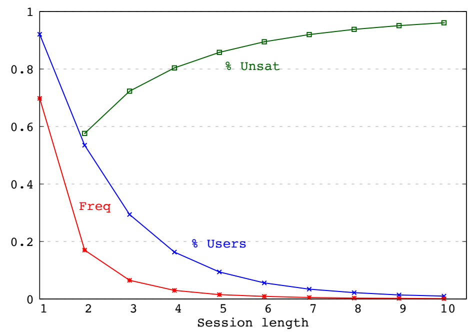


Figure 6.1: For each session length, I plot three values: frequency of sessions (red), percentage of users that issued at least one session of that length (blue), and percentage of users with at least one unsatisfactory experience (green).

because users couldn't find what they were looking for on the first try. Since the deployed system does not have interaction or dialogue capabilities, we need a proxy for measuring user satisfaction. For each session in the voice query log (details in Chapter 6.4.1), I computed the character edit distance between every pair of queries (normalized by the sum of query lengths) and labeled the session "unsatisfactory" if the minimum was less than 25% (two identical queries will have a zero edit distance), the intuition being that reformulated queries are likely to be similar. To verify, I sampled some of these sessions and confirmed that this heuristic does indeed capture reformulation-heavy sessions. In the dataset, over half of the users had at least one voice session with multiple queries (blue line in Figure 6.1), and among these users, close to 60% of them had at least one unsatisfactory experience (green line). This rate keeps going up as the session length increases.

Initial analyses suggest that better modeling of context in multi-query sessions may lead to an improved user experience. For example, compare two sessions: ["ncis", "cargo fire", "chicago fire"] and ["espn", "chicago sports", "chicago fire"]. Although both end in the same query, it is fairly clear that in the first case, the viewer is interested in the TV drama series “Chicago Fire” (previous queries all mention other drama series), whereas in the latter, the viewer is interested in the sports team with the same name. Modeling context in search sessions is of course not a new idea, and my approach is inspired by previous work in web search [132–137]. However, I am working in a completely different domain. Building on recent advances in deep learning applied to information retrieval [46, 91, 138, 139], I decided to implement these ideas using neural networks.

6.3 Model Architecture

6.3.1 Problem Formulation

Given a voice query session $[q_1, \ldots, q_n]$, the task is to predict the program p that the user intends to watch. The model performs this prediction cumulatively at each time step $t \in [1, n]$ on each successive new voice query q_t, exploiting all previous queries in the session, $[q_1, \ldots, q_{t-1}]$. For example, in a three-query session $s_i = [q_{i_1}, q_{i_2}, q_{i_3}]$, there will be three separate predictions: first with $[q_{i_1}]$, second with $[q_{i_1}, q_{i_2}]$, and third with $[q_{i_1}, q_{i_2}, q_{i_3}]$. I sessionize the voice query logs heuristically based on a time gap (in this case, 45 seconds—more details later), similar to how web query logs are sessionized based on inactivity. As described above, each query is a text string, the output of a third-party “black box” ASR system that we do not have internal access to.

I aim to learn a mapping function Θ from a query sequence to a program prediction, modeled using a probabilistic framework:

$$\begin{aligned} &\text{Data: } D = \{(s_i, p_i) \mid s_i = [q_{i_1}, ..., q_{i_{|s_i|}}],\ p_i \in \Phi\}_1^{|D|} \\ &\text{Model: } \hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{|D|} \prod_{t=1}^{|s_i|} P(p_i | q_{i_1}, ..., q_{i_t}; \theta) \end{aligned} \tag{6.1}$$

where D denotes a set of labeled sessions (s_i denotes the i-th session with $|s_i|$ queries), p_i is the intended program for session i, Φ is the global set of programs, and θ is the set of parameters in the mapping function Θ. The goal is to maximize the product of prediction probabilities.

I decompose the program prediction task into learning three mapping functions: a query embedding function $\mathbb{F}(x; \theta_{\mathbb{F}})$, a contextual function $\mathbb{G}(x; \theta_{\mathbb{G}})$, and a classification function $\mathbb{H}(x; \theta_{\mathbb{H}})$. The query embedding function $\mathbb{F}(\cdot)$ takes the text of the query as input and produces a semantic representation of the query. The contextual function $\mathbb{G}(\cdot)$ considers representations of all the preceding queries as context and maps them to a high-dimensional embedding vector to capture both semantic and contextual features. Finally, the classification function $\mathbb{H}(\cdot)$ predicts possible programs from the learned contextual vector. I adopt the following decomposition:

$$\begin{aligned} P(p_i | q_{i_1}, ..., q_{i_t}) \sim\ & P(p_i | c_{i_t}) \cdot P(c_{i_t} | v_{i_1}, ..., v_{i_t}) \\ & \cdot P(v_{i_1}, ..., v_{i_t} | q_{i_1}, ..., q_{i_t}) \end{aligned} \tag{6.2}$$

where c_{i_t} denotes the *contextual* embedding of the first t queries in the i-th session and v_{i_t} denotes the embedding of the t-th query of the i-th session. The relationship between these embeddings can be formulated using the three mapping functions above: $\mathbb{F}$ maps a query q_{i_j} to its embedding v_{i_j} in vector space; $\mathbb{G}$ maps a sequence of query embeddings $[v_{i_1}, ..., v_{i_t}]$ to a contextual embedding c_{i_t}; and $\mathbb{H}$ maps the contextual embedding to a program p_i:

$$v_{i_t} \sim \mathbb{F}(q_{i_t}; \theta_{\mathbb{F}}), \quad c_{i_t} \sim \mathbb{G}(v_{i_1}, ..., v_{i_t}; \theta_{\mathbb{G}}), \quad p_i \sim \mathbb{H}(c_{i_t}; \theta_{\mathbb{H}})$$
$$1 \leq t \leq |s_i|$$

By assuming that each query is embedded independently, we can reduce the last term in Equation (7.2) as follows:

$$P(p_i | q_{i_1}, ..., q_{i_t}) = P(p_i | c_{i_t}) \cdot P(c_{i_t} | v_{i_1}, ..., v_{i_t}) \cdot \prod_{j=1}^{t} P(v_{i_j} | q_{i_j})$$

I model the query embedding function $\mathbb{F}(\cdot)$ and the contextual function $\mathbb{G}(\cdot)$ by organizing two Long Short-Term Memory (LSTM) [73] models in a hierarchical manner. The decision function $\mathbb{H}(\cdot)$ uses a feedforward neural network.

I obviate the introduction of LSTM here, but more details can be found in Chapter 2.2.2. Given an input sequence $\mathbf{x} = (x_1, ..., x_T)$, an LSTM model outputs a sequence of hidden vectors $\mathbf{h} = (h_1, ..., h_T)$. A memory cell at position t digests the input element x_t and previous state information h_{t-1} to produce updated state h_t using the standard LSTM equations [73]. In the following sections, we refer to the size of the output vector h as the LSTM size.

6.3.2 Query Representation

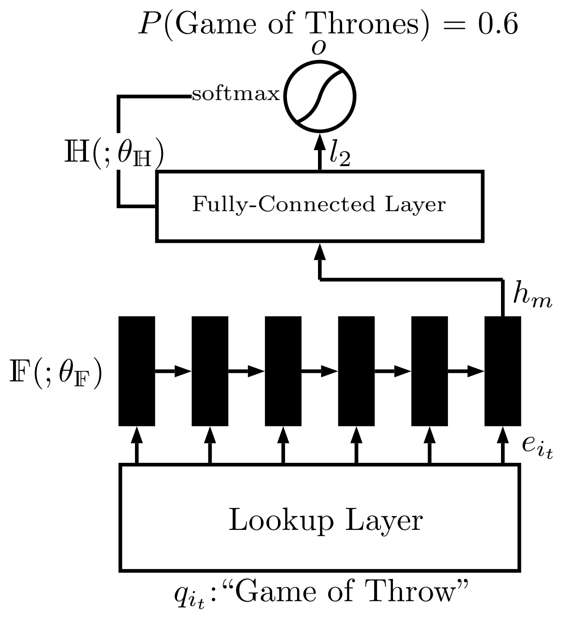
Since query strings serve as the sole input to my model, an expressive query representation is essential to accurate predictions. I represent each query as a sequence of elements (words or characters); each element is passed through a lookup layer and projected into a d-dimensional vector, thereby representing the query as an $m \times d$ matrix (m is the number of elements in the query). I consider three variations of this representation:

(1) Character-level representation, which encodes a query as a sequence of characters and the lookup layer converts each character to a one-hot vector. In this case, m would be the number of characters in the query and d would be the size of the character dictionary of the entire dataset.

(2) Word-level representation, which encodes the query as a sequence of words, and the word vectors are read from a pre-trained word embedding, e.g., word2vec [12]. In this case, d would be the dimensionality of the word embedding.

(3) Combined representation, which combines both the character-level and word-level representations by feeding the representations to two separate query embedding functions $\mathbb{F}_c$ and $\mathbb{F}_w$, respectively, and then concatenating the two learned vectors v_c and v_w as the combined query embedding vector.

My intuition for these different representations is as follows: Based on my query log analysis, I observe many unsatisfactory responses due to ASR errors. For example, voice queries intended for the program "Caillou" (a Canadian children's education television series) are often recognized as "Cacio" or "you". Capturing such variations



$P(\text{Game of Thrones}) = 0.6$
o
softmax
$\mathbb{H}(;\theta_{\mathbb{H}})$
l_2
Fully-Connected Layer
h_m
$\mathbb{F}(;\theta_{\mathbb{F}})$
e_{i_t}
Lookup Layer
q_{i_t}:"Game of Throw"

Figure 6.2: Architecture of the Basic Model.

with a word-level representation would likely suffer from data sparsity issues. On the other hand, initializing a query through word embedding vectors would encode words in a semantic vector space, which would help in matching queries to programs based on semantic relatedness (e.g., the query "Portland Trail Blazers" is semantically similar to the intended program "NBA basketball" without any words in common). Word embeddings are also useful for recognizing semantically-similar contextual clues such as "Search", "Find" or "Watch". With a character-level representation, such similarities would need to be learned from scratch. Whether the benefits of either representation balance its drawbacks is an empirical question I study through experiments, but I expect that a combined representation can capture the best of both worlds.

6.3.3 Basic Model

In the basic context-independent model, queries in a session are assumed to be independent and thus I do not attempt to model context. That is, each query is treated as a complete sample for model inference and prediction. The mapping function Θ from query to program in Equation (7.1) can be simplified as follows:

$$\begin{aligned}\Theta &\sim \arg\max_{\theta} \prod_{i=1}^{|D|} \prod_{t=1}^{|s_i|} P(p_i | q_{i_t}) \\ &= \arg\max_{\theta} \prod_{i=1}^{|D|} \prod_{t=1}^{|s_i|} P(p_i | v_{i_t}) P(v_{i_t} | q_{i_t})\end{aligned} \tag{6.3}$$

$$v_{i_t} \sim \mathbb{F}(q_{i_t}; \theta_{\mathbb{F}}), \quad p_i \sim \mathbb{H}(v_{i_t}; \theta_{\mathbb{H}}), \quad 1 \leq t \leq |s_i|$$

Here, the program p_i is only dependent on the current query q_{i_t}. The contextual function $\mathbb{G}(\cdot)$ is modeled as an identity function since there is no context from the assumption.

The architecture of the basic model is shown in Figure 6.2. In the bottom, I use an LSTM as the query embedding function $\mathbb{F}(\cdot)$. The text query is projected into an $m \times d$ dimensional matrix through the lookup layer, then fed to the LSTM, which has m memory cells and each cell processes an element vector. The hidden state at the last time step h_m is used as the query embedding vector v. At the top, there is a fully-connected layer followed by a softmax layer for learning the classification function $\mathbb{H}(\cdot)$. The fully-connected layer consists of two linear layers with one element-wise activation layer in between. Given the query embedding vector v as input, the fully-connected layer computes the following:

$$l_2 = W_{h_2} \cdot \sigma(W_{h_1} \cdot v + b_{h_1}) + b_{h_2}$$

where the W terms are the weight matrices and the b terms are bias vectors. I use the tanh function as the non-linear activation function σ, which is commonly adopted in neural network architectures. The softmax layer normalizes the vector l_2 to an $L1$ norm vector o, with each output score $o[p_j]$ denoting the probability of producing program p_j as output:

$$o[p_j] = \frac{\exp(l_2[p_j] - \text{shift})}{\sum_{p_k=1}^{|\Phi|} \exp(l_2[p_k] - \text{shift})}$$

where shift $= \max_{p_k=1}^{|\Phi|} l_2[p_k]$, $|\Phi|$ is the total number of programs in the dataset.

I adopt the negative log-likelihood loss function to train the model, which is derived from Equation (6.3):

$$\begin{aligned} L &= -\sum_{i=1}^{|D|} \sum_{t=1}^{|s_i|} \log P(p_i | q_{i_t}) + \lambda \cdot \| \langle \theta_{\mathbb{F}}, \theta_{\mathbb{G}}, \theta_{\mathbb{H}} \rangle \|^2 \\ &= -\sum_{i=1}^{|D|} \sum_{t=1}^{|s_i|} \log o_{i_t}[p_i] + \lambda \cdot \| \langle \theta_{\mathbb{F}}, \theta_{\mathbb{G}}, \theta_{\mathbb{H}} \rangle \|^2 \end{aligned}$$

where o_{i_t} is the score vector computed from query q_{i_t} and p_i is the true program for session i; λ is the regularization weight and $\langle \theta_{\mathbb{F}}, \theta_{\mathbb{G}}, \theta_{\mathbb{H}} \rangle$ is the set of model parameters. The optimization goal is to minimize the loss criterion L.

The training process is shown in Algorithm 1. The overall structure is to iter-

Algorithm 1 Training the Basic Model

```
1: for each session $s_i$ in the dataset $i = 1...|D|$ do
2:     for each query $q_{i_t}$ in session $s_i$ with $t = 1...|s_i|$ do
3:                                                    ▷ Forward Prediction Start
4:         $e_{i_t} = \text{encode}(q_{i_t})$
5:         $h_{1,...,m} = \text{LSTM:forward}(e_{i_t})$
6:         $l_2 = \text{FC:forward}(h_m)$
7:         $o = \text{softmax:forward}(l_2)$
8:         $loss = \text{criterion:forward}(o, p_i)$
9:                                                    ▷ Backward Propagation Start
10:        grad_criterion = criterion:backward($o, p_i$)
11:        grad_soft = softmax:backward($l_2$, grad_criterion)
12:        grad_linear = FC:backward($h_m$, grad_soft)
13:        grad_lstm = zeros($m$, lstm_size)
14:        grad_lstm[$m$] = grad_linear
15:        LSTM:backward($e_{it}$, grad_lstm)
16:        update_parameters()
17:    end for
18: end for
```

ate over each query in all sessions to perform the forward prediction and backward propagation operations. The *forward* phase follows the model architecture in Figure 6.2. A query is first encoded as a matrix in Line 4 by specifying the encoding method (i.e., character, word, or combined). Lines 5 and 6 feed the input matrix to the LSTM and fully-connected (FC) layer sequentially. In the *backward* phase, each module requires the original inputs and the gradients propagated from its upper layer to compute the gradients with respect to the inputs and its own parameters. It is worth noting that in Lines 13-15 the gradients grad_lstm are initialized to zero for the first $m - 1$ cells. This is because in the forward phase, I only use the last LSTM state h_m as the query embedding vector for the upper layers. Line 16 performs gradient descent to update model parameters. All the forward and backward functions used here are written as black box operations, and interested readers can refer to Goodfellow et al. [140] for more details.

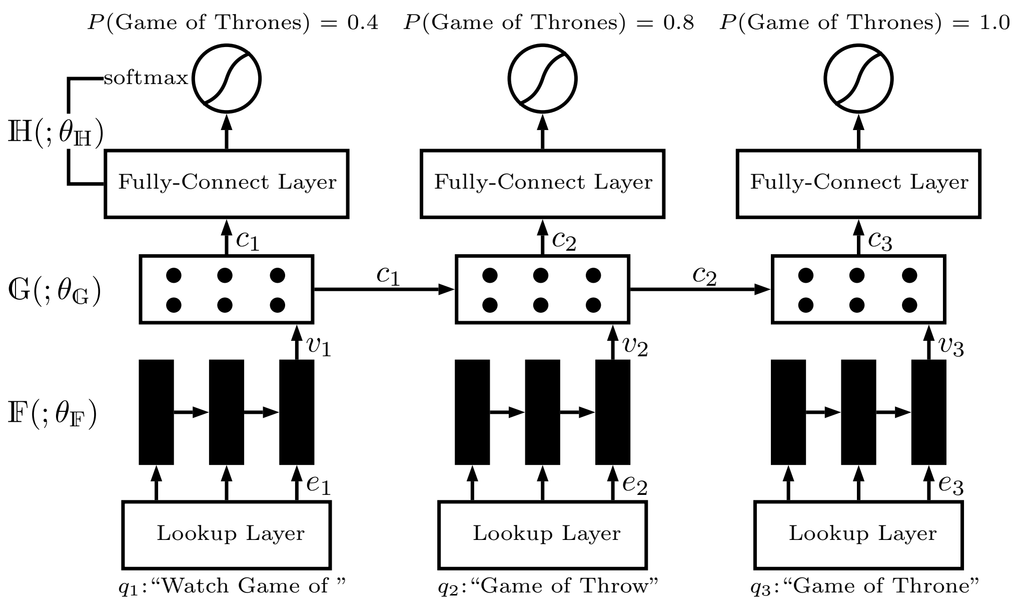


Figure 6.3: Architecture of the Full Context Model.

