OPTIMIZATION OF AN EDUCATIONAL SEARCH ENGINE USING LEARNING TO RANK ALGORITHMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Arif Usta

September, 2015

Optimization of an Educational Search Engine Using Learning to Rank Algorithms
By Arif Usta
September, 2015

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy(Advisor)

Prof. Dr. Uğur Güdükbay

Assist. Prof. Dr. İsmail Sengör Altıngövde

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

OPTIMIZATION OF AN EDUCATIONAL SEARCH ENGINE USING LEARNING TO RANK ALGORITHMS

Arif Usta
M.S. in Computer Engineering
Advisor: Prof. Dr. Özgür Ulusoy
September, 2015

Web search is one of the most popular internet activities among users. Due to high usage of search engines, there are huge data available about history of user search issues. Using query logs as a source of implicit feedback, researchers can learn useful patterns about general search behaviors. We employ a detailed query log analysis provided by a commercial educational vertical search engine. We compare the results of our query log analysis with the general web search characteristics. Due to difference in terms of search behavior between web users and students, we propose an educational ranking model using learning to rank algorithms to better reflect the search habits of the students in the educational domain to further enhance the search engine performance. We introduce novel features best suited to the educational domain. We show that our model including educational features outperforms two baseline models which are the original ranking of the commercial educational vertical search engine and the model constructed using the state of the art ranking functions, up to 14% and 11%, respectively. We also employ different learning to rank models for different clusters of queries and the results indicate that having models for each cluster of queries further enhances the performance of our proposed model. Specifically, the course of the query and the grade of the user issuing the query are good sources of feedback to have a better model in the educational domain. We propose a novel *Propagation Algorithm* to be used for queries having lower frequencies where information derived from query logs is not enough to exploit. We report that our model constructed using the features generated by our proposed algorithm performs better for singleton queries compared to both the educational learning to rank model we introduce and models learned with common features introduced in the literature.

Keywords: Information Retrieval, Web Search, Vertical Search Engine, Learning to Rank Algorithms, Educational Domain.

ÖZET

SIRALAMA AMAÇLI ÖĞRENME ALGORİTMALARI KULLANARAK EĞİTİM TABANLI ARAMA MOTORU OPTİMİZASYONU

Arif Usta
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Prof. Dr. Özgür Ulusoy
Eylül, 2015

İnternet üzerinden arama yapmak, İnternet kullanıcılarının tercih ettiği en popüler aktivitelerden birisidir. Arama motorlarının oldukça fazla kullanılması sayesinde, kullanıcıların yapmış oldukları aramaların yer aldığı bilgilerin kullanılabilmesi oldukça kolaylaşmıştır. Araştırmacılar bu bilgileri kullanarak kullanıcıların sahip oldukları arama davranışları ile ilgili faydalı sonuçlar elde etmişlerdir. Çalışmamızda, ticari olarak kullanılan eğitim amaçlı arama motoruna ait sorgu bilgilerini detaylı bir şekilde analiz ettik. Analiz sonrası aldığımız sonuçları, İnternet kullanıcılarının sahip olduğu arama davranışlarıyla karşılaştırarak aralarındaki farkları belirledik. Bu farkları da göz önünde bulundurarak eğitim alanına daha iyi uyum sağlayacak, sıralama amaçlı öğrenme algoritmalarını kullanarak bir model ortaya çıkardık. Sahip olduğumuz sorgu kümesi öğrencilere ait olduğu için, modeli oluştururken eğitim alanına has özgün özellikler kullandık. Modelin performansını karşılaştırmak adına, sahip olduğumuz sorgu kütüklerindeki orjinal sıralamayı ve bu alanda sıklıkla kullanılan modern yöntemlerle oluşturulan modeli referans olarak aldık. Sonuçlarımıza göre her iki referans modeline kıyasla sırasıyla %14 ve %11'lik bir gelişme sağladık. Bunun yanında, yapılan sorgunun türlerine göre ayrılmak kaydı ile, her bir sorgu grubu için farklı öğrenme modelleri tanımladık. Elde ettiğimiz verilere göre, sorgunun ders bilgisinin ve sorguyu soran öğrencinin sınıf bilgisinin farklı model oluştururken oldukça fayda sağladığını öğrendik. Sorgu kütük kümesinde yer alan bazı sorgular (yalnızca bir defaya mahsus sorulan sorgular) için elimizde yeteri kadar bilgi olmadığından, modelin genel performansını artırmak adına bu sorgular için özgün bir algoritma geliştirdik. Yaptığımız deneylere göre, bu özel sorgular için geliştirdiğimiz algoritma, oluşturduğumuz genel modelin performansını da artırmaktadır.