6.3.4 Full Context Model

I propose two approaches to model context: the full context model (presented here) and the constrained context model (presented next). The architecture of the full context model is shown in Figure 6.3, which uses the basic model as a building block. I use another LSTM (the dotted rectangle in the middle of Figure 6.3) to learn the contextual function $\mathbb{G}(v_1, ..., v_t; \theta_{\mathbb{G}})$. Previous query embedding vectors $[v_1, ..., v_{t-1}]$ are encoded as a context vector c_{t-1}, which is combined with the current query embedding vector v_t and fed to the LSTM memory cell at time t. This allows the LSTM to find an optimal combination of signals from prior context and the current query. For sessions with a single underlying intent (i.e., the user is consistently looking for a specific program), the model can learn the relatedness between successive queries and continuously reinforce confidence in the true intent.

In reality, context is sometimes irrelevant, which might introduce noise. When the context diverges too much from the current query embedding, the model should be able to ignore the noisy context signals to reduce their negative impact.

I adopt a many-to-many hierarchical architecture. The query embedding layer $\mathbb{F}(\cdot)$ and the classification layer $\mathbb{H}(\cdot)$ are applied to each query for program prediction at each time step t. We hope to find the true user intent as early as possible to reduce interactions between the user and our product. The parameters of the query embedding layer $\mathbb{F}(\cdot)$, as well as the classification layer $\mathbb{H}(\cdot)$, are shared by all queries regardless of their position in the session. For instance, two identical queries with different positions in a session will have the same query embedding vector. Except for the contextual layer $\mathbb{G}(\cdot)$, all other modules (e.g., query embedding, fully-connected layer, softmax layer, loss function) remain the same as in the basic (context-independent) model.

The training process for this model (Algorithm 2) starts with forward predictions for multiple queries in the session (Lines 2-7). Similar to Algorithm 1, only the last LSTM state h_m is selected as the query embedding vector (Line 6). Since sessions can have a variable number of queries, I dynamically clone a new LSTM to ingest the query at time t (i.e., LSTM[t] in Line 5) when the arriving query results in a longer session than all previously seen sessions. Line 8 utilizes another LSTM model to compute the context from sequential query embeddings. Lines 9-18 perform forward predictions and backward propagations for multiple queries in the classification layer. The queries are processed in a sequential manner such that for each query all forward operations are immediately followed by all backward opera-

Algorithm 2 Training the Full Context Model

```
1: for each session $s_i$ in the dataset $i = 1...|D|$ do
2:     $v$ = zeros($|s_i|$, lstm_size)                          ▷ query embedding vectors
3:     for each query $q_{i_t}$ with $t = 1...|s_i|$ do
4:         $e_{i_t}$ = encode($q_{i_t}$)
5:         $h_{1,...,m}$ = LSTM[t]:forward($e_{i_t}$)
6:         $v_t = h_m$
7:     end for
8:     $c_{1,...,|s_i|}$ = C_LSTM:forward($v_{1,...,|s_i|}$)                ▷ contextual vectors
9:     grad_linear = zeros($|s_i|$, lstm_size)
10:    for each query $q_{i_t}$ with $t = 1...|s_i|$ do
11:        $l_2$ = FC:forward($c_t$)
12:        $o$ = softmax:forward($l_2$)
13:        loss = criterion:forward($o, p_i$)
14:        session_loss = session_loss + $loss$
15:        grad_criterion = criterion:backward($o, p_i$)
16:        grad_soft = softmax:backward($l_2$, grad_criterion)
17:        grad_linear[$t$] = FC:backward($c_t$, grad_soft)
18:    end for
19:    grad_context = C_LSTM:backward($v_{1,...,|s_i|}$, grad_linear)
20:    for each query $q_{i_t}$ with $t = 1...|s_i|$ do
21:        grad_lstm = zeros($m$, lstm_size)
22:        grad_lstm[$m$] = grad_context[$t$]
23:        LSTM[t]:backward($e_{i_t}$, grad_lstm)
24:    end for
25:    update_parameters()
26: end for
```

tions before moving to the next query. Lines 19-24 propagate the gradients through the contextual and embedding LSTMs. Line 25 updates model parameters for each session by optimizing the session loss in Line 14.

It is important to note that the prediction task is applied at the query level: my model tries to predict the program after *each* query in the session. The alternative is to optimize for program prediction given *all* queries in the session—this is a much easier task, since the entire session has been observed. However, it defeats the purpose of the initial setup since we wish to satisfy viewer intents as quickly as possible.

6.3.5 Constrained Context Model

Finally, I explore a variant called the *constrained* context model. The model architecture is the same as the full context model (Figure 6.3). The difference, however, lies in how we learn the model. For the constrained context model, I adopt a pre-training strategy as follows: I first train the basic model (Algorithm 1) and then use the learned LSTM parameters to initialize the constrained context model's query embedding layer. The embedding layer is then fixed and purely used for generating query embeddings. That is, Lines 20-24 are removed from Algorithm 2.

The intuition behind this model is to restrict the search space during model training, aiming to reduce the complexity of optimization compared to the full context model. Whether this reduction in optimization complexity is beneficial to the prediction task is an empirical question I study in the following sections.

6.4 Experimental Setup

6.4.1 Data Preparation

I collected voice queries submitted to Xfinity X1 remote controllers during the week of Feb. 22 to 28, 2016, a total of 32.3M queries from 2.5M unique viewers. Based on preliminary analyses, I selected 45 seconds as the threshold for dividing successive queries into sessions, yielding 20.0M sessions.

To build a training set for supervised learning, we need the true user intent for each session. I automatically extracted noisy labels by examining what the viewer watched after the voice session; this exactly parallels inferring user intent from clickthrough data in the web domain. If the viewer began watching a program p at most K seconds after the last query in the voice session v and kept watching it for at least L seconds, I label the session with p. The selection of K and L represents a balance between the quantity and quality of collected labels. After some initial exploration, I set K to 30 seconds and L to 150 seconds, which yields a good balance between data quantity and quality (based on manual spot-checking). Using these parameters, I extracted 13.0M session-program pairs. These sessions contain mixed modalities, as in reality users navigate with a combination of voice queries and keypad entry.

Without any further processing, these voice queries might reflect arbitrary intent (e.g., “closed caption on”, “the square root of eighty one”, or “change to channel 36”). In order to limit ourselves to voice sessions with a single clear intent, I used two heuristics as follows: First, I define a way to reliably predict whether a query is program-related (i.e., the query is primarily associated with a TV series, movie, video, or sports program). This is obtained from the deployed X1 platform, which categorizes every query into one of many action types based on existing hand-crafted patterns and partial string matching. A query is program-related if it is categorized as one of the following: {SERIES, MOVIE, MUSICVIDEO, SPORTS}. Based on this knowledge, I restricted the data to sessions in which over 2/3 of queries are program-related and the final query in the session is also program-related. Since channel changes are a large portion of the data, this reduces the number of labeled pairs to 2.1M.[1]

Second, I computed the normalized edit distance between each query pair in the session, and kept only those sessions where *any* pair of queries has a distance less than 0.5. The goal here is to ensure that there is at least *some* cohesion in the sessions. This heuristic has a relatively minor effect: the resulting filtered dataset contains 1.96M sessions in total.

Naturally, data pre-processing plays an important role in any problem setup, and one might wonder if the above heuristics might generate a dataset that favors my proposed approach. This is unlikely because the data filtering is a function of two independent decisions: (i) the action types are chosen by the currently-deployed system, and (ii) viewing behaviors after the voice sessions are purely dependent on users. Both decisions are unrelated to my methods. Furthermore, as we are ultimately interested in improving the production system, it is only natural to bootstrap off existing log data, much like the development of web search.

From this data, I sampled five splits: a training set used in all experiments, and two groups of development and test sets. The first development and test sets contain only single-query sessions, called SingleDev and SingleTest. These are used to study whether the context-based models hurt accuracy in sessions without context. The second group contains only multiple-query sessions (i.e., at least two queries in each session), called MultiDev and MultiTest. To build the global set of programs Φ, I only kept a program if there are at least 50 associated sessions in the training set,

[1]Although channel changes can be handled like programs, I decided to ignore them due to the additional complexity of regional variations (e.g., HBO West and HBO East are two different channels and cannot be disambiguated solely based on the query).

Dataset	sessions	queries	avg. session len	avg. query len
Train	126016	181058	1.44	2.34
SingleDev	24792	24792	1.00	2.40
SingleTest	24572	24572	1.00	2.36
MultiDev	28427	82828	2.91	2.30
MultiTest	28173	82272	2.92	2.30

Table 6.1: Dataset statistics.

yielding 471 programs that account for 77.8% of all the viewing behaviors of the 1.96M filtered sessions during that week. Statistics for each of the splits after all pre-processing are summarized in Table 7.2.

6.4.2 Model Training

In total, we have three options for query representation, *char*, *word*, and *combined* (Chapter 6.3.2), and three options for the model, *basic*, *full context*, and *constrained context* (Chapters 6.3.3-6.3.5). Therefore, we have a total of nine experimental settings, by crossing the three representations with the three models.

The entire dataset contains 80 distinct characters in total, which means that the size of the one-hot vector used in the *char* setting is 80. For the word representation, I used 300-dimensional GloVe word embeddings [118] to encode each word, which is trained on 840 billion tokens and freely available. The word vocabulary of the dataset is 20.4K, with 1759 words not found in the GloVe word embeddings. Unknown words were randomly initialized with values uniformly sampled from [-0.05, 0.05].

During training, I used the stochastic gradient descent algorithm together with RMS-PROP [141] to iteratively update the model parameters. The learning rate was initially set to 10^{-3} and then decreased by a factor of three when the development set loss stopped decreasing for three epochs. The maximum number of training epochs was 50. For the constrained context model, the number of pre-training epochs was selected as 15. The output size of the LSTMs was set to 200 and the size of the linear layer was set to 150. The regularization weight λ was chosen as 10^{-4}. At test time, I selected for evaluation the model that obtained the highest P@1 accuracy on the development set. My models were implemented using the Torch framework. I ran all experiments on a server with two 8-core processors (Intel Xeon E5-2640 v3 2.6GHz) and 1TB RAM, with each experiment running on 6 CPU threads.

6.4.3 Baselines

My methods were compared against the following baselines:

TF-IDF/BM25: I built a 3-gram (character-level) inverted index of the program set Φ. During retrieval, the matching score is computed on 3-gram overlaps between query and programs using TF-IDF or Okapi BM25 weighting ($k_1 = 1.2$, $b = 0.75$).

Learning-to-rank SVM$^{\text{rank}}$ [142]: I first used BM25 to retrieve at most 20 candidate programs per query, then designed three types of features for each pair of query and candidate program: (1) BM25 score, (2) max/mean/min value of cosine similarities between word embeddings of the query and the candidate program, and (3) popularity score of the candidate program.

DSSM [11]: This neural ranking model for web search uses word hashing to model interactions between queries and programs at the level of character 3-grams. This method is an appropriate baseline for the problem since it can handle ASR transcription errors, unlike neural ranking models based on word matching [10, 46, 91].

DSSM+S: I train and evaluate DSSM by concatenating queries in one session with a special boundary token between neighboring queries. This variant can be considered a context-aware baseline, where I aim to study whether simple concatenation is able to capture context signals in a session.

Basic: I applied the basic model described in Chapter 6.3.3 with character-level, word-level, and combined representations.

Basic+S: Same as above, with a concatenated query representation.

6.5 Experimental Results

I used four metrics in the evaluation, averaged over all queries: precision at one (P@1), precision at five (P@5), Mean Reciprocal Rank (MRR), and Query Reduction (QR). The first three are standard retrieval metrics and don't require an explanation. QR is a measure of how many queries a viewer has "saved" in a session. For a session with n queries, the number of reductions is $n - i$ if the model returns the correct prediction at the i-th query, which means that the viewer does not need to issue the next $n - i$ queries, hence a *reduction* of $n - i$. I average this metric over all sessions.

There are few important questions the experiments are designed to answer: First, how do my proposed models compare against lexical overlap, a learning-to-rank baseline, and an existing neural ranking model? Second, within my model,

ID	Model	Query	P@1	P@5	MRR
1	TF-IDF	3-gram	0.895^{10}	0.937^{10}	0.923^{10}
2	BM25	3-gram	$0.907^{1,10}$	0.939^{10}	0.927^{10}
3	SVM$^{\text{rank}}$	-	$0.917^{1,2,10}$	$0.951^{1,2,10}$	$0.933^{1,9}$
4	DSSM	3-gram	$0.918^{1,2,10}$	$0.949^{1,10}$	$0.931^{1,10}$
5	DSSM+S	3-gram	$0.916^{1,10}$	$0.949^{1,10}$	$0.936^{1,10}$
6	Basic	char	$0.944^{1\text{-}5,9,10}$	$0.953^{1,2,9,10}$	$0.962^{1\text{-}5,10}$
7	Basic	word	$0.943^{1\text{-}5,9,10}$	$0.953^{1,2,9,10}$	$0.962^{1\text{-}5,10}$
8	Basic	comb	$\mathbf{0.947}^{1\text{-}5,9,10}$	$\mathbf{0.955}^{1,2,9,10}$	$0.964^{1\text{-}5,10}$
9	Basic+S	comb	$0.923^{1,2,10}$	0.938^{10}	$0.953^{1\text{-}5,10}$
10	Context-f	char	0.753	0.794	0.837
11	Context-f	word	$0.926^{1,2,10}$	0.942^{10}	$0.959^{1\text{-}5,10}$
12	Context-f	comb	$0.932^{1,2,10}$	0.947^{10}	$\mathbf{0.967}^{1\text{-}5,9}$
13	Context-c	char	$0.938^{1\text{-}5,9,10}$	$0.950^{1,9,10}$	$0.963^{1\text{-}5,10}$
14	Context-c	word	$0.943^{1\text{-}5,9,10}$	$0.950^{1,9,10}$	$0.961^{1\text{-}5,10}$
15	Context-c	comb	$0.944^{1\text{-}5,9,10}$	$0.953^{1,2,9,10}$	$0.963^{1\text{-}5,10}$

Table 6.2: Model effectiveness on *single-query* sessions. The second column denotes the model: baselines compared to the basic, full context (Context-f), and constrained context (Context-c) models. The third column indicates the query representation. Remaining columns show evaluation metrics. Superscripts indicate the row indexes for which a metric difference is statistically significant at $p < 0.01$. Rows are numbered in the first column for convenience.

which is the most effective query representation (character, word, or combined)? Finally, what are the important differences between single query and multiple-query sessions, and which is the most effective context model: ignoring context, query concatenation, full context, or constrained context?

Results for the single-query and multiple-query sessions, on the SingleTest and MultiTest splits, respectively, are shown in Table 6.2 and 6.3. Each row represents an experimental condition (numbered for convenience). The second column specifies the model: "Context-f" and "Context-c" denote the full and constrained context models, respectively. The third column indicates the query representation (char, word, or combined), and the remaining columns list the various evaluation metrics. Superscripts indicate the row indexes for which a metric difference is statistically significant ($p < 0.01$) based on Fisher's two-sided, paired randomization test [1]. A dash symbol "-" connecting two indices "*a*-*b*" is shorthand for $a, \ldots, b$.

Let's first consider the non-context baselines (numbered 1–4 and 6–8): TF-IDF and BM25 achieve fairly high accuracies (P@1 of 0.895 and 0.907) on single-query sessions. The effectiveness of these two simple baselines is not surprising, since single-query sessions are by construction the "easy queries" in which users reach their intended program in only a single voice query (but may include keypad navigation). The SVM$^{\text{rank}}$ predictor achieves better accuracy than the BM25 baseline because it also takes advantage of word embeddings to consider semantic relatedness as well as the popularity priors of programs in a supervised setting. DSSM achieves comparable performance to the SVM$^{\text{rank}}$ approach, significantly outperforming both TF-IDF and BM25. We also notice that the basic (context-independent) model outperforms DSSM by quite a bit. I identified two main reasons: (1) There are about 2% sports-related queries in which the user searched for a particular sports team (e.g., "Detroit Red Wings") but ended up watching a program (e.g., "NHL Hockey") that shares no common 3-grams with the query; (2) Major ASR errors sometimes results in very little lexical overlap between the query and the intended program title (e.g., "Dr. Seuss's The Lorax" is transcribed as "The Laura").

Turning our attention to non-context models on the multiple-query sessions in Table 6.3, we see that accuracy drops significantly compared to single-query sessions. The relative effectiveness of the various methods is consistent, but we observe a much wider range of prediction accuracy separating the simple methods (TF-IDF and BM25) from the more sophisticated models (DSSM and the basic model). These

ID	**Model**	**Query**	**P@1**	**P@5**	**MRR**	**QR**
1	TF-IDF	3-gram	0.518^{10}	0.593^{10}	0.543^{10}	0.932^{10}
2	BM25	3-gram	$0.533^{1,10}$	0.596^{10}	$0.565^{1,10}$	$0.947^{1,10}$
3	SVM$^{\text{rank}}$	-	$0.547^{1,2,10}$	$0.621^{1,2,10}$	$0.582^{1,2,10}$	$0.962^{1,2,10}$
4	DSSM	3-gram	$0.568^{1,2,3,10}$	$0.617^{1,2,10}$	$0.584^{1,2,10}$	$1.001^{1,2,3,4,10}$
5	DSSM+S	3-gram	$0.550^{1,2,10}$	$0.618^{1,2,10}$	$0.576^{1,2,10}$	$0.972^{1,2,10}$
6	Basic	char	$0.605^{1\text{-}5,10}$	$0.647^{1\text{-}5,10}$	$0.690^{1\text{-}5,10}$	$1.108^{1\text{-}5,10}$
7	Basic	word	$0.609^{1\text{-}5,10}$	$0.644^{1\text{-}5,10}$	$0.677^{1\text{-}5,10}$	$1.086^{1\text{-}5,10}$
8	Basic	comb	$0.614^{1\text{-}5,9,10}$	$0.651^{1\text{-}5,10}$	$0.687^{1\text{-}5,10}$	$1.113^{1\text{-}5,9,10}$
9	Basic+S	comb	$0.596^{1\text{-}5,10}$	$0.645^{1\text{-}5,10}$	$0.697^{1\text{-}5,10}$	$1.061^{1\text{-}5,10}$
10	Context-f	char	0.482	0.532	0.580^{1}	0.856
11	Context-f	word	$0.599^{1\text{-}5,10}$	$0.638^{1\text{-}5,10}$	$0.687^{1\text{-}5,10}$	$1.075^{1\text{-}5,10}$
12	Context-f	comb	$0.598^{1\text{-}5,10}$	$0.643^{1\text{-}5,10}$	$0.688^{1\text{-}5,10}$	$1.039^{1\text{-}5,10}$
13	Context-c	char	$0.639^{1\text{-}12}$	$0.684^{1\text{-}12}$	$0.731^{1\text{-}12}$	$1.117^{1\text{-}7,9\text{-}12}$
14	Context-c	word	$0.639^{1\text{-}12}$	$0.683^{1\text{-}12}$	$0.729^{1\text{-}12}$	$1.112^{1\text{-}7,9\text{-}12}$
15	Context-c	comb	$\mathbf{0.643}^{1\text{-}12}$	$\mathbf{0.687}^{1\text{-}12}$	$\mathbf{0.734}^{1\text{-}12}$	$\mathbf{1.128}^{1\text{-}12}$

Table 6.3: Model effectiveness on *multiple-query* sessions. The second column denotes the model: baselines compared to the basic, full context (Context-f), and constrained context (Context-c) models. The third column indicates the query representation. Remaining columns show evaluation metrics. Superscripts indicate the row indexes for which a metric difference is statistically significant at $p < 0.01$. Rows are numbered in the first column for convenience.

results suggest that neural network models are able to better capture signals beyond lexical overlap and the few other manually-defined features I employed.

For the first question, we observe that in the basic and constrained context models, the word-level query representation is quite close to the character-level query representation. However, in nearly all conditions, across nearly all metrics, the combined condition further improves (albeit only slightly) upon both representations, which shows that character-level and word-level representations provide signals that supplement each other.

In terms of the context models, the constrained context model significantly outperforms all others, including the basic, query concatenation, and full context models. We observe that query concatenation hurts the effectiveness of the DSSM and basic models. From the output logs, I find that query concatenation is less robust to noise in the session data, both at training and at inference time. For example, consider sessions beginning with a general-purpose query such as “Find me all free movies on demand”: the context of all subsequent queries will be “polluted” by irrelevant information, from which the model might not be able to recover.

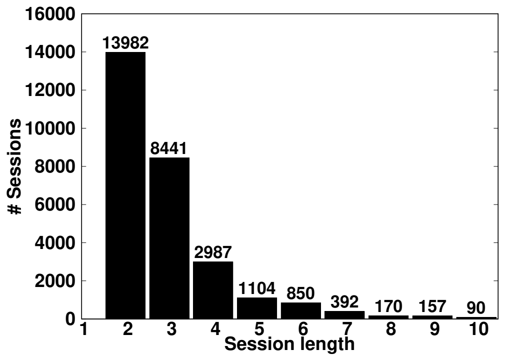
In contrast, modeling query dependencies through an LSTM model is more effective. Since the context models copy their query embedding layers from the basic models, we conclude that the upper LSTM layer is able to capture contextual information to improve prediction. Note that the full and constrained models share exactly the same architecture; the only difference lies in whether or not we back-propagate to the query embedding layer during training—the constrained model is designed to restrict the model search space. The effectiveness gap on the multiple-query session dataset (0.599 vs 0.643) demonstrates that the query embedding layer obtained by the constrained context model through pre-training is of higher quality. This is likely due to insufficient data for the full context model to effectively learn parameters for both LSTM levels. However, a caveat: it is conceivable that with even more training data, the full context model will improve. But as it currently stands, the constrained context model displays a better ability to exploit contextual information for predicting viewers’ intent. Overall, for the multiple-query sessions, the constrained model with the combined representation yields a 21% relative improvement over BM25 and 13% over DSSM in terms of P@1.

For single-query sessions, we want to make sure that the context models do not “screw up” these queries. I confirm that this is indeed the case. It is no surprise that the basic model performs the best on single-query sessions: since there is no context to begin with, all the “contextual machinery” of the richer models can only serve as a distraction. I find that the constrained model with the combined representation (best condition above) still performs well—slightly worse than the basic model with the combined representation, although the differences are not statistically significant. It is also interesting to note that the full context model with the character representation is terrible, suggesting that the search space is too large given the combination of longer query representations and session-level optimization.

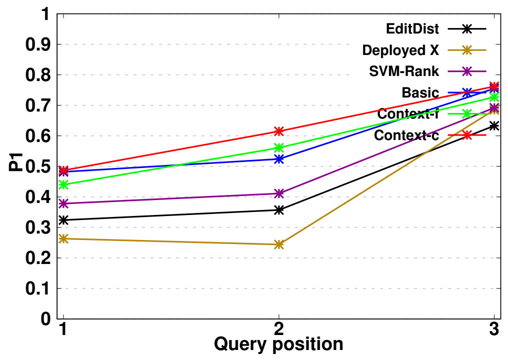
6.5.1 Context Analysis

To better understand how my models take advantage of context, I focused on multiple-query sessions and examined how accuracy evolves during the course of a session. Results are shown in Figure 6.4. The leftmost plot shows the histogram of session lengths (i.e., number of queries in each session) in the MultiTest split; each bar is annotated with the actual count. In Figures 6.4(b)-(d), I show the average P@1 score from MultiTest at different positions in the session (on the x axis), i.e., at the first query in the session, the second query, etc. For illustrative purposes I focus on “short” sessions with a length of three (8441 sessions), “medium” sessions with a length of six (850 sessions), and “long” sessions with a length of nine (157 sessions). For clarity, in all cases the models used the combined query representation.

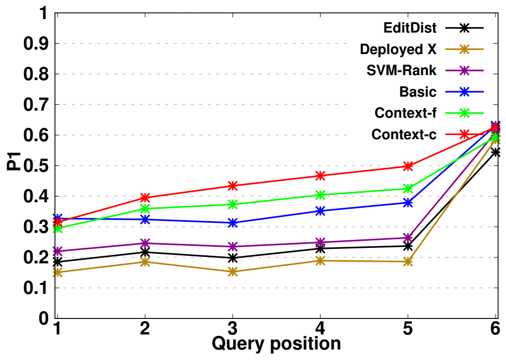
We observe several interesting patterns in Figures 6.4(b)-(d). First, for the non-context models (SVM$^{\text{rank}}$, DSSM, and basic), the accuracy stays flat until the



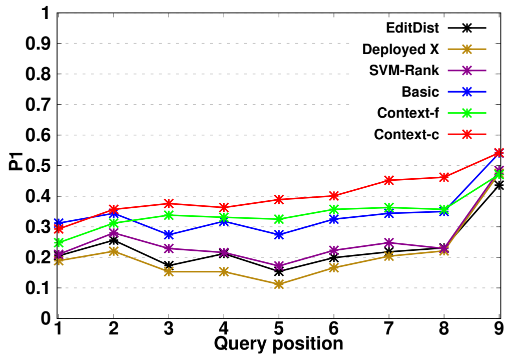
(a) Session Length Distribution



(b) Short Sessions



(c) Medium Sessions



(d) Long Sessions

Figure 6.4: Context analysis of the multiple-query session (MultiTest) test set. The leftmost plot shows the distribution of session lengths. Subfigures (b)-(d) show the average P@1 score at different positions (i.e., the i-th query) in the session.

final query (except small fluctuations due to noise). Accuracy for the final query rises significantly because the viewer finally found what she was looking for (and thus is likely to be an "easy" query). Also, we can see that DSSM with the concatenated query representation (DSSM+S) is not able to capture context clues in the sessions, illustrated by a curve as flat as the non-context models. The Basic+S model is interesting: At the beginning of sessions, it is able to outperform the basic model, yet its effectiveness degrades as the session progresses. At the end of sessions, it is consistently worse than the basic model. This suggests that simple concatenation is prone to accumulating too much noise during longer sessions, eventually outweighing the benefits of having context. However, for the context-aware models (Context-f and Context-c), we observe a consistent increase in the accuracy curves as the session progresses. This demonstrates that the LSTM model is able to better capture context signals in the sessions, and as the model accumulates more context, it can better identify the user's true intent. The constrained context model performs about the same as the basic model at the first query (where there is no context) and the final query (which is easier since the viewer found the desired program). However, the constrained context model outperforms the basic model in the middle of sessions, highlighting the ability of the LSTM to capture context.

In Table 6.4, I provide two real example sessions to illustrate how each model responds to the sequence of viewer queries. The session is shown in the first row, where each query is separated by a colon. The second row shows the viewer's intent (i.e., ground truth label). The remaining rows show the output of each model; due to space limitations, I only show the top predictions along with their confidence scores for my models. Each prediction in the sequence is also separated by a colon. To save space, I use the symbol $\star$ to indicate that the prediction is correct.

In the first example (left), the viewer is consistently looking for the program "Caillou", but the production system fails three times in a row due to ASR errors. For the first three queries, all models based on n-gram matching fail (BM25, SVM$^{\text{rank}}$, DSSM, DSSM+S) because there is no 3-gram overlap between the queries and the intended program. For my models, both the basic/char and Context-c/char models can predict the correct program from the query "Cacio" with high confidence. However, models with word-level representations all fail for this query. This

Session		Cacio : You : You : Caillou	Sienna cover : Color : Casey undercover
Program		Caillou	K.C. Undercover
Model	**Query**	**Example 1**	**Example 2**
BM25	-	Pacific Rim : Now You See Me : Now You See Me : $\star$	Recovery Road : College Basketball : $\star$
SVM^{rank}	-	Pacific Rim : Now You See Me : Now You See Me : $\star$	Recovery Road : Dora the Explorer : $\star$
DSSM	3-gram	Pacific Rim : Young : Young : $\star$	$\star$: College Basketball : $\star$
DSSM+S	3-gram	Pacific Rim : The Young and the Restless : The Young and the Restless : $\star$	$\star$: College Basketball : $\star$
Basic	char	$\star$ (0.81) : $\star$ (0.80) : $\star$ (0.80) : $\star$ (1.0)	$\star$ (0.76) : Carolina (0.07) : $\star$ (0.99)
Basic	word	Child Genius (0.03) : $\star$ (0.57) : $\star$ (0.57) : $\star$ (1.0)	Recovery Road (0.48) : $\star$ (0.08) : $\star$ (0.75)
Basic	comb	Paw Patrol (0.17) : $\star$ (0.83) : $\star$ (0.83) : $\star$ (1.0)	$\star$ (0.37) : Magic Mike XXL (0.31) : $\star$ (0.98)
Basic+S	comb	Paw Patrol (0.15) : Paw Patrol (0.25) : $\star$ (0.75) : $\star$ (0.98)	$\star$ (0.34) : $\star$ (0.20) : $\star$ (0.85)
Context-f	char	Lego Ninjago (0.30) : $\star$ (0.79) : $\star$ (0.90) : $\star$ (0.99)	$\star$ (0.43) : $\star$ (0.67) : $\star$ (0.89)
Context-f	word	Paw Patrol (0.30) : $\star$ (0.62) : $\star$ (0.98) : $\star$ (1.0)	$\star$ (0.29) : $\star$ (0.65) : $\star$ (1.0)
Context-f	comb	Lego Ninjago (0.03) : $\star$ (0.60) : $\star$ (0.98) : $\star$ (1.0)	$\star$ (0.41) : $\star$ (0.54) : $\star$ (0.99)
Context-c	char	$\star$ (0.96) : $\star$ (0.99) : $\star$ (0.99) : $\star$ (1.0)	$\star$ (0.81) : $\star$ (0.96) : $\star$ (0.99)
Context-c	word	Wallykazam (0.07) : $\star$ (0.59) : $\star$ (0.86) : $\star$ (1.0)	$\star$ (0.89) : $\star$ (0.80) : $\star$ (1.0)
Context-c	comb	Paw Patrol (0.17) : $\star$ (0.93) : $\star$ (1.0) : $\star$ (1.0)	$\star$ (0.65) : $\star$ (0.83) : $\star$ (0.97)

Table 6.4: Two sample sessions and top predictions for each model. Each query and prediction in the session is separated by a colon. For each prediction from my models, I show the confidence score. $\star$ indicates that the model response was correct.

is no surprise as the word "Cacio" is a rare mis-transcription of the word "Caillou" and thus rarely seen in the training set. For the next two successive queries "You", the basic models are able to succeed with high confidence scores. However, the Basic+S model remains mistaken after the first occurrence of "You", suggesting that concatenation with the previous query "Cacio" serves as a distractor. For both the full and constrained context models, confidence for the second query "You" is higher (thanks to context). This is an example of how contextual clues help, confirming our intuition. Since the second example behaves similarly, I omit a description for space consideration.

6.5.2 Efficiency Analysis

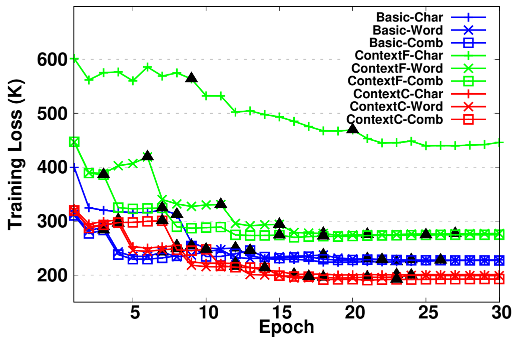
The final set of experiments examined efficiency in terms of training time and prediction latency, both important consideration for production deployments. Results are shown in Table 6.5. For each setting in the first two columns, "#Params" shows the total number of parameters in the model, "Training" denotes the training time for each epoch, and "Test" shows the prediction latency per query. "Avg." and "Conf." indicate the average value and the 95% confidence interval of training/test times over 30 epochs. Overall, the training time of all models is less than around 100 minutes per epoch, and the per-query prediction latency is less than 8 milliseconds. Most model configurations converge in the first 20 epochs.

Comparing different query representations, we observe that *combined* has the most number of parameters and was also the slowest to train and test; this is no surprise. For the character-level representation, the size of the one-hot vectors is smaller, resulting in fewer parameters at the query embedding layer. However, the number of characters in a query is much larger, leading to longer training times. With the same query representation, the full context models have more parameters and take longer to train than the corresponding basic and constrained context models, as expected. The constrained context models have the same number of parameters and similar prediction latencies as the full context models since they share the same architecture. However, the constrained context models are much faster to train, suggesting that most of the training effort is spent on the query embedding layer in the full context models.

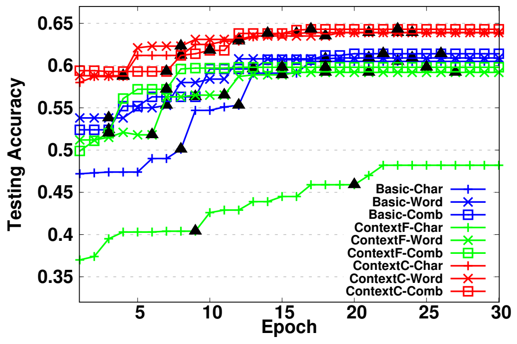
I plot training loss and testing accuracy curves in Figure 6.5: (a) shows the training loss curve as a function of epoch, (b) shows the P@1 curve in the MultiTest set at each epoch. The symbol ▲ denotes the epochs where the learning rate was divided by three because development loss had not decreased for three epochs. We see that most models converged within 20 epochs. In the basic and full context models, the *char* representation took longer to converge than *word* or *combined*,

Model	Query	#Params	Training (min) Avg.	Training (min) Conf.	Test (ms) Avg.	Test (ms) Conf.
Basic	char	326,871	62.1	[59.7, 64.7]	6.4	[6.2, 6.6]
Basic	word	502,871	32.6	[32.2, 33.0]	3.0	[3.0, 3.1]
Basic	comb	758,471	94.5	[91.4, 97.2]	6.9	[6.6, 7.0]
Context-f	char	648,471	72.1	[68.4, 74.5]	6.6	[6.4, 7.0]
Context-f	word	824,471	58.8	[57.2, 60.8]	4.0	[4.0, 4.1]
Context-f	comb	1,210,071	102.4	[100.8, 103.8]	7.0	[6.8, 7.2]
Context-c	char	648,471	32.1	[30.9, 33.7]	6.6	[6.4, 6.8]
Context-c	word	824,471	30.1	[29.5, 31.2]	4.0	[3.9, 4.1]
Context-c	comb	1,210,071	42.5	[41.8, 43.1]	6.9	[6.8, 7.0]

Table 6.5: Model efficiency: Column “Training” shows the training time per epoch and “Test” shows prediction latency per query. “Avg.” indicates the average value of training/test times, and “Conf.” indicates the 95% confidence interval.