Anahtar sözcükler: Bilgi Erişimi, Dikey Arama Motorları, Sıralama Amaçlı Öğrenme Algoritmaları, Eğitim Alanı.

Acknowledgement

First and foremost, I would like to thank my supervisor, Prof. Dr. Özgür Ulusoy, for his invaluable guidance during my work. His support and dedication helped me to be motivated throughout my academic research.

I am deeply thankful to Prof. Dr. Uğur Güdükbay and Assist. Prof. Dr. İsmail Sengör Altıngövde for kindly accepting to read and review this thesis.

I would also express my gratitude to Assist. Prof. Dr. Rıfat Özcan for his insightful comments during my work.

I would like to acknowledge the scholarship provided by The Scientific and Technological Research Council of Turkey (TUBITAK) throughout my graduate education under BİDEB program. I would also acknowledge TUBITAK for supporting me financially for this work with grant number 113E065.

I would like to use this opportunity to thank SEBIT for providing us the data we used throughout this thesis.

Finally, I would like to dedicate this thesis to my family. Thank you for your endless love, understanding and belief in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Scope

Document retrieval has been one of the hottest research topics in Information Retrieval. The problem of document retrieval gets more interesting when web search is the activity to retrieve the documents due to characteristics of web users. Web users tend to leave the search engine by only considering documents appearing in the first page while avoiding all documents displayed at latter pages, which makes document retrieval problem more challenging. State of the art document retrieval methods do not take the order of the documents into consideration. Therefore, a new approach is introduced, called learning to rank, which is basically a machine learning approach with the purpose of ranking.

Recently, various works have been performed on the subject of learning to rank algorithms, including different types of machine learning algorithms involving different approaches in terms of preparing the training and test datasets and the annotation methodology to be used. The learning to rank methodology is a supervised machine learning process, therefore it requires an annotation of the instances to be used in the learning phase. Depending on the approach used, generally the learning to rank algorithms are modified versions of either

classification or regression problems. However, the only and the most important difference with the learning to rank algorithms is that they are not interested in the final classified or regressed class value but rather deal with the order of instances classified or regressed.

Since the learning to rank algorithms are basically machine learning algorithms, what determines their performance is the choice of features to be used in the learning. In the literature of web search, numerous features are introduced. Yet, the choice of the features highly depends on the environment of the search including the domain of the search, the context of the documents available, the characteristics of the users making searches, etc. Therefore, generating the features to be used in the learning is the core of learning to rank algorithms and mostly query logs are good sources of implicit feedback about user behavior while figuring out the possible features to be extracted.

In this thesis, we try to exploit a query log provided from a commercial educational vertical search engine. We carry out detailed analysis on the query log to find out user behaviors to further use them as features in the model we propose using the learning to rank algorithms.

1.2 Contributions

Our contributions in the thesis are three fold, which are explained as follows:

- We perform a detailed analysis on a query log provided by an educational vertical search engine. We compare our findings with the general search behaviors web users have. We show that students in the scope of educational domain have different search habits and therefore the problems defined in the information retrieval area should be addressed accordingly. Specifically, we also try to find the *Refinding* behavior in our query log, that is, we analyze how often students tend to find the documents they once visited while searching for an education material with the goal of learning.

- Using the learning to rank algorithms, we employ learning models to rank the educational materials provided in our query log. Analyzing the query log, we use our findings to further improve the learning model to better capture the search behaviors of students. We introduce novel learning to rank features specifically better suited for the educational domain. We employ a learning model designed for educational domain and show that our model outperforms both the original ranking of the documents and baselines created using the state of the art ranking functions in the literature in terms of search engine performance. We also employ different models for query types by clustering queries regarding their different properties, which are namely the course of the query, the grade of the user issuing query and the query frequency. We show that using different models considering query types improves the search engine performance compared to the general model we introduce.

- We propose a feature generation algorithm to be used specifically for queries having low frequencies, where query log fails to give useful and adequate feedback to exploit in the learning phase. We also implement a similar idea of generating feature values introduced in [1] to make a comparison. We report that our proposed algorithm outperforms both the referenced model and the general model for queries having a single issue (singleton queries).

1.3 Thesis Organization

The rest of the thesis is organized as follows:

In Chapter 2, we review the early work done on both query log analysis and the web search problems using learning to rank algorithms. For each category, we first consider the general works in the literature and then give examples of more specific works. We put more emphasis on works that are closely related to our work.