(a) Training Loss



(b) Testing Accuracy

Figure 6.5: Training loss and testing accuracy for each epoch; ▲ denotes epochs where the learning rate was reduced.

which shows that a character-level representation is more difficult to learn. It is also interesting that the gap in training loss between the basic and full context models is larger than the gap in test accuracy, which means that although the full context model is difficult to train, the benefit of context enables it to generalize well. The constrained context model walks a middle ground in terms of model complexity and the ability to capture context information, leading to both lower training loss and higher test accuracy.

6.6 Production Deployment

Following the above promising laboratory experiments, I collaborated with the engineering team in Comcast and packaged the constrained context model with word-level inputs, which we term as HRNN in abbreviation of hierarchical recurrent neural network model, into a standalone software module that was deployed into production as part of the X1 software package on January 5, 2018. In this section, I describe deployment details and lessons learned from the first month of live traffic. Our experience provides a case study of technology transfer from research into production, detailed in my previous KDD paper [41].

6.6.1 Implementation Details

To balance efficiency, effectiveness, and coverage, my model was deployed in production as part of a cascade, where the HRNN module was run after a number of simpler NLP modules (based on pattern matching and some machine-learned components). A cascade architecture has several advantages: While it would have been possible to run the HRNN in parallel with the existing modules, this setup introduces a new ensemble selection problem that complicates deployment. Since we are introducing completely new technology (this is the first neural network model that has been deployed in production), we adopted a conservative approach to minimize adverse effects. Because the HRNN is placed at the end of the cascade, it is given queries that would have otherwise gone unhandled (more details below). Finally, a cascade deployment allows us to control query latencies and bring richer models to bear only when they are needed (e.g., there is no need to run a deep neural network to respond to the query "CNN"). This is similar in spirit to cascade architectures for ranking [143].

Implemented in Keras with the TensorFlow backend, my model has over 17M parameters, 15M of which come from the embedding matrix. The model is serialized into a 69 MB file and deployed as part of a Docker image. Inference for a single query takes around 70ms on a GPU server (Tesla M60 GPU, Xeon E5-2643 v4 CPU), while latency increases to 750ms on a server with a single Xeon E5-2660 v3 CPU. Since feedforward inference is embarrassingly parallel, we can scale up easily by load balancing across an arbitrary number of servers to obtain the desired throughput. At peak load, we use a small cache to skip model inference for the most frequent N queries, which lowers average latency considerably.

6.6.2 Query Coverage

Before we deployed the HRNN, the production system was unable to produce any response for 8% of all queries—in this case, the customer sees a special "cannot handle this query" message. Thus, the queries given to my model are the most difficult queries by construction. A manual analysis revealed numerous challenges: speech recognition errors, references to brand new programs, ambiguous intent, or even complete gibberish. By deploying my HRNN as the last step of the cascade, we hoped to handle as many of these queries as possible without making too many mistakes. As an additional control mechanism, we implemented a confidence threshold, so that if the model confidence (output of the softmax layer) is below this value, the HRNN does not return any response. This threshold, which was hand tuned, allows us to trade off coverage for precision.

After deployment, we monitored logs in the week from January 27 to February 2, 2018: the complete system received 70.1M queries, 5.7M of which (8%) were sent to the HRNN, the last step of the cascade. Among these 5.7M queries (for which the viewer would have gotten an error message previously), the model confidence was above the threshold for 4.2M (6%). In other words, the HRNN received 8% of the total traffic and responded to 6%; for the remaining 2%, my model chose not to handle the query, and the platform resorted to the existing behavior (displaying the error message).

Figure 6.6 shows the breakdown of per-module query coverage. The main pattern-based module is shown at the top (yellow) and the HRNN is shown in orange at the bottom. A few other specialized modules (sports, events, trivia questions, etc.) can be seen as small slivers. After the HRNN was deployed, unhandled queries (purple) gradually turned orange. The HRNN quickly became the second most impactful module in the production system.

Based on this, we can conclude that the coverage of the HRNN module is 74% (4.2M/5.7M). The coverage of the entire end-to-end system increased from 92% to 98% after deployment. In other words, the HRNN dramatically increased coverage, reducing the number of unhandled queries by three quarters.

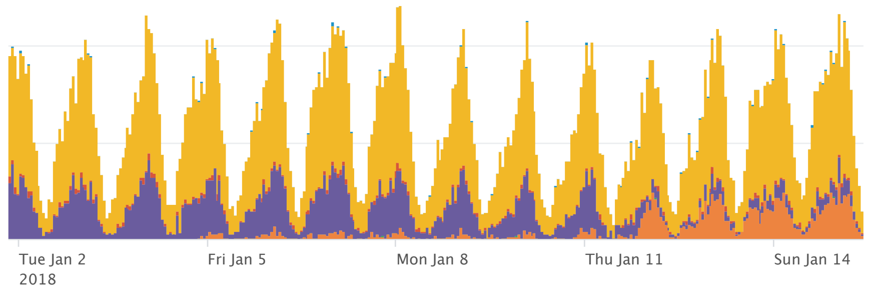


Figure 6.6: Production traffic after HRNN deployment. The fraction of queries served by the HRNN module (orange) gradually replaces queries without any responses (purple).

6.6.3 Quality Evaluation

In addition to coverage, we are also interested in accuracy: When the model generates a response, how good is it? And more importantly, what is the impact on the customer experience? Recall that these queries were previously not handled and the system responded with an error message. Therefore, any relevant response represents an improvement. Furthermore, it is unclear if a non-relevant response is actually worse than an error message.

To formally evaluate output quality, we devised a simple three-grade relevance scale: 1 means the response was completely not relevant, 2 means the response was somewhat relevant (i.e., there might be a better response, but the system output is reasonable), and 3 means the response was completely relevant.

Every week, our quality assurance team examines a random sample of queries that were sent to the HRNN and received a response: this resulted in a dataset of 809 annotated queries. The annotator listened to the audio and looked at the final output to determine its relevance. Results showed that 29% of responses were graded as completely relevant (e.g., query was "Missoula gumball" and the HRNN response was "The Amazing World of Gumball"), while 38% received a grade of 2, somewhat relevant (e.g., "Letterman" led to the movie "Dying to Do Letterman"). Only 33% were considered non-relevant. However, further analysis shows that these non-relevant responses were usually "interpretable" by viewers, e.g., an erroneous partial match ("Ally Wong" returned "Austin & Ally"). We did not observe many responses that were wildly off-base that would perplex the viewer.

In summary, for two thirds of queries that the HRNN provided a response (4.2M queries), the customer experience improved (since the alternative was an error message). In the remaining third, where the system provided a non-relevant response, arguably we haven't made anything worse. Considering these were the most difficult questions to begin with, we were extremely pleased with the coverage and accuracy of my model on live production traffic.

6.7 Conclusion

Our vision is that future entertainment systems should behave like intelligent agents and support voice interactions. As a first step, I tackle a specific problem, voice navigational queries, to help users find the TV programs they are looking for. I articulate the challenges associated with this task, which I tackle by modeling session contexts using hierarchically-arranged neural network modules. Results on a large real-world voice query log show that my methods can effectively cope with ambiguity and compensate for ASR errors. In addition, I helped deploy my model into the Xfinity entertainment system to serve millions of queries daily, demonstrating increased end-to-end query coverage and answer accuracy. Indeed, this work allows viewers to talk to their TVs, and for customers who learn of this feature for the first time, it is a delightful experience!

Chapter 7: Temporal Modeling with Multi-Task Learning for Voice Search

7.1 Introduction

In Chapter 6, I introduced my first attempt at tackling challenges focused on directly identifying the program a viewer intends to watch from a voice query, termed *voice query navigation*. The key insight is to exploit session context to disambiguate queries and to cope with speech recognition errors. For example, the query "game of throw" can either refer to the television series "Game of Thrones" (because of a transcription error) or a TV game called "Fish Throw Game". However, if the viewer just uttered "HBO series" a moment ago, then it is far more likely that she is looking for the former since we know the show is playing on HBO. This intuition is operationalized using a Hierarchical Recurrent Neural Network (HRNN) model.

The HRNN model was recently deployed into production to serve live traffic at the tail end of a cascade architecture, as part of a risk-averse deployment strategy. At present, the model serves millions of queries daily for which the previous modules provide no response (in other words, the most difficult queries). We have substantially increased end-to-end coverage, reducing the number of unhandled queries by three quarters. On these queries, the HRNN definitively improved the customer experience two thirds of the time and arguably did not hurt in the other third.

Despite the success of the HRNN in production, I noticed two main shortcomings. First, the model adopts a classification-based approach, which is unable to predict unseen programs (e.g., newly-added content). Furthermore, its formulation has difficulty handling the long tail of rarely-watched programs. Second, my analysis of millions of queries [43] reveals that they span the gamut from program navigation to vague entertainment intents (e.g., looking for kids cartoons) to direct commands (e.g., turning on closed captioning) to queries that have nothing to do with entertainment (e.g., checking the weather). In fact, I find that around 40% of queries are either ambiguous viewing intents or not related to viewing a program at all. Obviously, a model based on program prediction cannot handle such queries. These two main shortcomings motivated us to explore a different design.

To this end, I propose a novel multi-task neural architecture for query understanding that jointly performs three distinct tasks:

(1) **Program prediction** to directly identify the program or channel referenced in a viewer's utterance, out of a catalog of tens of thousands of programs and hundreds of channels.

(2) **Intent classification** to understand what the viewer wishes to do. Our system recognizes around one hundred intents, ranging from TV commands (record a particular show) to entertainment intents that vary in specificity to non-entertainment intents (e.g., how to troubleshoot the wifi connection).

(3) **Query tagging** of each token in a viewer's utterance with domain-specific labels such as "entity", "channel", "modifier", etc., drawn from a tag set of roughly a dozen.

Program prediction, intent classification, and query tagging work together in a complementary way. In cases where the decision overlaps—for example, the system detects that the viewer's intent is to switch channels, which is confirmed by the tagging and program prediction modules—multiple sources of evidence reinforce the system's confidence in the decision. In cases where program prediction fails, tagged tokens in the query can serve as keywords for searching the program catalog. For example, given the query "watch Tom Hanks movies on HBO", program prediction may fail since the viewer is not looking for a specific program. The system, however, can parse the query into a logical form via the query tags: [*person*="Tom Hanks" $\wedge$ *category*="movies" $\wedge$ *channel*="HBO"] and return a list of options to the viewer.

I evaluate my multi-task model in a carefully-controlled setting on large real data, demonstrating effectiveness gains beyond my HRNN model and other competitive baselines. More importantly, the multi-task problem formulation provides a unified framework for understanding voice queries that express a multitude of intents.

This chapter makes the following contributions:

- I provide a descriptive data analysis of viewers' voice queries from multiple perspectives, including a number of standard measures such as query frequency, query/session length, etc. In addition, I propose a taxonomy of user intents and explain the need for fine-grained domain-specific query tagging.

- I articulate a novel framework for understanding voice queries posed to an entertainment platform, decomposed into the three tasks of program prediction, intent classification, and query tagging. In particular, I explain why all three are necessary to properly understand queries.

- I describe a neural architecture that *jointly* learns how to perform all three tasks, explaining the intuition behind my design choices. Evaluation on a large voice query log demonstrates how joint learning of the three tasks improves accuracy on each task individually. The multi-task model provides the basis of an end-to-end system for handling queries that can draw from approximately one hundred different intents.

This chapter is organized as follows: first I present a comprehensive log analysis over 81M real voice queries in Chapter 7.2, attempting to answer the question "what do viewers say to their TVs?". Motivated by the data analysis, I propose a novel multi-task framework to interpret users' voice queries in Chapter 7.3. Then I present the evaluation of the multi-task model on large-scale query logs in Chapter 7.4. Finally I conclude this work in Chapter 7.5.

7.2 Voice Log Analysis

In this section, I present an analysis of log data collected from the Comcast Xfinity X1 platform during the week of Feb. 22 to 28, 2017. The dataset contains 81.4M voice queries from 8.1M unique devices.

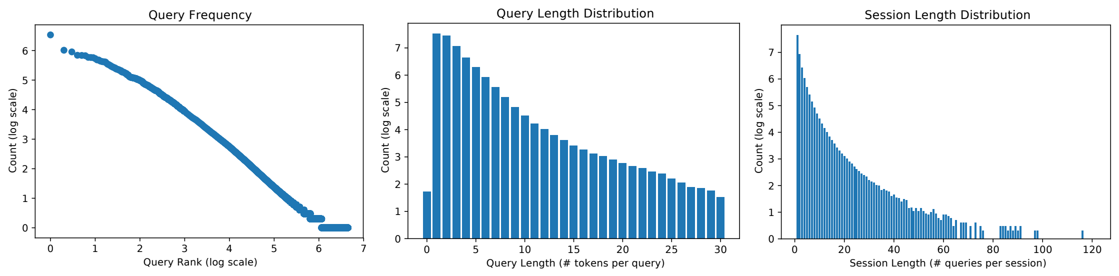


Figure 7.1: Characteristics of voice queries directed at entertainment systems: distribution of query frequency (left), query length (middle), and session length (right).

Recall that our system receives as input the one-best result of a black-box third-party ASR system, which is a text string. We do not have access to transcription lattices or n-best lists. Although the ASR system is specifically tuned to our domain, it needs to recognize millions of program titles, hundreds of thousands of person names, and tens of thousands of sports teams, all of which overlap with each other. Television content is often very localized, e.g., a viewer wants to watch local sporting event with the “Augsburg Auggies”, making domain adaptation difficult.

Another challenge is the diversity of customers in terms of age, ethnicity, etc. For example, I have observed that many ASR errors come from kids wanting to access their favorite cartoon; see Liao et al. [144] for a summary of ASR challenges with children.

Finally, it is important to recognize that this analysis represents a (recent) snapshot in time. The model deployed today has been improved, and there is always a co-evolution of system capabilities and customer queries.

7.2.1 Query and Session Lengths

Out of the 81.4M voice queries, there are 4.46M unique queries, indicating that despite the presence of frequent head queries (e.g., “CNN”), there is plenty of linguistic diversity in the data. A query has 1.96 tokens and 9.70 characters on average, and the number of unique tokens is 199K (constrained by the ASR system). Around 7.4% of tokens are out of vocabulary (OOV) with respect to the Google News corpus used to train word2vec [92]; 13.8% queries have OOV words. Most OOV words are due to a mismatch between the vocabulary of the ASR system and our text processing tools.

Figure 7.1 presents three views of the dataset. The left panel shows a standard log–log (base 10) plot of query frequencies. The top five most frequent queries are “Netflix”, “CNN”, “Fox News”, “ABC”, and “free movies”, uttered by viewers hundreds of thousands of times per week. In the tail, we observe 3.3M unique queries. Unsurprisingly, the distribution is Zipfian. I examine query intents in more detail in Chapter 7.2.2, but note here that in addition to channel names and favorite apps, some of the most frequent queries are intended for browsing the catalog, where the viewer does not have a specific program in mind; “free movies”, “on demand”, and “movies” are among the top 20 in terms of frequency.

The center panel shows the distribution of query lengths in terms of the number of tokens; for clarity, I only show queries with lengths up to 30 tokens. After removing punctuations and normalizing text, around 42% of incoming queries consist of a single token, many of which are single-word channel names. Zero-length queries, comprised solely of punctuations, are mostly ASR errors. Some of the longer queries can be quite specific movie descriptions (e.g., "Go on the movie when the kids are on the bold and 22 of them got stranded on island.") or just an excited kid repeating the same query over and over (e.g., "the amazing world of gumball" repeated four times). I also recognize movie quotes and lyrics in the dataset, which tend to be longer in length.

Finally, the right panel in Figure 7.1 shows the number of queries in a "voice session", which I define as a sequence of consecutive queries with a maximum gap less than 45 seconds between queries. More than 77% of the sessions contain only a single query. However, a considerable number of very long sessions exist, sometimes up to a hundred or more queries. Some of these tend to be exploratory, where the viewer uses voice to navigate the catalog around a central theme: exploring the cast of a movie or a series of similarly-themed movies are two such examples. Others are more mechanical—for example, there are viewers who "zap" through channels by uttering channel names or numbers one by one (e.g., "channel 22", "channel 20", "CNN", etc.). There are also cases where the viewer appears to be having fun with the remote by saying random things.

7.2.2 Intent Classification

In this section, I introduce a taxonomy for viewer intents to categorize different types of queries, originally developed by the Comcast voice remote team. Note that my analysis is based on the output of the production system that was deployed at a particular point in time. The system was based on a combination of hand-crafted rules and machine-learned models to detect viewer intents, over a taxonomy that has organically evolved over time. I would characterize the accuracy as “reasonable”, but certainly not perfect. Although system output error is a confound, I do not believe errors substantively alter my findings.

The distribution of intents in the dataset is shown in Figure 7.2. Not surprisingly, queries to entertainment systems revolve around a desire to watch something. At a high level, we break this intent down into whether the viewer is looking for a specific program (View) or not (Browse). In the query logs, the View intent comprises approximately 66% of all queries, and can be further broken down into the following three categories:

- View Channel (29.7%): the viewer wishes to watch a specific channel such as HBO or ESPN. These voice queries obviate the need for the viewer to remember specific channel numbers.
- View Program (27.1%): the viewer wishes to watch a specific program by name. This could be a series (e.g., “Game of Thrones”), a specific movie (“Back to the Future”), a comedy act, etc.
- View Event (8.8%): the viewer wishes to watch the broadcast of an event such as the Super Bowl or the Oscars. These events are almost always manually curated.