We present the results of our detailed query log analysis in Chapter 3. We

categorize our findings into two categories which are *Search Characteristics* and *Refinding Behavior*. In the first section, we analyze the search habits of the users and compare our findings with the users in general web search. In the latter section, we try to analyze the *Refinding* behavior students have and explain our results with their reasoning regarding the educational domain.

In Chapter 4, we explain the learning to rank algorithms. We give their structure to use the algorithms and explain how to obtain the learning setup to employ a better model. We briefly discuss the approaches used in the learning to rank environments. Then, we provide the evaluation metrics that are used in the literature to evaluate proposed models.

In Chapter 5, we provide our proposed educational learning to rank model. We give our constructed features along with their correspondence category and explain how each feature is calculated. We discuss the cluster models generated regarding the different query types we have in our query log. We also provide our *Propagation Algorithm* to be used for singleton queries to further improve the model we learned.

We report the results of our experiments in Chapter 6. We compare our findings with state of the art baselines and show that our proposed models improve search engine performance significantly. We conclude the thesis in Chapter 7.

Chapter 2

Related Work

2.1 Query Log Analysis

Due to immense online information available on the Web, online search is one of the most popular internet activities users prefer. According to statistics available on [2], the web sites providing search engine service are among the top visited sites. *Google* is at the top of the list while Yahoo and Baidu (the leading search engine provider in China) are among the top five. As a result, there are huge data available to exploit to learn some patterns. Therefore, in recent years there has been a growing interest in analyzing query logs of search engines to find out possible search characteristics that may lead to different research aspects. The idea behind the query log analysis is to figure out how web users deploy a search engine. In other words, it is a tool to find answers for the rel ated research questions such as: What do users search in Web? What are the characteristics of the user using search engines? How do they search? Answering such questions may lead researchers to new research aspects in web search area, where they can use the results from the query log analysis to further exploit potential solutions.

One of the very first large scale query log analysis papers explores search characteristics of the search engine users using query logs gathered from AltaVista

[3]. They did their calculations on a query log consisting of 1 billion queries over a period of six week time. Their findings include that queries mostly consist of 1-3 query terms, the most popular query terms in the query log are generally sexual related terms and most sessions include single query, which means users are unlikely to reformulate their queries as a result of a possible unsatisfied query issue. They also state that users tend look at only the first page, therefore they indicate that traditional information retrieval techniques may fail in the scope of the web search.

Queries are what users use for defining their search intents through search engines, therefore they are the answers for what users search in Web. One of the research aspects in search is to determine user intents. Hence, it is important to classify the user intents if possible prior to retrieving the search results to be shown to the user. One of the early works on this problem [4] tries to classify the queries according to their intents, which are mainly informational, navigational and transactional. Informational queries are described as queries that cover broad range of topic for which there may thousands of relevant documents, while navigational and transactional queries are defined as queries carrying an intent of finding a particular web site and completing a transaction such as downloading a video, respectively. An exploration of query log data reveals that most queries in a search engine are of informational $(40 - 50\%)$, followed by transactional ones $(30 - 36\%)$.

Session is another important aspect of the search. The decision of how to choose session highly affects the certain behaviors of the search engine. According to Spink et al. [5], a search session is described as the time interval between the first query issued by the user and the time the user leaves the search engine. Time based search session detection techniques are among the most popular ones. Generally, a cut of value (threshold value) is used to determine whether two consecutive search issues by a particular user belongs to the same session or not. Jansen et al. [6] use 30 minutes as cut off value to determine search sessions in and they state that the cut off value they introduced is better to explore the different query types a user issues in a single search session.

In recent years, there is a growing interest in analyzing the vertical search engines which are used to search for particular type of information. These vertical engines are important to gather results to use them in general ones. One of the early works on vertical search engines is done by Lawrence at al. [7], in which they try to retrieve scientific documents only. Weerkamp at al. [8] try to find search characteristics of a commercial vertical search engine in which users search for people only. We followed the practices they made to compare our findings with theirs and general Web as well.

2.2 Refinding Analysis

One of the problems in web search is the behavior of the users having intent to find out the documents that they once visited, which is called *Refinding*. There have not been many works on this subject, until it is noted in [9] that, 17% of the web users surveyed reported that "Not being able to find out a page once I visited" is one of the biggest problems to be solved in web search.