The Browse intent, where viewers do not have a specific program in mind, represents 6.5% of queries. Examples are “show me free kids movies” or “HD movies

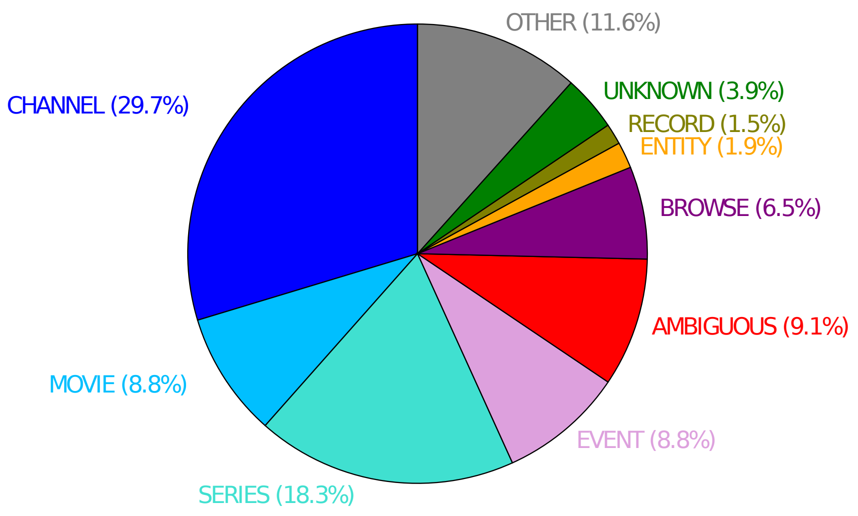


Figure 7.2: Intent distribution from the query logs.

with Julia Roberts". In these cases, the viewer has some idea of the desired program but is expecting suggestions from the system. Any query that involves filtering the program catalog is identified with this intent.

Beyond VIEW and BROWSE, the taxonomy includes three other less frequent categories:

- ENTITY (1.9%): the viewer wishes to examine a particular entity profile (e.g., of an actor such as Tom Hanks). This profile includes the actor's picture, bio, filmography, etc.

- RECORD (1.5%): the viewer is accessing DVR functions.

- OTHER (11.6%): there is a long tail of infrequent intents (a few dozen) that we lump together. These include everything from toggling closed captioning, accessing the home security system, debugging wifi connections, and engaging external apps.

Finally, there are two categories that are specifically artifacts of the production system:

- AMBIGUOUS (9.1%): the system identified two or more possible intents and prompts the viewer with a "did you mean..." dialog.
- UNKNOWN (3.9%): the system was not able to identify an intent, either due to algorithmic limitations or genuine cases in which no clear intent was expressed.

The VIEW intent is analogous to known-item retrieval in the document retrieval context and captures what we have previously called navigational voice queries in Chapter 6.

7.2.3 Query Tagging

In Chapter 6, query understanding is formulated as multi-way classification over a set of programs. Although queries with the VIEW intent dominate the dataset, there are at least two reasons why such an approach falls short: First, for intents other than VIEW, program prediction obviously makes no sense. Second, even for VIEW intents, a classification-based formulation has difficulty handling tail programs. There are typically tens of thousands of programs accessible to viewers at any time, especially including on-demand titles. For programs that are not frequently watched, there is insufficient training data; for example, the previous HRNN model handles less than a thousand programs. It would be desirable to give viewers voice access to the entire catalog.

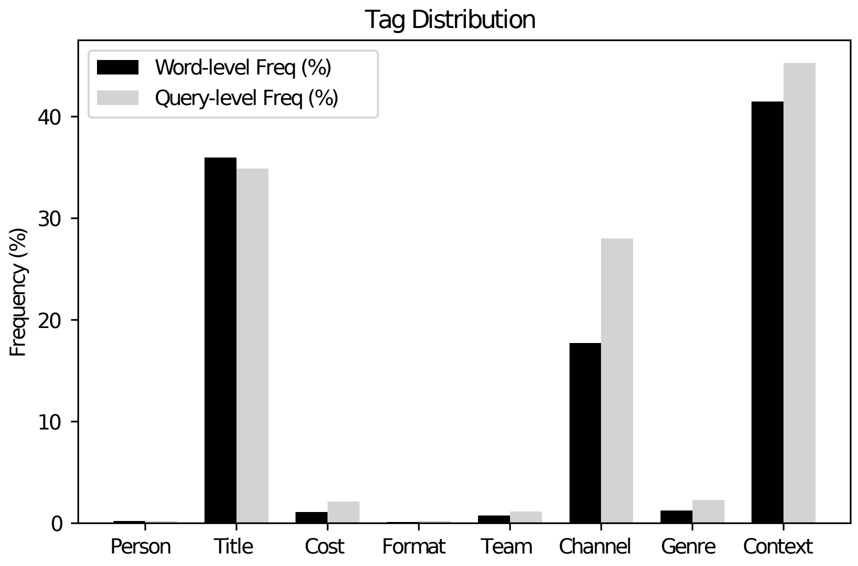


Figure 7.3: Distribution of query tags.

To address these issues, I employ query tagging, which works in conjunction with intent classification to provide a fine-grained analysis of viewers' queries. Here, the problem is formulated as a sequence labeling task, with the following tag set:

- PERSON: a person named entity.
- TITLE: the title of a program.
- TEAM: a sports team or sports-related term (e.g., "NFL").
- COST: terms related to cost (e.g., "free").
- FORMAT: terms related to format (e.g., "HD", "4K").
- ASSET: e.g., "movie", "series", "music video", etc.
- GENRE: e.g., "drama", "action", "comedy", etc.
- CONTEXT: a catch-all for all other terms.

For example, from the query "Watch Tom Hanks movies in HD", we extract a sequence of tags: Context Person Person Asset Context Format. Similar to intent detection, the current system takes advantage of handcrafted patterns as well as machine-learned models to parse the query into the logical form, e.g., (Person="Tom Hanks" $\wedge$ Asset="Movie" $\wedge$ Format="HD"). This is then used to filter the program catalog to provide a list of suggestions.

In Figure 7.3, the solid dark bars show the distribution of tags over all tokens observed in the dataset, whereas the lighter gray bars show the percentage of queries in which each tag exists. Based on this, we see that about 58% of tokens are part of either a named entity or modifier (not Context). Only 29% of queries are entirely made up of context tokens (i.e., no entities or modifiers were extracted). In this entity-heavy dataset, title and channel mentions alone constitute over half of all tokens. Even though some of the tag types occur less frequently than others (e.g., only 1% of tokens are tagged as Genre), high accuracy for all tags are necessary to produce a good user experience. For example, genre-based movie browsing requires reliably identifying Genre tags.

Intent classification, program prediction, and query tagging work together in a complementary fashion. They can be combined to resolve ambiguity and reinforce confidence on clear intents, or can be used as increasingly-broad backoff mechanisms to cope with queries that have vague intents. For example, if we identify a clear viewing intent and also a specific program, there is a high degree of confidence that the joint prediction is correct. On the other hand, if the system identifies a Browse intent, the various modifiers from tagging (e.g., Format, Cost, etc.) play an important role to understand a viewer's query. In this example, intent prediction and query tagging need to work together to generate an appropriate response.

7.2.4 Beyond Navigational Queries

Finally, I present a preliminary linguistic analysis of the query logs to provide a glimpse into the diversity of viewer queries posed to entertainment systems. In order to score queries based on some "naturalness" measure, I trained a language model using the Hansard parliament speech corpus (0.76M sentences) and the IMDB movie review dataset (1.22M sentences). As a filtering step, I removed all queries that matched a title in the catalog exactly as well as any query with five tokens or less. This yielded a set of 2.9M queries (1.1M unique), which were then scored by the language model and sorted by the LM score plus the log of the frequency of occurrence. The result is a ranked list of frequently-occurring "natural" utterances directed at the voice remote.

Analyzing the results, we observe a wide range of intents. In fact, the percentage of UNKNOWN queries is 50% higher in this subset of the logs, pointing to an increased level of complexity. The percentage of BROWSE queries is also much higher (15% vs. 6%), which affirms the need for a tagging-based approach (as presented in Chapter 7.2.3) to properly understand complex queries.

Queries ranked highly in the "naturalness" measure ranged from movie quotes and music lyrics (e.g., "All I want to say is that they don't really care about us.") to very specific requests (e.g., "Return to the movie that I did not finish last night.").

On the other hand, there were also open-ended questions (e.g., “Do you have a movie about the Vietnam War?”) as well as factual questions (e.g., “Who is being nominated for best picture in the Academy Awards?”).

To gain a little more insight into the syntactic structure of the queries, I ran a dependency parser [145] on all 1.1M unique queries in this subset. The most common root word was “show” with part-of-speech verb (`show/VB`), comprising 12% of all queries. In fact, root words of verb forms (`VB`, `VBP`, `VBZ`, etc.) comprised half of all queries. The remaining queries had a root with the part-of-speech noun (40%), adjective (2%), preposition (1%), and determiner/pronoun (negligible). The most frequently observed noun root was `movies/NNS`; for adjective and prepositions, `free/JJ` and `on/IN` topped the list, respectively.

7.3 Multi-Task Learning Model

Inspired by the previous log analysis, in this section, I present a multi-task learning architecture with detailed explanations about model design for program prediction, intent classification, and query tagging. I first define the problem in a probabilistic manner, then present the neural models specifically-designed for each of the three tasks.

7.3.1 Problem Formulation

Given a voice query session $[q_1, \ldots, q_n]$, the task is to predict three types of information: 1) a tag sequence associated with each query, 2) an intent type of

Symbol	**Description**
s_i	the i-th session in the whole dataset D
q_{i_t}	the t-th query in the i-th session
a_{i_t}	the intent type (i.e., *channel*) of query q_{i_t}
τ_{i_t}	the tag sequence (i.e., *context* -> *entity_name*) of query q_{i_t}
p_i	the program label of the i-th session
A_i	a list of intent types $(a_{i_1}, ..., a_{i_n})$ for queries in session s_i
$\mathcal{T}_i$	a list of tag sequences $(\tau_{i_1}, ..., \tau_{i_n})$ for queries in session s_i
v_{i_t}	embedding vector for query q_{i_t}
c_{i_t}	contextual vector for the first t-th query $[q_{i_1}, ..., q_{i_t}]$ in s_i

Table 7.1: Notation Table.

each query and 3) the program p that the user intends to watch. I perform the three tasks in parallel on each successive new voice query q_t $(t \in [1, n])$, exploiting all previous queries in the session $[q_1, \ldots, q_{t-1}]$ as context. For example, in a three-query session $s_i = [q_{i_1}, q_{i_2}, q_{i_3}]$, there will be three sets of predictions: first with input as $[q_{i_1}]$, second with $[q_{i_1}, q_{i_2}]$, and third with $[q_{i_1}, q_{i_2}, q_{i_3}]$. At each time step t, the three prediction tasks (program prediction, intent classification and tagging) will be performed simultaneously. The input to the system is a sequence of voice queries (i.e., a voice session), which are text strings transcribed from a third-party "black box" ASR system.

In general, we aim to learn a mapping function Θ from a query sequence to a set of predictions, including the intended program, the intent type and the tag sequence of each query. I model this mapping through a probabilistic framework:

$$\text{Data: } D = \{(s_i, p_i, A_i, \mathcal{T}_i) \mid s_i = [q_{i_1}, ..., q_{i_n}],\ p_i \in \Phi,$$
$$A_i = [a_{i_1}, ..., a_{i_n}], \mathcal{T}_i = [\tau_{i_1}, ..., \tau_{i_n}]\}_1^{|D|}$$

$$\begin{aligned} \text{Model: } \hat{\theta} &= \arg\max_{\theta} \prod_{i=1}^{|D|} \prod_{t=1}^{n} P(p_i, A_i, \mathcal{T}_i | q_{i_1}, ..., q_{i_t}; \theta) \\ &= \arg\max_{\theta} \prod_{i=1}^{|D|} \prod_{t=1}^{n} P(p_i, a_{i_1}, ..., a_{i_t}, \tau_{i_1}, ..., \tau_{i_t} | q_{i_1}, ..., q_{i_t}; \theta) \end{aligned} \tag{7.1}$$

where D denotes a set of labeled sessions (s_i denotes the i-th session with n queries), p_i is the intended program for session i, Φ is the global set of programs, A_i represents a list of intent types for the i-th session where a_{i_t} is the intent type (a *scalar*) of the t-th query in the i-th session, $\mathcal{T}_i$ is a list of tag sequences with each τ_{i_t} denotes the tag *sequence* of the t-th query in the i-th session, and θ is the set of parameters in the mapping function Θ. All the symbols used in this chapter are also presented in Table 7.1 for quick reference. Overall, the goal is to maximize the product of prediction probabilities for all queries in the dataset D (Equation (7.1)).

As mentioned above, the program prediction, intent classification and tagging tasks can share information to reinforce the learning of better discriminative features. I model such interactions in a multi-task learning framework where 1) the three tasks share some underlying layers and have a task-specific component to enable information sharing and task-specific optimization, and 2) the objective loss functions of the three tasks are weighted and summed together during optimization. To this end, I decompose the prediction tasks in Equation (7.1) into learning three components: a query embedding component $\mathbb{F}(x; \theta_{\mathbb{F}})$, a contextual component $\mathbb{G}(x; \theta_{\mathbb{G}})$, and a family of task-specific components $\mathbb{H}(x; \theta_{\mathbb{H}})$. The query embedding component $\mathbb{F}(\cdot)$ takes the text of a query as input and produces a semantic embedding representation of the query. The contextual component $\mathbb{G}(\cdot)$ considers representations of all the preceding queries as context and maps them to a high-dimensional embedding vector to capture both semantic and contextual features. The query embedding and contextual components are shared across tasks. Finally, the task-specific components $\mathbb{H}(\cdot)$ perform separate task-specific predictions based on the learned contextual vector. I adopt the following decomposition:

$$\begin{aligned} P(p_i, a_{i_t}, \tau_{i_t} | q_{i_1}, ..., q_{i_t}) \sim P(p_i, a_{i_t}, \tau_{i_t} | c_{i_t}) \cdot P(c_{i_t} | v_{i_1}, ..., v_{i_t}) \\ \cdot P(v_{i_1}, ..., v_{i_t} | q_{i_1}, ..., q_{i_t}) \end{aligned} \tag{7.2}$$

where c_{i_t} denotes the *contextual* embedding of the first t queries in the i-th session and v_{i_t} denotes the embedding of the t-th query of the i-th session. The relationship between these embeddings can be formulated using the three component mappings above: $\mathbb{F}$ maps the query q_{i_t} to its embedding v_{i_t} in vector space; $\mathbb{G}$ maps the sequence of query embeddings $[v_{i_1}, ..., v_{i_t}]$ to a contextual embedding c_{i_t}; and $\mathbb{H}$ maps the contextual embedding to task-specific predictions:

$$\begin{aligned} v_{i_t} \sim \mathbb{F}(q_{i_t}; \theta_{\mathbb{F}}), \quad c_{i_t} \sim \mathbb{G}(v_{i_1}, ..., v_{i_t}; \theta_{\mathbb{G}}), \\ (p_i, a_{i_t}, \tau_{i_t}) \sim \mathbb{H}(c_{i_t}; \theta_{\mathbb{H}}) \\ 1 \leq t \leq n \end{aligned}$$

By assuming that each query is embedded independently, we can reduce Equation (7.2) as follows:

$$P(p_i, a_{i_t}, \tau_{i_t} | q_{i_1}, ..., q_{i_t}) = P(p_i|c_{i_t}) \cdot P(a_{i_t}|c_{i_t}) \cdot P(\tau_{i_t}|c_{i_t}) \cdot P(c_{i_t}|v_{i_1}, ..., v_{i_t}) \cdot \prod_{j=1}^{t} P(v_{i_j}|q_{i_j})$$

Note that for a tagging task, we only need the query itself to generate tags for each word, while the contextual information can be useless. Therefore I replace the term $P(\tau_{i_t}|c_{i_t})$ with $P(\tau_{i_t}|q_{i_t})$, through which we can reformulate the model framework in Equation (7.1) as follows:

$$\begin{aligned}\text{Model: } \hat{\theta} &= \arg\max_{\theta} \prod_{i=1}^{|D|}\prod_{t=1}^{n} P(p_i, A_i, \mathcal{T}_i | q_{i_1}, ..., q_{i_t}; \theta) \\ &= \arg\max_{\theta} \prod_{i=1}^{|D|}\prod_{t=1}^{n} P(p_i|c_{i_t}) \cdot P(a_{i_t}|c_{i_t}) \cdot P(\tau_{i_t}|q_{i_t}) \\ &\cdot P(c_{i_t}|v_{i_1}, ..., v_{i_t}) \cdot \prod_{j=1}^{t} P(v_{i_j}|q_{i_j}) \end{aligned} \quad (7.3)$$

After decomposing the probabilistic framework into distinct components, I propose a neural network based approach to model each component and adopt multi-task learning to optimize the model simultaneously.

7.3.2 Model Architecture

The overall model architecture is shown in Figure 7.4 and consists of three distinct components:

(1) Query embedding component $\mathbb{F}(\cdot)$, shown in the bottom lookup layer and the blue BiLSTM: the lookup layer converts a raw query string into a sequence

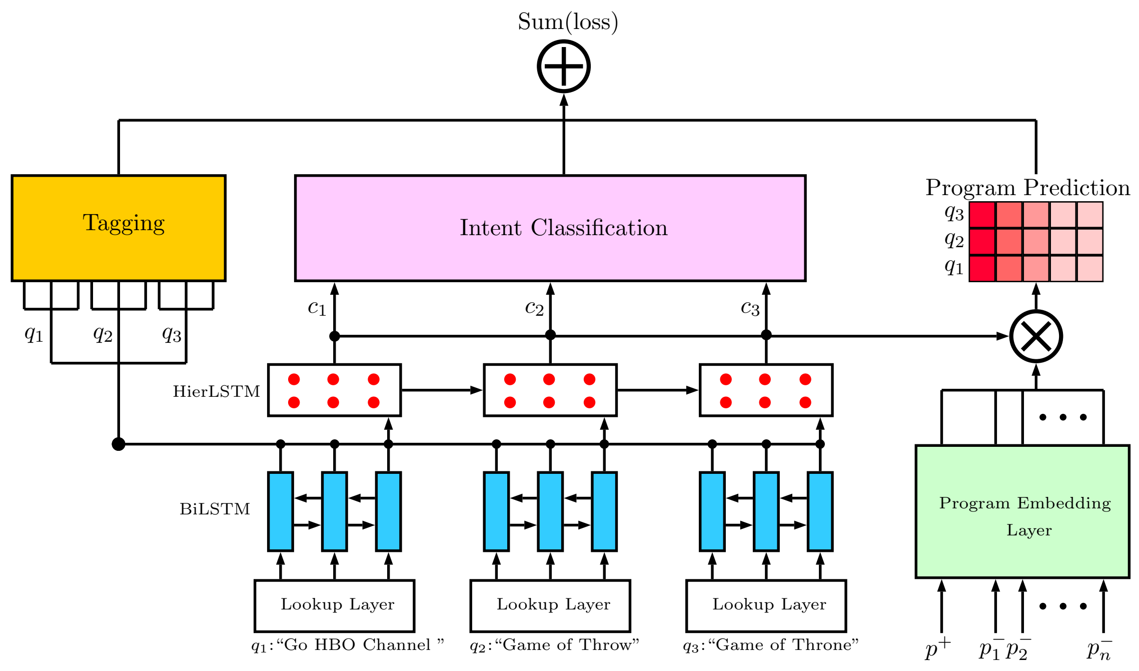


Figure 7.4: The proposed model architecture, which contains three distinct components: (1) a query embedding component at the bottom (blue rectangles) that converts a query string into a learned representation, (2) a contextual component (red dotted rectangles) that models the context between queries in a session, and (3) task-specific components designed for three tasks: program prediction (in red), intent classification (in pink), and query tagging (in yellow).

of vectors (through word2vec [12]) and the BiLSTM learns a semantic embedding for the query. More formally, given a query q_{i_t} represented as a sequence of words $\{\mathbf{w}_{i_t}\}$, the output is a sequence of hidden vectors $\{\mathbf{h}_{i_t}^{bi}\}$ learned from the BiLSTM. The last hidden vector is used as the query embedding v_{i_t}, which is passed to the contextual component. The sequence of hidden vectors $\{\mathbf{h}_{i_t}^{bi}\}$ serves as input to the tagging model to generate a word-level tag sequence.

(2) Contextual component $\mathbb{G}(\cdot)$, shown as the red-dotted rectangles, is a LSTM model that takes all the preceding query embeddings $\{v_{i_1}, ..., v_{i_t}\}$ as context to produce a contextual vector c_{i_t} that captures both semantic and contextual features. The contextual vectors $\{c_{i_t}\}$ are then fed into the intent classification and program prediction components.

(3) Task-specific components $\mathbb{H}(\cdot)$ for query tagging (the yellow rectangle), intent classification (the pink rectangle), and program prediction (the red rectangle). At the top, I weight the losses from the three tasks and sum them together for unified multi-task learning. Details of the three task components are provided in Chapters 7.3.2.1–7.3.2.3, and multi-task optimization in Chapter 7.3.3.

7.3.2.1 Program Prediction

The first task-specific component is responsible for program prediction, which models the probability $P(p|c_{i_t})$ of generating program p given the contextual vector c_t for the i-th ongoing session $\{q_{i_1}, ..., q_{i_t}\}$. Unlike the HRNN model that treats program prediction as multi-way classification (and thus is unable to handle rarely-watched tail programs and newly-added programs), I model the task as a ranking problem (i.e., modeling the relevance between a query and a program). My model introduces a novel triplet ranking approach that can directly exploit interactions between training (query, positive program, negative program) triples and can take advantage of program embeddings to integrate different sources of evidence (e.g., program title, viewers' search and viewing histories) in a flexible manner. The learned program embeddings are paired with the query embeddings in a triplet loss to identify the most relevant program in a contrastive manner. In my model, a channel (e.g., "HBO") is treated exactly like a program.

Let Φ denote the set of all programs in the training dataset. I reformulate maximizing the classification probability $P(p|c_{i_t})$ as a ranking objective:

$$P(\text{rel} \mid c_{i_t}, p^+) > P(\text{rel} \mid c_{i_t}, p^-), \quad \forall p^+, p^- \in \Phi$$

That is, we wish to assign a higher relevance score to positive programs from the training data than any negative program. I propose three ways to learn a program embedding:

(1) Search-based program representation. We can define programs purely based on how viewers search for them. In this representation, the program embedding layer is a simple lookup layer that maps a program id (scalar) to a learned embedding vector, which is randomly initialized and updated during training. My assumption is that given enough data, the embeddings will converge to a meaningful representation, reflecting how viewers search for programs. This is similar to how word2vec [12] is trained from neighboring word associations; in this case, I use search-based associations instead. Like word2vec, such representations can overcome lexical mismatches. For example, viewers may search for a channel by its number (i.e., "210" $\rightarrow$ "HBO"), where there is no lexical overlap; this representation can learn such correspondences from log data. However, the drawback of a search-based representation is its dependence on observations; it is unable to handle cases where the viewer refers to an unseen program.

From this representation, the relevance score between an ongoing session $\{q_{i_1}, ..., q_{i_t}\}$ and a program can be measured by the cosine similarity between the program embedding p and the contextual vector c_{i_t}:

$$P(\text{rel} \mid p, q_{i_1}, ..., q_{i_t}) = P(\text{rel} \mid p, c_{i_t}) = \text{cosine}(p, c_{i_t})$$

(2) Title-based program representation. Alternatively, we can model query–program similarity lexically. To accomplish this, I copy the query embedding component as the program embedding layer and apply it to the program title, from which we can obtain another sequence of BiLSTM vectors representing the program. Let the query representation be $\{\mathbf{h}_q^{bi}\}$ and program representation be $\{\mathbf{h}_p^{bi}\}$. I adopt an interaction-based method (similar to [9]) to model the semantic similarity between each word pair in (query, program):

$$\text{sim}(j, k) = \text{cosine}(\mathbf{h}_q^{bi}[j], \mathbf{h}_p^{bi}[k])$$

From this we obtain a similarity matrix where each entry (j, k) denotes the cosine similarity between query word j and program title word k. I apply max pooling along the rows, which returns a query-sized feature vector. Each feature j denotes the highest similarity between any program title word and query word j. To capture the relative importance of different query words, I weight this feature vector (element-wise) by inverse document frequency (IDF). Similarly, I repeat the same operation along the columns to obtain a program-sized feature vector, where each element k denotes the highest similarity between any query word and program title word k (also IDF-weighted). These two feature vectors are concatenated and passed to a linear layer, which computes a relevance score for the query–program pair.

(3) Combination-based program representation. To capture the best of both representations, I stack another linear layer on top to combine the search-based and title-based relevance scores. This can be considered a linear learning-to-rank approach with only two feature inputs.

At this point, I have introduced three approaches for computing the relevance of a program with respect to the current query in the session. Given such a relevance scoring function $P(\text{rel})$, I explore the interactions between positive and negative programs through a softmax function:

$$o[p^+] = P(p^+|q_{i_1}, ..., q_{i_t}) = \frac{\exp(P(\text{rel} \mid p^+, q_{i_1}, ..., q_{i_t}))}{\sum_{p' \in C} \exp(P(\text{rel} \mid p', q_{i_1}, ..., q_{i_t}))}$$

where p^+ is the positively labeled program for session i, $o[p^+]$ denotes the probability of predicting p^+, and C denotes the set of candidate programs to be ranked. Ideally, C should be equal to the program set Φ; in practice, I approximate this through negative sampling, by selecting k (e.g., $k = 10$) programs from the top ranked results of query q_{i_t} using a standard retrieval algorithm (i.e., BM25). The goal is to maximize the likelihood of generating positive programs given queries across the dataset, which can be equivalently formulated as minimizing the program loss function:

$$L_p = -\sum_{i=1}^{|D|} \sum_{t=1}^{n} \log o_i[p^+] = -\sum_{i=1}^{|D|} \sum_{t=1}^{n} \log P(p^+|q_{i_1}, ..., q_{i_t})$$

where the outer sum iterates over all sessions in the dataset D, and the inner sum iterates over all queries in session i.

7.3.2.2 Intent Classification

Similar to program prediction, I also aim to predict the intent type a_{i_t} for query q_{i_t} given its contextual vector c_{i_t}. I model this task as a classification problem since the vocabulary of intent types is relatively small and stable. On the top section of Figure 7.4 (the pink rectangle labeled "Intent Classification"), I build a fully-connected layer followed by a softmax layer for learning the classification function. The fully-connected layer consists of two linear layers with a ReLU element-wise activation layer in between. The contextual vector c_{i_t} is fed into the fully-connected layer first, followed by $L1$ normalization via the softmax function. In the normalized vector o, each output score $o[a_j]$ denotes the probability of producing intent type a_j as output. The intent classification loss is summed over all queries in the dataset, as follows:

$$L_i = -\sum_{i=1}^{|D|}\sum_{t=1}^{n} \log P(a_{i_t}|c_{i_t}) = -\sum_{i=1}^{|D|}\sum_{t=1}^{n} \log o_{i_t}[a_{i_t}]$$

7.3.2.3 Query Tagging

Unlike program prediction and intent classification, which are query-level predictions, query tagging is a sequence labeling task. More formally, the query tagging component takes the output vector sequence $\{\mathbf{h}_{i_t}^{bi}\}$ from the bottom BiLSTM as input and generates a tag sequence $\{\tau_{i_t}\}$, where each tag label corresponds to a word in the query q_{i_t}.

I use a conditional random field (CRF) [146] as my tagging model (the yellow rectangle labeled "Tagging" in Figure 7.4) on top of the BiLSTM. In addition to capturing the neighboring word context through the BiLSTM, a CRF can exploit correlations between tag labels in neighborhoods to jointly decode the best label sequence for a given query. I use standard maximum likelihood estimation for training the CRF and the Viterbi algorithm for decoding. I omit the technical description of CRFs for space reasons, but refer readers to Lafferty et al. [146] for details.

7.3.3 Multi-Task Learning

Since the three tasks have their own optimization objectives, I adopted a multi-task learning strategy to train the entire model end to end, jointly optimizing all three tasks. This is accomplished in two stages: Following a commonly adopted strategy [147, 148], in the first stage, all tasks are jointly trained by summing up their losses based on a mixing ratio. Let w_p, w_i, and w_t be the contribution weight to the combined loss from each task (program prediction, intent classification, and query tagging, respectively). The overall loss is computed as follows:

$$L = w_p \cdot L_p + w_i \cdot L_i + w_t \cdot L_t$$

where w_p, w_i, and w_t sum to 1.0. In the second stage, I fix the underlying shared layers and fine-tune the top task-specific layers. In the model, the weights of the bottom BiLSTM and hierarchical LSTM layer are held constant, and the top layers are trained separately with task-specific losses.

7.4 Evaluation

7.4.1 Data Collection

To build a dataset for supervised learning, we need the following for each session: the program title (one per session), the intent type (one per query), and query tags (one for each query word). I use a combination of logs and human annotations to obtain such data. The program title is extracted entirely from logs, in a manner analogous to harvesting clickthroughs in the web domain: If the viewer began watching program p after the final query in a session and continued watching it for at least 150 seconds, I label the session with p (this duration parameter was explored in Chapter 6.4.1). For the intent type and the query tags, I used a semi-supervised approach to collect ground truth data. Initially, hand-crafted patterns were manually designed by the annotation team, which were then applied to parse queries into a logical form, from which I extracted the intent type and query tags. New patterns were gradually added over time to increase coverage. This bootstrapping process can be thought of as a simple yet practical human annotation strategy when exhaustive hand-labeling is infeasible in a large-scale setting.

Using this process, I extracted a total of 8.8M training instances (labeled sessions) from 81.4M queries received in the week of February 22 to 28, 2017, which I then randomly sampled and split into training, validation, and test sets. The intent and query tags represent the output of a particular set of hand-crafted patterns at a particular point in time. Basic statistics are summarized in Table 7.2. All three

Dataset	Sessions	Queries	Avg. Session Len	Avg. Query Len
Train	870941	1186937	1.36	2.24
Valid	623142	823565	1.32	2.23
Test	622959	825639	1.33	2.23

Table 7.2: Dataset statistics.

sets are sufficiently large to realistically capture the diversity of viewer queries. The percentage of single-query vs. multi-query sessions is about 80:20 for all three sets. The program set contains 26247 distinct programs and 244 channels. About 10% of the queries in the validation and test sets have program labels that are not seen in the training set. The total number of intent and tag types are 109 and 11, respectively.

7.4.2 Model Training

I used 300-dimensional word2vec [12] embeddings to encode each word, which is trained on the Google News dataset and freely available. The word vocabulary of the training set is 29.3K and 4282 lack word2vec vectors. These words were randomly initialized with values uniformly sampled from $[-0.05, 0.05]$. Words unseen in the validation and test sets were treated as out of vocabulary.

During training, I used stochastic gradient descent together with the Adam optimizer to iteratively update model parameters. The learning rate was initially set to 10^{-3} and then decreased by a factor of three when the validation set loss stopped decreasing for three epochs. The LSTM output size and the size of the linear layer was set to 150. The batch size was set to 256.

At test time, I selected for evaluation the model that obtained the lowest task-specific loss on the validation set. I used the top 20 programs retrieved by BM25 from the program vocabulary as the input candidates to my models for reranking. I also add all channels into the candidates pool when the detected viewer intent is to view a channel. This helps cases when the query and program share no lexical overlap (i.e., the query is a channel number and the "program" is the channel name). My models were implemented using Keras, running on a server with 8 GPUs (GeForce GTX TITAN X) and 256GB RAM.

In order to demonstrate the effectiveness of multi-task learning, I compared two different approaches to training my models:

Single-Task Learning (STL). Although the architecture is designed for multi-task learning, it can still be trained for a single task. In this mode, the training process only optimizes the intended task loss (e.g., intent classification), while ignoring losses from the other two tasks (by assigning zero to their mixing weights). Typically, the training process converges in five epochs and each epoch takes about 1.5 hours.

Multi-Task Learning (MTL). For multi-task learning, I used the two stage approach described in Chapter 7.3.3. For tuning the weights of the individual task-specific loss, I performed cross-validation on the validation set to select the best mixing ratio that minimizes the weighted sum of the three task-specific losses. In practice, I found a mixing ratio of (0.55, 0.05, 0.4) for program prediction, intent classification, and query tagging, respectively, works well for the search-based program representation, and (0.1, 0.2, 0.7) worked well for the title-based and combination-based program representations. Compared to STL, the MTL training process takes much longer, typically 15 epochs per stage.

7.4.3 Metrics and Baselines

Intent classification and query tagging tasks were evaluated based on accuracy. For program prediction, I used three metrics (averaged over all queries): precision at one (P@1), precision at five (P@5), and Mean Reciprocal Rank (MRR). Use of these metrics was motivated by the precision-oriented nature of television navigation, as limited input options require our system to satisfy the viewer's query as quickly as possible. A number of baselines are described below; note that some baselines are designed for a particular task, while others can be extended to all three tasks.

BM25: I built a 3-gram (character-level) inverted index of the program set Φ. During retrieval, the matching score is computed on 3-gram overlaps between query and candidate programs using Okapi BM25 weighting ($k_1 = 1.2$, $b = 0.75$).

SVM$^{\textbf{rank}}$ [142]: I reused the learning-to-rank baseline introduced in the previous chapter 6.4, which includes exact and soft-match features (BM25 and embedding-based) as well as popularity priors.

DSSM [11]: This neural ranking model for web search uses word hashing to model interactions between queries and programs at the level of 3-grams. This method is an appropriate baseline for my problem since it can handle noisy ASR output, unlike neural ranking models based primarily on word matching [10, 46].

DSSM+S: Using the same model as above, I concatenate queries in one session with a special boundary token between neighboring queries. The goal here is to examine whether simple query concatenation is sufficient to capture context signals in a session.

Stanford CRF Tagger[1] [149]: A standard baseline for token-level tagging problems, I trained a linear CRF that combines the most popular local and global features, including features based on n-grams, context windows, etc.

HRNN w/ LSTM/BiLSTM [42]: The original model, as described in Chapter 6.3, can be extended to intent classification and query tagging task by adding separate fully-connected layers for each task. I also tried replacing the bottom LSTM layer with a BiLSTM to examine the effects of bidirectional query modeling.