Cockburn at al. distill several years of knowledge on the Refinding behavior to improve how people return to their previously visited pages in [10]. Being one of the early extensive studies on Refinding, they state that their results indicate revisitation in Web is one of the most dominant activities, with an average of 4 out of 5 page visits being to previously seen pages. They also introduce three different interfaces to help users to find out their previously visited pages when desired using *back* and *forward* buttons. They also strongly suggest that using temporally ordered lists of previously visited pages can significantly improve the revisitation in Web.

People often repeat Web searches whether to find new information for a topic they have already explored or to find what they once found about a particular topic. Queries associated with a repeat behavior may have different texts, yet they lead users into same clicks. Teevan et al. [11] demonstrate that as many as 40 % of all queries lead users into repeat behavior. They describe the intent of

user issuing the search with categorization of different types to be derived using query string and click-through sets.

Sanderson et al. [12] confirmed refinding behavior observed by Teevan et al.[11] and extended their work to include temporal properties of repeat searches and clicks. They indicate that users show seven day and 24 hour periodicities in their search, which is also consistent throughout entire period of time of the query log. Queries issued repeatedly from different users tend to show the information need of users for a particular temporary event or news.

2.3 Learning to Rank - LETOR

In recent years, there has been significant works on learning to rank for information retrieval. Many powerful algorithms have been developed for learning to rank information retrieval and some of them have been applied to the some particular problems such as web search. One of the indepth papers on this area is published by Qin et al. [13]. They constructed a benchmark dataset called *LETOR* to be used by the researchers interested in learning to rank in the scope of information retrieval. They introduced the features to be used in learning algorithms and they categorized their features into four groups, which are low-level content features (e.g., tf*idf [14]), high-level content features (e.g., BM25 [15] and LMIR [16]), hyperlink features (e.g., PageRank [17]), and hybrid features. In total, they extracted 44 features, which are still mainly used by the researches in the learning to rank setups. Besides, they discussed data partitioning methods to split the data set to be used as training and test separately and they provided with the performance results of the benchmark dataset using precision and NDCG metrics.

There is a book published by Li [18] devoted to the learning to rank for information retrieval area. The algorithms and their corresponding approaches, which are pointwise, pairwise and listwise, are explained in an exhaustive manner. Learning to rank can be applied to many problems arising in the area of

information retrieval, specifically web search. Giannopoulos et al. [19] try to find out the user intent using learning to rank algorithms. They state that user intents in web search are subject to change and hard to capture. In order to capture the user intent, they exploit the click-through feedback of users using the learning to rank approaches. They cluster the user intents considering the query log data to learn different ranking models for each cluster. They assert that their cluster models significantly outperform the baseline which is a single model.

Query auto-completion (QAC) is one of most prominent features in modern search engines, which makes it an important research topic in information retrieval area. In [20], Shokouhi introduces a supervised framework for personalizing auto-completion ranking. He compares existing features including user-specific and demographic features to find out their effectiveness in the scope of personalization of auto-completion. He concludes that user's long search history and location information perform best among the features used. Besides, he asserts that his ranking models supported by personalization features outperforms the existing popularity based baselines, in terms of mean reciprocal rank (MRR) by up to 9%.

One of the hot topics in information retrieval related to the learning to rank methodologies are *click models*. In a learning to rank environment for a model to be learned, an annotation must be done since it is a supervised learning process. Making relevance annotation by judges is of high cost, which leads the researchers to come up with models that try to predict the relevance of the documents automatically with high effectiveness.

In one of the early works on click models, Dupret et. al. [21] propose a set of assumptions made on user browsing behavior that allows predicting the relevance of the documents using estimation of the probability that a document is seen. They try to estimate the probability of a result document being clicked as the ratio of the number of times a user clicked on the document to the expected number of times the document is examined, instead of pure click through rate (CTR) values. Apart from the position and the cascade model [22], they give a baseline model, which only depends on the attractiveness of the document (URL,

snippet etc.) They also state that, the cascade model outperforms significantly the other models in explaining the clicks at higher ranks. On the other hand at lower ranks, it is slightly worse than the other models, including the baseline model they proposed.

Click through can provide an important source of user feedback and therefore can be used to determine the relevance labels of the documents using clicks of users rather than editorial judgment. However, due to the *position bias*, documents appearing at lower ranks are less likely to be clicked, which makes it difficult to capture the relevance information for such documents.

Chapelle at al. [23] propose a Dynamic Bayesian Network to give an unbiased estimation of the relevance from click logs. Their model tries to combine the advantages of both the position model and the cascade model. They claim that their model is similar to the position model in the sense that a click occurs if and only if the user examines the URL and finds it relevant. Similar to the cascade model, their model assumes that user sequentially looks at all the results and clicks based on the perceived relevance. The user keeps examining the next URLs if he is not satisfied with the current URL (based on actual relevance). The difference between their proposed model and the cascade model is that click does not necessarily mean that document is relevant. Also, there is no limit in terms of click numbers in their model. They assert that their experiments show that their proposed model outperforms existing click models in predicting both click-through data and relevance.

Clicks on search results are helpful source determining the user satisfaction in a search session. Yet, click information can be noisy due to position and caption bias and some other factors. A common approach to remove the noisy clicks may have is to use 30 seconds threshold [24] to categorize clicks as satisfied (SAT) or dissatisfied (DSAT) clicks. However, Kim et al. [25] claim that using a single threshold value (e.g., 30 seconds) to determine whether the user is satisfied by the click is not rational due to different page characteristics. They state that topic of the page, its readability level and its length are crucial in determining the amount of dwell time needed to figure out whether any click can be labeled

as satisfied.

To understand whether a user is satisfied with the current search results, implicit behavior is a useful data source, with clicks being the best-known implicit signal. Yet, it is possible that a non-clicking user to be satisfied and a clicking user to be dissatisfied. In [26], Hassan et al. used the user behavior among search sessions as the implicit signal to determine whether a click is satisfied or not. Specifically, they focused on reformulation behavior, which is defined as consecutive queries having similarities in textual content and interior time less than a certain threshold. They claim that their query based model with additional query specific features is better to figure out user satisfaction than the click based models.

The click through data has been used in various subjects in web search using the learning to rank algorithms. However, there is a limit to the usage of click logs when there is not enough information about some particular queries in the query log. This problem generally occurs for the queries having low frequencies, called tail queries. In recent years, researchers have started to focus on this issues trying to generalize learned models to perform well enough for tail queries as well. Aktolga et al [27] try to boost rarely clicked queries in a system where limited click-through data are available. They proposed a probabilistic approach to re-rank the result list of documents for sparse queries. They utilized the information of co-click queries, which share clicks on the same documents. They try to generate click-through features using the set of queries similar to the given query which has no to little click data available. They categorized similar queries into three groups, which are similar queries that share at least one co-click, synonym queries that are lexically related to each other and subset queries where one is included in the other as a subset. They assert that their models using three sets of similar query sets are significantly better than the Lucene based baseline model.

Incorporating click-through features using query logs to the learning to rank algorithms is useful to have better performing model, yet there may be some occasions where click-through features cannot be calculated. Gao et al. [1] introduce

clickthrough stream as the set of queries having co-click for a particular document in the query log given a certain document. Their calculation of click-through features differs from the ones in the literature in the sense that they also consider whether a click is the last click in the search session to give more importance for the documents that are clicked last.

They further try to improve the performance of features by using two smoothing techniques, which are Discount method and Random Walk. There are certain queries in the query log whose result lists include documents that have insufficient clickthrough stream that is not big enough to calculate the features effectively. For those queries, they use random walk approach to generate artificial connections between the query having a sparse click through stream for associated documents in the result list and other random queries. However, this method is not applicable for queries that have zero click in the query log and therefore zero queries in their click-through streams. For those queries, they simply try to estimate the values of the click through features, initially calculated as zero prior to Discount method, by using the values of the features of the queries having a single query in their clickthrough stream. The motivation behind this idea is to reduce the penalization of the ranker training instances which have zero valued features. They report that using Discount method and Random Walk along with their combination they further improve the model learned by the state of the art ranking functions by up to 4%

Chapter 3

Query Log Analysis

Search is a key web activity among all kinds of users towards a large variety of goals. While the lion's share of previous works on query analysis focus on general web search, the need for analyzing the search behavior of certain user groups and/or users searching for a certain type of information has emerged as an important research direction. Recent studies show that children and teenagers, who constitute a large and dynamic subset of web users, deserve special attention as their search behavior differ from the adults in several ways while using search engines [29, 30, 31]. Other studies address alternative search tasks that are usually carried out via verticals, and analyze query logs obtained from the systems specialized for digital libraries, audio-visual archives and searching people on the web [8].