7.4.4 Experimental Results

Results for all three tasks are shown in Table 7.3. Each row represents an experimental setting (numbered for convenience). The second column specifies the model, and remaining columns indicate the results for program prediction, intent classification, and query tagging, respectively. MTLA refers to my multi-task learning architecture described in Chapter 7.3, trained either using the single-task learning or multi-task learning conditions described in Chapter 7.4.2. Results are shown in the two subtables titled "Single-Task Learning" and "Multi-Task Learning", respectively.

[1]`https://nlp.stanford.edu/software/CRF-NER.shtml`

ID	Model	Program			Intent	Tagging
		P@1	**P@5**	**MRR**	**Accuracy**	
1	BM25	0.674	0.750	0.711	-	-
2	SVM$^{\text{rank}}$	0.682	0.758	0.718	-	-
3	DSSM	0.703	0.765	0.732	-	-
4	DSSM+S	0.699	0.758	0.728	-	-
5	Stanford CRF Tagger	-	-	-	-	0.821
6	HRNN LSTM	0.724	0.783	0.755	0.915	0.884
7	HRNN BiLSTM	0.725	0.786	0.753	0.916	0.939
	Single-Task Learning					
8	MTLA (search-based)	0.715	0.770	0.744		
9	MTLA (title-based)	0.720	0.796	0.754	0.917	0.944
10	MTLA (comb-based)	**0.738**	**0.802**	**0.768**		
	Multi-Task Learning					
11	MTLA (search-based)	0.721	0.780	0.758	0.923	**0.946**
12	MTLA (title-based)	0.728	0.803	0.762	0.924	0.945
13	MTLA (comb-based)	**0.757**	**0.812**	**0.792**	**0.925**	0.945

Table 7.3: Model effectiveness, where each row represents an experimental setting. MTLA refers to multi-task learning architecture described in Chapter 7.3, trained either using the single-task learning or multi-task learning conditions. Columns show results for program prediction, intent classification, and query tagging.

7.4.4.1 Program Prediction

First, we can see that BM25 achieves reasonably-high accuracies (P@1 of 0.674) on the test set. The SVM$^{\text{rank}}$ predictor achieves slightly better accuracy than BM25 by taking advantage of multiple hand-crafted features in a supervised setting. Taking a closer look at the learned model weights, we find that these additional features are largely dominated by the BM25 feature and provide only modest benefit to the overall model. DSSM significantly outperforms SVM$^{\text{rank}}$ as well as BM25, whereas DSSM+S performs slightly worse than DSSM, suggesting that simple query concatenation is not able to capture context signals in a session.

As expected, the HRNN is able to outperform the other baselines by quite a bit. I identified two main reasons: (1) There are about 3% of queries (about 10% of all channel-intent queries) in which the viewer searched for a particular channel by memorizing its channel number; (2) Major ASR errors sometimes results in very little lexical overlap between the query and the intended program title (e.g., "Dr. Seuss's The Lorax" is transcribed as "The Laura"). In both cases, the query and the program lack common 3-grams, so approaches based on text matching like DSSM cannot predict the correct program. I further examine the session context by decomposing the dataset into single-query and multi-query sessions. Comparing DSSM and HRNN, the accuracy gap is much larger on multi-query sessions (P@1 of 0.546 vs. 0.483) than single-query sessions (P@1 of 0.770 vs. 0.757), thus affirming the value of modeling session context. Single-query sessions obtain much higher accuracies since they are by construction "easy queries" in which viewers reach their intended program in a single query. The above findings are consistent with Chapter 6.5. Finally, we see that bidirectional modeling (BiLSTM) provides little benefit for program prediction.

Turning our attention to the "Single-Task Learning" subtable (rows 8–10), we observe that the search-based program representation performs worse than the HRNN model. Considering that these two models share the same underlying query embedding and contextual component, the difference comes from the problem formulation (classification vs. ranking) and how we train the model. In the search-based model, I selected k negative programs for each query-program sample during training. This can be less effective than a classification formulation since the classification loss enforces the HRNN model to select the positive program against *all* negative programs, thus giving it stronger discriminative power. However, this does not mean that classification is better in this problem setting, as its drawbacks are pointed out in Chapter 7.1. This is further verified by the results of the combination-based ranking approach (row 10), which significantly outperforms HRNN at $p < 0.05$ using Fisher's two-sided, paired randomization test [1]. Incorporating signals based on program titles helps the combination-based approach answer queries that are intended for programs not observed during training, which cannot be answered using a classification approach.

Subtable "Multi-Task Learning" (rows 11–13) confirms the benefits of multi-task learning. Regardless of program representation, program prediction is consistently and significantly ($p < 0.05$) better than training in isolation. As expected, the improvements from multi-task learning come from partial task overlap with intent classification and query tagging. High confidence in a predicted intent is able to help the program prediction component discard outputs that conflict. For example, if the query "Disney channel shows" is predicted as having intent Browse, the program prediction should be *NA* (no answer) instead of a specific program.

7.4.4.2 Intent Classification

As shown in the "Intent" column in Table 7.3, the HRNN models are strong baselines with fairly high accuracies, suggesting that the intent classification problem is an easier task due to the limited size of intent set. Bidirectional modeling doesn't help improve the accuracy here. Since the approaches in subtable "Single-Task

Intent	Channel	Movie	Series	Event	Browse
Channel	97.9%	0.4%	0.5%	0.0%	0.2%
Movie	0.4%	89.7%	2.4%	0.1%	3.3%
Series	0.2%	0.8%	96.2%	0.0%	1.3%
Event	0.4%	3.7%	1.6%	87.3%	0.0%
Browse	0.1%	1.9%	1.8%	0.0%	94.1%

Table 7.4: Confusion matrix for the top five intent types, where the rows indicate the actual labels and the columns the predicted labels.

Learning" are essentially the same model as the HRNN BiLSTM (with respect to intent classification), the differences are negligible. However, by jointly learning all three tasks, we see consistent improvements. The best method (combination-based) achieves an accuracy of 0.925, which we consider quite impressive given the diversity of real voice queries.

For further insights, I show the confusion matrix of the best method (combination-based) for the most frequent five intent types in Table 7.4 (where the rows indicate the actual labels and the columns the predicted labels). The Channel intent has the highest accuracy, while the Event intent has the lowest accuracy. This matches our intuition that channel tuning is an easier task while the Event intent is harder to identify given its somewhat vague definition. We also see that the model is often confused between the remaining three intent types: Movie, Series, and Browse. Again, this is likely due to blurred lines between these intent types. For example, the query "life of pets" can be interpreted either as an intent to watch the movie with that title, the television series, or to Browse the catalog for documentaries about pets.

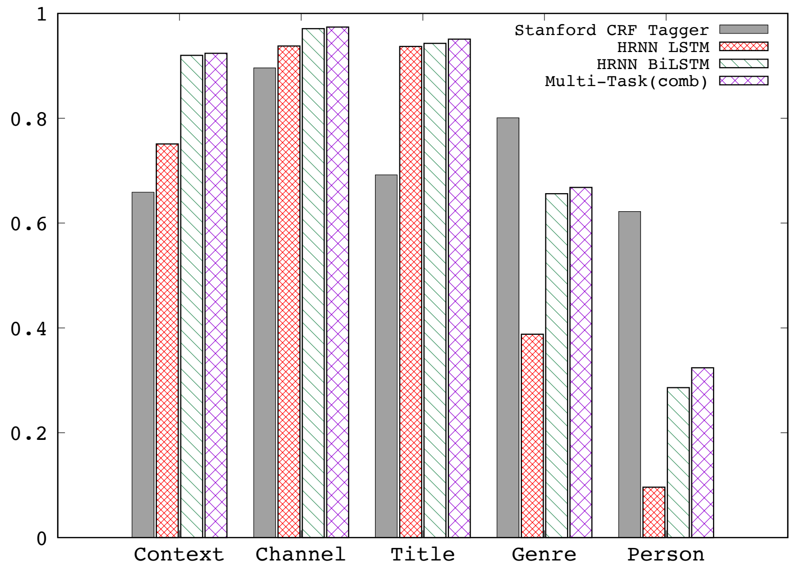


Figure 7.5: Tagging accuracy of all methods for five tags.

7.4.4.3 Tagging

In the final "Tagging" column in Table 7.3, we see that the Stanford CRF tagger achieves the lowest accuracy of 0.821 among all baselines—the HRNN LSTM is able to outperform the CRF-only approach by more than six absolute points. Unlike the other two tasks, bidirectional modeling is crucial for the tagging problem because the tag of a particular word is dependent on both the previous and next words. By introducing a CRF on top of the BiLSTM in my multi-task model ("Single-Task Learning"), my model is able to significantly beat the HRNN BiLSTM and CRF baselines ($p < 0.05$). Finally, multi-task learning provides an additional small boost to tagging accuracy.

Not all tags are created equal, which is why I explored the accuracies of various methods for five representative tags, shown in Figure 7.5. For the most common three tags (*context*, *channel*, and *title*) that make up 95% of tokens, my multi-task learning approach consistently performs the best while the Stanford CRF Tagger performs poorly. However, the Stanford CRF Tagger performs well on the least frequent tags (*genre* and *person*). This is likely due to insufficient training data for the *genre* and *person* tags (each tag appears in less than 1.5% of tokens). The LSTM-based approaches have a much larger parameter space, making them more data hungry. Overall, my best multi-task approach achieves a tagging accuracy of nearly 95% on average. Once again, we believe that these results are quite impressive given the diversity of real-world queries.

7.5 Conclusion

To tackle the limitations of my initial classification-based solution for voice query navigation, I designed a novel neural architecture to jointly accomplish three related tasks: program prediction, intent classification, and query tagging. This chapter articulates how the three tasks complement each other to understand a wide range of intents. The experiments demonstrate how joint learning improves the effectiveness of each task individually, yielding significant gains over strong baselines. More importantly, my multi-task framework provides an opportunity to build a complete end-to-end system for understanding voice queries. This new model is now being prepared for deployment and will soon be serving millions of Comcast customers, providing natural voice-based interactions for the entertainment domain.

Chapter 8: Conclusion and Future Work

8.1 Summary

This dissertation introduces families of techniques for modeling temporal information as contexts to assist applications with streaming inputs, such as tweet search and voice search. In tweet search, the temporal distribution of relevant documents can be a useful relevance signal to boost ranking effectiveness. I explore two directions, *pseudo trend* and *query trend*, using different sources of temporal signals to estimate the distribution of relevant documents. In addition, motivated by the ineffectiveness of existing neural ranking models for tweet search, I propose a multi-perspective lexical modeling approach with a customized architecture to incorporate multiple sources of lexical signals. In voice search, queries in the same session can help disambiguate the user's real intent. I propose successively richer neural network architectures to model the session contexts and cope with queries that express a multitude of intents.

Pseudo Trend Modeling for Tweet Search. In Chapter 3, I explore methods to estimate the distribution of relevant documents from timestamps of a list of initially retrieved documents. I first present an end-to-end neural framework to model pseudo trends as a sequence learning problem, where temporally-ordered documents can have impact on their neighbors. This is achieved by a lexical modeling component that first converts query-document pairs to vector representations denoting their similarities, then by a bi-directional RNN component that models the interactions of neighboring documents. Next, I propose a continuous HMM based approach to model pseudo trends, and utilize the estimated bursty HMM states for selecting more expressive terms for query expansion. Experimental results indicate: 1) coupled with the best lexical modeling component, the neural temporal framework obtains significant improvements over competitive temporal baselines, suggesting that neural network-based techniques are promising for temporal modeling; 2) the continuous HMM model selects better terms for query expansion and achieves further effectiveness gains.

Query Trend Modeling for Tweet Search. Beyond pseudo trends, I explore another source of temporal signal in Chapter 4 that can be captured from time-aware collection statistics of query terms, which is called a query trend. It enables us to recover the distribution of relevant documents directly from the term statistics stored offline without an initial retrieval, which can be substantially faster than pseudo trend-based methods. I first explore different compression methods to compress the sparse time-sliced term statistics into compact representations, then present two methods to model query trend signals: a linear feature-based ranking method and a non-linear regression-based method to incorporate all query trends. In addition, I combine features derived from the two query trend methods and from previous pseudo trend methods in an ensemble approach. Experimental results suggest that query trend methods alone are competitive with the state-of-the-art pseudo trend methods, while combining both sources of evidence yields significant better results.

Multi-Perspective Lexical Modeling for Tweet Search. In Chapter 5, I introduce a novel neural ranking model for *ad hoc* retrieval over social media posts. This model is motivated by the ineffectiveness of existing neural models on tweet search, and addresses three major challenges in the social media domain: shorter document length, informal language use, and heterogeneous relevance signals. The model uses hierarchical convolutional layers with multiple input-level modeling to capture different relevance signals from queries, social media posts, as well as URLs contained in the posts – at the character-, word-, and phrase-levels. Extensive experiments demonstrate the effectiveness of the proposed model and ablation studies verify the source of effectiveness, suggesting that the customized architecture indeed captures the characteristics of the domain-specific ranking challenges.

Session Context Modeling for Voice Search. In Chapter 6, I introduce the novel problem of voice search on an entertainment platform, where user interacts with voice-enabled remote controller with voice requests to specify the TV programs to watch. This problem is formally defined as voice query navigation, where sessions are modeled as contexts to help disambiguate user's true intent and recover from ASR errors. I propose a hierarchical recurrent neural network (HRNN) model to integrate word- and character-level query representations and to model contextual dependencies in query sequences. The model not only demonstrates superior results against competitive baselines in an experimental setting, also is deployed into production and improves user experience on millions of queries per day.

Multi-Task Learning for Voice Search. In Chapter 7, I present a multi-task learning model to address the drawbacks of the HRNN model. By examining deployment logs, we see that the HRNN model suffers for predicting the newly-added and rarely-watched programs due to its classification setting. In addition, large-scale log analysis suggests that query understanding requires performing three related task simultaneously: program prediction, intent classification, and query tagging. Therefore, I articulate the design choices of each task and propose a novel neural architecture that *jointly* learns how to perform all three tasks. Evaluation on a large voice query log demonstrates how joint learning of the three tasks improves accuracy on each task individually. More importantly, the multi-task model provides the basis of an end-to-end system for handling queries that can draw from approximately one hundred different intents.

8.2 Future Work

In the rest of this chapter, I present a few directions that might be interesting to pursue in the future, complementing the work described throughout the dissertation.

Attention-based Neural Networks for Pseudo Trend Modeling. In the neural temporal framework we introduced in Chapter 3, each document contributes equally to the estimation. However, it's obvious that some documents are more important than others, thus deserve "special" treatments. For example, a highly-ranked documents can tell us more about the bursty interval of pseudo trend than a random document. This intuition also aligns with the superior results of the KDE approach with rank-based weightings (see Table 4.5). Thus I believe integrating attention mechanism to the temporal modeling can be promising.

Better Query Trend Modeling. In Chapter 4, we define the potential contribution of a query term by the burstiness of its query trend. We see two future explorations of this work. First, how to detect "bad" query trends can be promising. In the previous experiments, I observe there are some query trends that are bursty but also diverge from the true relevance distribution. This suggests measuring the goodness of a query trend only based on burstiness is not enough. Some other linguistic and text-based criteria can be proposed to filter out those bad trends. Second, incorporating query trend signal into query expansion techniques can be promising. Many expansion approaches (like RM3) refine the original query terms through a linear interpolation of the expanded term list.

Those expanded terms are selected by the frequency of their appearances in the top-ranked documents. In addition, original query terms are weighted equally in the interpolation. A possible extension is to consider query trend signal as another source for weighting query terms and selecting expanded terms. If we can come up with some effective strategy that map term bursty to relevance, we could expect improvements over traditional lexical feedback models.

Combine Relevance Matching with Semantic Matching. Relevance matching aims to match a query with a document with more emphasis on term co-occurrence. In comparison, semantic matching focuses more on understanding semantic meaning of a pair of texts, which have many applications in NLP, such as question answering, paraphrase detection, and reading comprehension. The differences in the nature of these two problems motivate divergences in their model designs. For example, the interaction-based approaches [9, 10] have been shown more effective on relevance matching problems, while representation-based approaches [30, 150] are more commonly adopted on semantic matching problems. A natural question is: would modeling relevance matching and semantic matching be complementary to each other? In fact, I have tried to extend the multi-perspective model in Chapter 5, which is an interaction-based method for relevance matching, to incorporate representation-based features for semantic matching. Preliminary experiments on tweet search and question answering tasks show that effectiveness improvements are indeed additive. Further experiments need to be performed to verify the generalizability of this idea and understand the inner working mechanisms.

Personalization with Periodic Patterns for Voice Search. Currently the HRNN and multi-task model described in Chapter 6 and 7 only consider session-level temporality for disambiguating user's real intent. Aside from this, long-term temporal patterns reflecting user's periodic behavior can be complementary signals. It's quite common that user watches TV in a periodic manner, such as watching an episode every Monday. Identifying these periodic patterns, including daily, weekly, and monthly patterns, could be helpful for disambiguation. Temporal models with attention mechanism are well-suited in these cases. However, another challenge is that a TV is often shared amongst the household, so the feasibility of reliable personalization is not as clear as on a smartphone or computer (i.e., not obvious low-hanging fruit). How to combine personalization and periodic patterns would be an interesting question to explore.

Bibliography

[1] Mark D Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *CIKM*, pages 623–632, 2007.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[4] Stephen E Robertson and Steve 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*, pages 232–241. Springer-Verlag New York, Inc., 1994.