In this chapter[1], we analyze the query logs of a commercial educational content developer and service provider for Turkish students at K-12 level. Turkey has the youngest population in Western Europe (by median age) and 42.9% of its total population, which is estimated to be around 77 millions as of December 2013, is young, i.e., under 24 years old. According to national statistics, the number

[1]This chapter is based on the work [28] published in Proceedings of the 37th International ACM SIGIR Conference. SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia. http://dx.doi.org/10.1145/2600428.2609532.

of students at primary and secondary schools adds up to 16,156,519 (excluding pre-school and open-education students) [32]. Not surprisingly, there are several governmental and industrial efforts to develop education services and products targeting this young and dynamic population. Vitamin™ is a commercial web-based educational framework that provides interactive content and performance assessment mechanisms for a large variety of courses covered in K-12 curriculum in Turkey. As of December 2013, Vitamin has more than 1.2 million registered users and about 4.3 million site visits per month. These users can utilize the navigational interface to reach to the content they need, or they can perform search over the entire set of educational materials.

3.1 Search Characteristics

Following the practice in [8], we provide the characteristics of search in Vitamin with respect to four major dimensions; namely, queries, sessions, users, and clicked results. We also compare and contrast our findings to those on general web search engines and/or earlier results on children's search behaviors. Our analysis helps understanding how students search with the purpose of learning in an educational vertical, and reveals new directions to improve the search performance in the education domain.

Vitamin search engine allows users to issue a keyword query along with a number of category filters, namely, content type, grade, and course filters. Figure 3.1 shows the GUI of the Vitamin's search system for the query "carbon dioxide". Then, users can click and display a particular query result, which is called a *learning object* and presented in text and/or audio-visual formats; or navigate to certain point in a topic hierarchy where this learning object belongs to. The system stores the queries and clicked results in the search log, while the navigational type of interaction is recorded separately as a different kind of event. Therefore, our preliminary analysis here involves a query log that includes a sample from the queries submitted to Vitamin's search system in December 2013 by the logged-in users (i.e., with paying or trial accounts), and followed by at least one click on

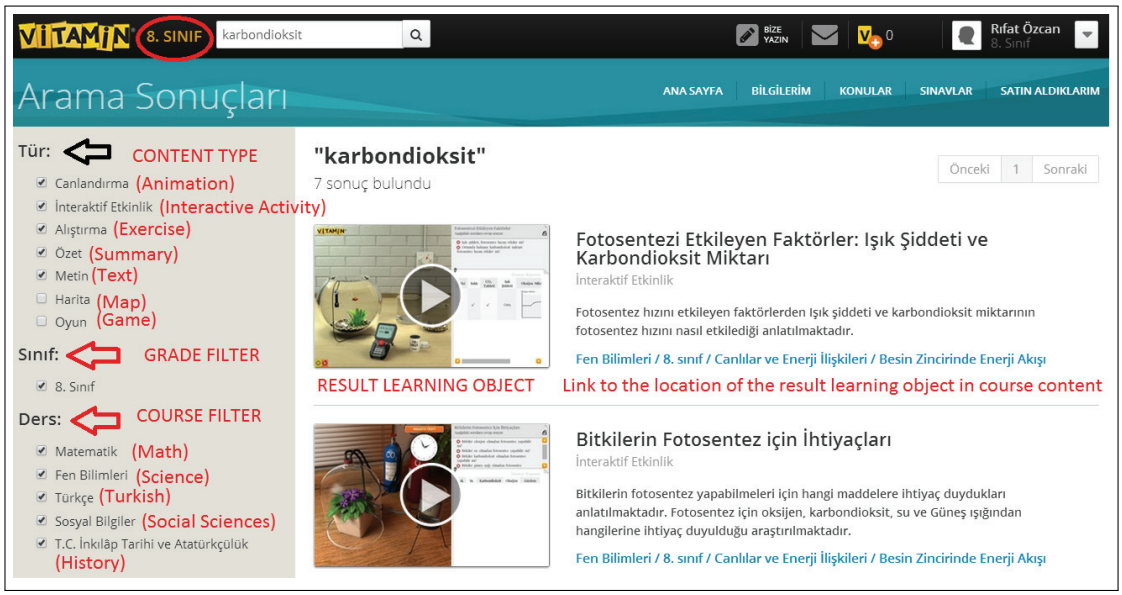

Figure 3.1: Vitamin search GUI for the query *carbon dioxide* (with annotations in English)

the displayed results.

3.1.1 Query Characteristics

According to Table 3.1, 27.8% of the query volume are unique queries and 69.3% of the latter are singletons, i.e., asked only once. These values differ from the web search trends, where 50% of the queries in a typical search log are unique and 88% of them are singletons [33]; and more similar to the trends obtained for a vertical for searching people [8]. This means that the queries are more likely to be repeated in this educational search engine, which is a good news for the mechanisms that exploit temporal locality, such as caching. On the other hand, distribution of query frequencies shown in Figure 3.2 confirms the power law distribution characteristics as in the case of web search [33].

On the average, a query includes 2.16 terms, which is slightly shorter than typical web queries (around 2.5 terms as reported in [33]) as well as the queries submitted to a major web search engine by the users between 10 and 18 years old

Table 3.1: Query characteristics

Number of queries	66,908	
Number of unique queries	18,638	(27.8%)
Number of singleton queries	12,926	(19.3%)
Average number of queries per day	2,230	
Busiest day in number of queries	3,855	
Average number of terms per query	2.16	
Average number of users per query	3.58	
Average number of results per query	114	

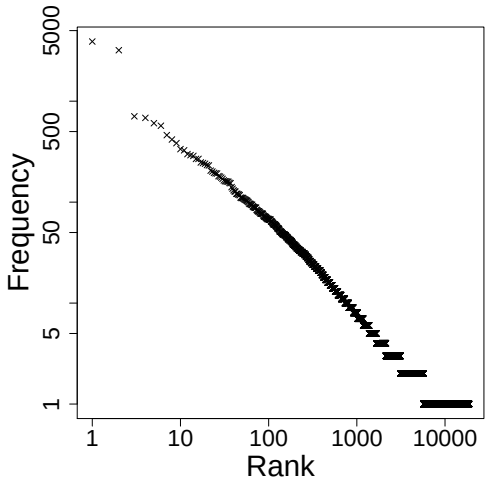

Figure 3.2: Distribution of query frequencies. The x-axis represents the rank according to the query frequency in the plot.

(around 2.6 terms [29]). This difference might be attributed to the fact that the educational search setup is a more restricted domain than web and even a couple of terms can yield the relevant resources from the available content.

Table 3.2 lists top-10 most frequent queries, which yields interesting findings. First, top-2 queries are "games" and "game", which means that the students enjoy the educational games provided by this system. Among the remaining 8 queries, 3 of them are simply the course names and too general to be useful (i.e., "science", "math", "Turkish"). This implies that the students who want to find a certain course still use the search box, rather than browsing through the list of courses. The other popular queries are related to Turkish and Math courses, and might be related to the topics that are being discussed in these courses at this time of the year.

Table 3.2: Top-10 popular queries in terms of the query frequencies and unique users.

Query	**Frequency**	**Users**
oyunlar (*games*)	3898	2290
oyun (*game*)	3197	1576
fen (*science*)	708	320
zarflar (*adverbs*)	683	466
trke (*Turkish*)	605	344
matematik (*math*)	571	368
fiilde atı (*verb forms*)	461	321
ses bilgisi (*phonetics*)	417	248
standart sapma (*standard deviation*)	384	309
olasılık (*probability*)	335	249

As mentioned before, Vitamin's search interface allows setting various filters along with a query, which we analyze next. Figure 3.3 shows the distribution of content type filters selected while submitting queries. It is seen that all content types are selected in the majority of the queries, which is the default setting in the GUI. This means that users leave this filter as-is most of the time, probably because they want to see all available content relevant to their query. We observe similar trends for the use of course filter, as shown in Figure 3.5.

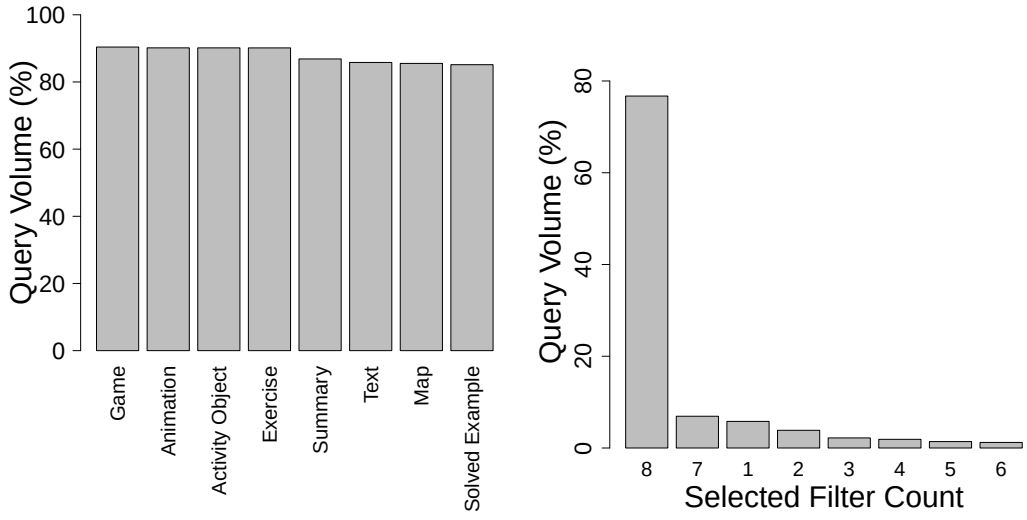

Figure 3.3: Distribution of content type filters used in queries.