[5] Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 403–410. ACM, 2001.

[6] Victor Lavrenko and W. Bruce Croft. Relevance based language models. In *SIGIR*, pages 120–127, 2001.

[7] Shipeng Yu, Deng Cai, Ji-Rong Wen, and Wei-Ying Ma. Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In *Proceedings of the 12th international conference on World Wide Web*, pages 11–18. ACM, 2003.

[8] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

[9] Jinfeng Rao, Wei Yang, Yuhao Zhang, Ferhan Ture, and Jimmy Lin. Multi-perspective relevance matching with hierarchical convnets for social media search. *arXiv preprint arXiv:1805.08159*, 2018.

[10] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *CIKM*, pages 55–64, 2016.

[11] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using click-through data. In *CIKM*, pages 2333–2338, 2013.

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[13] Nicholas Lester, Alistair Moffat, and Justin Zobel. Efficient online index construction for text databases. *ACM Transactions on Database Systems (TODS)*, 33(3):19, 2008.

[14] Anagha Kulkarni, Jaime Teevan, Krysta M Svore, and Susan T Dumais. Understanding temporal query dynamics. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 167–176. ACM, 2011.

[15] Rosie Jones and Fernando Diaz. Temporal profiles of queries. *ACM TOIS*, 25(3):Article 14, 2007.

[16] Miles Efron, Jimmy Lin, Jiyin He, and Arjen de Vries. Temporal feedback for tweet search with non-parametric density estimation. In *SIGIR*, pages 33–42. ACM, 2014.

[17] Anlei Dong, Ruiqiang Zhang, Pranam Kolari, Jing Bai, Fernando Diaz, Yi Chang, Zhaohui Zheng, and Hongyuan Zha. Time is of the essence: Improving recency ranking using twitter data. In *WWW*, pages 331–340, 2010.

[18] Jimmy Lin, Milad Gholami, and Jinfeng Rao. Infrastructure for supporting exploration and discovery in web archives. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 851–856. ACM, 2014.

[19] Na Dai and Brian D Davison. Freshness matters: in flowers, food, and web authority. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 114–121. ACM, 2010.

[20] Miles Efron. Linear time series models for term weighting in information retrieval. *Journal of the American Society for Information Science and Technology*, 2010.

[21] Jonathan L. Elsas and Susan T. Dumais. Leveraging temporal dynamics of document content in relevance ranking. In *WSDM*, pages 1–10, 2010.

[22] Miles Efron and Gene Golovchinsky. Estimation methods for ranking recent information. In *SIGIR*, pages 495–504, 2011.

[23] Wisam Dakka, Luis Gravano, and Panagiotis G. Ipeirotis. Answering general time-sensitive queries. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):220–235, 2012.

[24] Maria-Hendrike Peetz and Maarten de Rijke. Cognitive temporal document priors. In *ECIR*, pages 318–330, 2013.

[25] Iadh Ounis, Craig Macdonald, Jimmy Lin, and Ian Soboroff. Overview of the TREC-2011 Microblog Track. In *TREC*, 2011.

[26] Xiaoyan Li and W. Bruce Croft. Time-based language models. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM 2003)*, pages 469–475, New Orleans, Louisiana, 2003.

[27] Ilya Sutskever, Vinyals Oriol, and V. Le Quoc. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[28] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

[29] Jinfeng Rao, Hua He, Haotian Zhang, Ferhan Ture, Royal Sequiera, Salman Mohammed, and Jimmy Lin. Integrating lexical and temporal signals in neural ranking models for social media search. In *SIGIR Workshop on Neural Information Retrieval (Neu-IR)*, 2017.

[30] Hua He, Kevin Gimpel, and Jimmy J Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *EMNLP*, pages 1576–1586, 2015.

[31] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 373–382. ACM, 2015.

[32] Jinfeng Rao and Jimmy Lin. Temporal query expansion using a continuous hidden markov model. In *ICITR*, pages 295–298, 2016.

[33] Jinfeng Rao, Jimmy Lin, and Miles Efron. Reproducible experiments on lexical and temporal feedback for tweet search. In *Advances in Information Retrieval*, pages 755–767. Springer, 2015.

[34] Jinfeng Rao, Xing Niu, and Jimmy Lin. Compressing and decoding term statistics time series. In *ECIR*, 2016.

[35] Jinfeng Rao, Ferhan Ture, Xing Niu, and Jimmy Lin. Mining the temporal statistics of query terms for searching social media posts. In *ICTIR*, pages 133–140, 2017.

[36] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW*, pages 373–374, 2014.

[37] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In *AAAI*, pages 2793–2799, 2016.

[38] Greg Sterling. It's official: Google says more searches now on mobile than on desktop. `http://searchengineland.com/its-official-google-says-more-searches-now-on-mobile-than-on-desktop-220369`, 2015. Accessed: 2017-08-16.

[39] Sundar Pichai. Google i/o keynote, 2016.

[40] Martin Arlitt. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):50–63, 2000.

[41] Jinfeng Rao, Ferhan Ture, and Jimmy Lin. Multi-task learning with neural networks for voice query understanding on an entertainment platform. In *KDD*, 2018.

[42] Jinfeng Rao, Ferhan Ture, Hua He, Oliver Jojic, and Jimmy Lin. Talking to your tv: Context-aware voice search with hierarchical recurrent neural networks. In *CIKM*, pages 557–566, 2017.

[43] Jinfeng Rao, Ferhan Ture, and Jimmy Lin. What do users say to their tvs? an analysis of voice queries to an entertainment system. In *SIGIR*, 2018.

[44] Thorsten Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, pages 133–142, 2002.

[45] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136, 2007.

[46] Jinfeng Rao, Hua He, and Jimmy Lin. Noise-contrastive estimation for answer selection with deep neural networks. In *CIKM*, pages 1913–1916, 2016.

[47] Jinfeng Rao, Hua He, and Jimmy Lin. Experiments with convolutional neural network models for answer selection. In *SIGIR*, pages 1217–1220, 2017.

[48] Haotian Zhang, Jinfeng Rao, Jimmy Lin, and Mark D Smucker. Automatically extracting high-quality negative examples for answer selection in question answering. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 797–800. ACM, 2017.

[49] Royal Sequiera, Gaurav Baruah, Zhucheng Tu, Salman Mohammed, Jinfeng Rao, Haotian Zhang, and Jimmy Lin. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. *arXiv:1707.07804*, 2017.

[50] Jinfeng Rao, Jimmy Lin, and Hanan Samet. Partitioning strategies for spatio-textual similarity join. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pages 40–49. ACM, 2014.

[51] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[52] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, 1998.

[53] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 426–434. ACM, 2003.

[54] Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. Interval-based pruning for top-k processing over compressed lists. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 709–720. IEEE, 2011.

[55] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems (TOIS)*, 14(4):349–379, 1996.

[56] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. Vector-space ranking with effective early termination. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42. ACM, 2001.

[57] Jimmy Lin and Andrew Trotman. Anytime ranking for impact-ordered indexes. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 301–304. ACM, 2015.

[58] Matt Crane, J Shane Culpepper, Jimmy Lin, Joel Mackenzie, and Andrew Trotman. A comparison of document-at-a-time and score-at-a-time query evaluation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 201–210. ACM, 2017.

[59] Fredric C Gey. Inferring probability of relevance using the method of logistic regression. In *SIGIR*, pages 222–231, 1994.

[60] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581), 2010.

[61] Wisam Dakka, Luis Gravano, and Panagiotis G. Ipeirotis. Answering general time-sensitive queries. *TKDE*, 24(2):220–235, 2012.

[62] Mostafa Keikha, Shima Gerani, and Fabio Crestani. TEMPER: A temporal relevance feedback method. In *ECIR*, 2011.

[63] Olga Craveiro, Joaquim Macedo, and Henrique Madeira. Query expansion with temporal segmented texts. In *ECIR*, 2014.

[64] Jaeho Choi and W Bruce Croft. Temporal models for microblogs. In *CIKM*, 2012.

[65] Kira Radinsky, Krysta Svore, Susan Dumais, Jaime Teevan, Alex Bocharov, and Eric Horvitz. Modeling and predicting behavioral dynamics on the web. In *WWW*, pages 599–608, 2012.

[66] Milad Shokouhi and Kira Radinsky. Time-sensitive query auto-completion. In *SIGIR*, pages 601–610, 2012.

[67] Chao Zhang, Guangyu Zhou, Quan Yuan, Honglei Zhuang, Yu Zheng, Lance Kaplan, Shaowen Wang, and Jiawei Han. Geoburst: Real-time local event detection in geo-tagged tweet streams. In *SIGIR*, pages 513–522. ACM, 2016.

[68] Chao Zhang, Liyuan Liu, Dongming Lei, Quan Yuan, Honglei Zhuang, Tim Hanratty, and Jiawei Han. Triovecevent: Embedding-based online local event detection in geo-tagged tweet streams. In *KDD*, pages 595–604. ACM, 2017.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[70] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[71] Santiago Fernandez, Alex Graves, and Jurgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *International Conference on Artificial Neural Networks*, pages 220–229. Springer, 2007.

[72] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2009.

[73] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[74] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, pages 2673–2681, 1997.

[75] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[76] Hua He and Jimmy Lin. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *NAACL-HLT*, pages 937–948, 2016.

[77] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv:1512.05193*, 2015.

[78] Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Visual relationship detection with internal and external linguistic knowledge distillation. 2017.

[79] Ang Li, Jin Sun, Joe Yue-Hei Ng, Ruichi Yu, Vlad I Morariu, and Larry S Davis. Generating holistic 3d scene abstractions for text-based image retrieval. 2017.

[80] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*, pages 801–809, 2011.

[81] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *ACL*, 2017.

[82] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[83] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[84] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[85] Liu Yang, Qingyao Ai, Jiafeng Guo, and W Bruce Croft. anmm: Ranking short answer texts with attention-based neural matching model. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 287–296. ACM, 2016.

[86] Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D Manning. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45, 2017.

[87] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. Learning to attend via word-aspect associative fusion for aspect-based sentiment analysis. *AAAI*, 2018.

[88] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Multi-pointer co-attention networks for recommendation. *arXiv preprint arXiv:1801.09251*, 2018.

[89] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR*, pages 55–64, 2017.

[90] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM*, pages 126–134, 2018.

[91] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *WWW*, pages 1291–1299, 2017.

[92] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[93] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. Learning a deep listwise context model for ranking refinement. *arXiv:1804.05936*, 2018.

[94] Ye-Yi Wang, Dong Yu, Yun-Cheng Ju, and Alex Acero. An introduction to voice search. *IEEE Signal Processing Magazine*, 2008.

[95] Alex Acero, Neal Bernstein, Rob Chambers, Yun-Cheng Ju, Xinggang Li, Julian Odell, Patrick Nguyen, Oliver Scholz, and Geoffrey Zweig. Live search for mobile: Web services by voice on the cellphone. In *ICASSP*, 2008.

[96] Junlan Feng and Srinivas Bangalore. Effects of word confusion networks on voice search. In *EACL*, 2009.

[97] Ciprian Chelba and Johan Schalkwyk. Empirical exploration of language modeling for the google. com query stream as applied to mobile voice search. In *Mobile Speech and Advanced Natural Language Solutions*. 2013.

[98] Jiulong Shan, Genqing Wu, Zhihong Hu, Xiliu Tang, Martin Jansche, and Pedro J Moreno. Search by voice in mandarin chinese. 2010.

[99] Fabio Crestani and Heather Du. Written versus spoken queries: A qualitative and quantitative comparative analysis. *JASIST*, 2006.

[100] Jeonghe Yi and Farzin Maghoul. Mobile search pattern evolution: the trend and the impact of voice queries. In *WWW*, 2011.

[101] Ido Guy. Searching by talking: Analysis of voice queries on mobile web search. In *SIGIR*, 2016.

[102] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope. "your word is my command": Google search by voice: A case study. In *Advances in Speech Recognition*. 2010.

[103] Jiepu Jiang, Wei Jeng, and Daqing He. How do users respond to voice input errors?: lexical and phonetic query reformulation in voice search. In *SIGIR*, 2013.

[104] Ahmed Hassan Awadallah, Ranjitha Gurunath Kulkarni, Umut Ozertem, and Rosie Jones. Characterizing and predicting voice query reformulation. In *CIKM*, 2015.

[105] Milad Shokouhi, Rosie Jones, Umut Ozertem, Karthik Raghunathan, and Fernando Diaz. Mobile query reformulations. In *SIGIR*, 2014.

[106] Milad Shokouhi, Umut Ozertem, and Nick Craswell. Did you say u2 or youtube?: Inferring implicit transcripts from voice search logs. In *WWW*, 2016.

[107] Rich Caruana. Multitask learning. *Machine Learning*, pages 41–75, 1997.

[108] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv:1511.06114*, 2015.

[109] Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Visual relationship detection with internal and external linguistic knowledge distillation. In *ICCV*, pages 1974–1982, 2017.

[110] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv:1707.08114*, 2017.

[111] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167, 2008.

[112] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *WWW*, 2009.

[113] Donald Metzler and W Bruce Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[114] Peter Hall and Berwin A Turlach. Reducing bias in curve estimation by use of weights. *Computational statistics & data analysis*, 30(1):67–86, 1999.

[115] Hua He, John Wieting, Kevin Gimpel, Jinfeng Rao, and Jimmy Lin. UMD-TTIC-UW at SemEval-2016 Task 1: Attention-based multi-perspective convolutional neural networks for textual similarity measurement. In *SemEval*, pages 1103–1108, 2016.

[116] Jeff A Bilmes et al. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical report, International Computer Science Institute, 1998.

[117] Iadh Ounis, Craig Macdonald, Jimmy Lin, and Ian Soboroff. Overview of the trec-2011 microblog track. In *TREC*, volume 32, 2011.

[118] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[119] Nasreen Abdul-Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Donald Metzler, Mark D. Smucker, Trevor Strohman, Howard Turtle, and Courtney Wade. UMass at TREC 2004: Novelty and HARD. In *TREC*, 2004.

[120] Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *CIKM*, pages 623–632, 2007.

[121] Gilad Mishne, Jeff Dalton, Zhenghua Li, Aneesh Sharma, and Jimmy Lin. Fast data in the era of big data: Twitter's real-time related query suggestion architecture. In *SIGMOD*, 2013.

[122] Hugh E Williams and Justin Zobel. Compressing integers for fast file access. *The Computer Journal*, 42(3):193–201, 1999.

[123] Vo Ngoc Anh and Alistair Moffat. Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8(1):151–166, 2005.

[124] Jiangong Zhang, Xiaohui Long, and Torsten Suel. Performance of compressed inverted list caching in search engines. In *WWW*, 2008.

[125] David A Huffman et al. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[126] Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth JF Jones. Word embedding based generalized language model for information retrieval. In *SIGIR*, pages 795–798, 2015.

[127] Victor Lavrenko and W Bruce Croft. Relevance based language models. In *SIGIR*, pages 120–127, 2001.

[128] Donald Metzler and W Bruce Croft. A markov random field model for term dependencies. In *SIGIR*, pages 472–479, 2005.

[129] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv:1506.01066*, 2015.

[130] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281. ACM, 1998.

[131] Christopher Burges, Krysta Svore, Paul Bennett, Andrzej Pastusiak, and Qiang Wu. Learning to rank using an ensemble of lambda-gradient models. In *Proceedings of the Learning to Rank Challenge*, pages 25–35, 2011.

[132] Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM*, 2008.

[133] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. Context-aware query classification. In *SIGIR*, 2009.

[134] Paul N Bennett, Ryen W White, Wei Chu, Susan T Dumais, Peter Bailey, Fedor Borisyuk, and Xiaoyuan Cui. Modeling the impact of short-and long-term behavior on search personalization. In *SIGIR*, 2012.

[135] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 553–562. ACM, 2015.

[136] Dongyi Guan, Sicong Zhang, and Hui Yang. Utilizing query change for session search. In *SIGIR*, 2013.

[137] Jingjing Liu and Nicholas J Belkin. Personalizing information retrieval for multi-session tasks: The roles of task stage and task type. In *SIGIR*, 2010.

[138] Bhaskar Mitra and Nick Craswell. Neural models for information retrieval. In *arXiv:1705.01509v1*, 2017.

[139] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, 2017.

[140] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[141] Tijmen Tieleman and G Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Tech. Rep*, 2012.

[142] Thorsten Joachims. Training linear SVMs in linear time. In *SIGKDD*, pages 217–226, 2006.

[143] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *SIGIR*, pages 105–114, 2011.

[144] Hank Liao, Golan Pundak, Olivier Siohan, Melissa K Carroll, Noah Coccaro, Qi-Ming Jiang, Tara N Sainath, Andrew Senior, Françoise Beaufays, and Michiel Bacchiani. Large vocabulary automatic speech recognition for children. In *Interspeech*, 2015.

[145] Lingpeng Kong, Chris Alberti, Daniel Andor, Ivan Bogatyy, and David Weiss. Dragnn: A transition-based framework for dynamically connected neural networks. 03 2017.

[146] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[147] Ramakanth Pasunuru and Mohit Bansal. Multi-task video captioning with video and entailment generation. *arXiv:1704.07489*, 2017.

[148] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. *arXiv:1704.05742*, 2017.

[149] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pages 363–370, 2005.

[150] Phu Mon Htut, Samuel R Bowman, and Kyunghyun Cho. Training a ranking function for open-domain question answering. *arXiv preprint arXiv:1804.04264*, 2018.