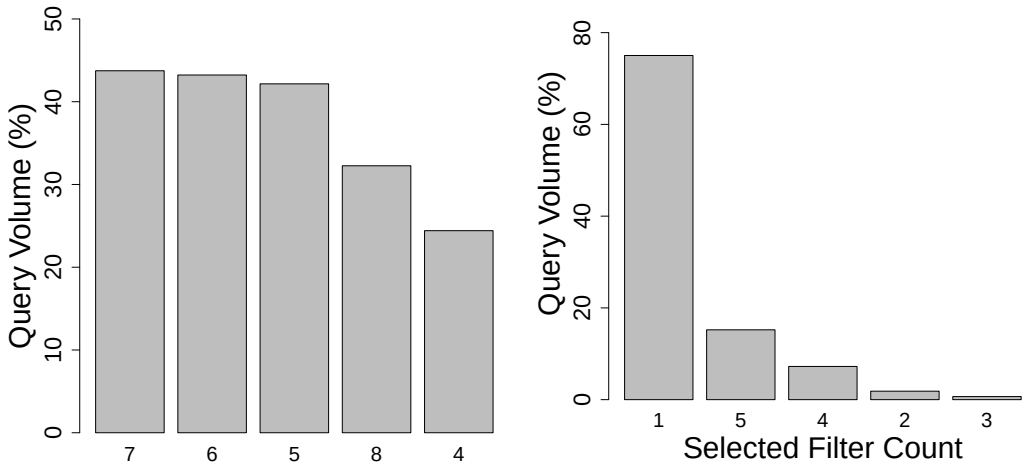

Figure 3.4: Distribution of grade filters used in queries.

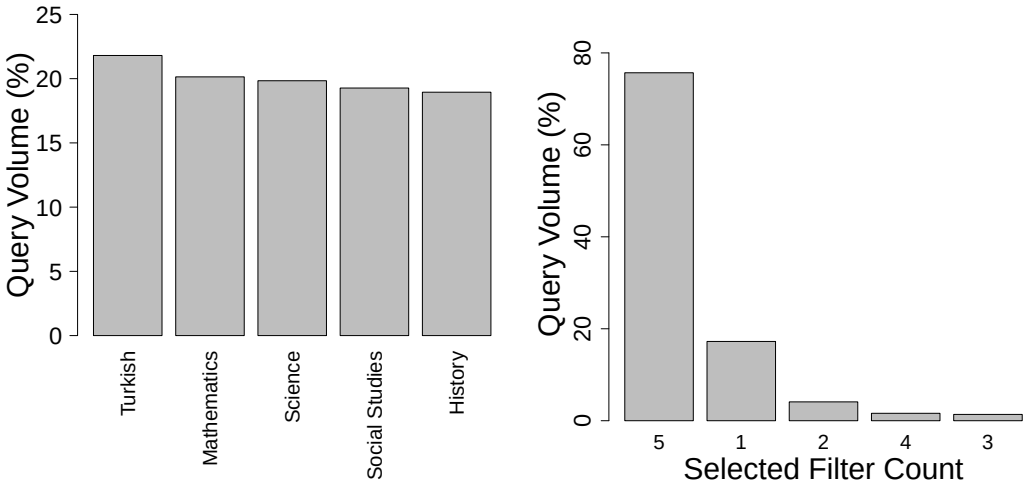

Figure 3.5: Distribution of course filters used in queries.

In contrast, the grade filter, at a first look, seems to be used more effectively as the majority (more than 70%) of the searches are restricted to a certain grade level (Figure 3.4); grades 5, 6 and 7 being the most popular ones. However, this difference in the behavior may not necessarily be caused by the students' awareness of this filter, as the search GUI for the trial accounts, by default, shows only the user's own grade level as selected. Therefore, for most of the searches, we can still claim that students are reluctant to change the default filter settings. This is an interesting finding that deserves further analysis, as it can provide useful insight for designing a better search interface.

Table 3.3: Session characteristics

Number of sessions	35,225	
Number of sessions having single query	20,914	59%
Avg. num. of queries in all sessions	1.74	
Avg. num. of queries in sessions with >1 query	1.86	
Longest session duration	133 min	
Avg. duration in all sessions	4.7 min	
Avg. duration in sessions with >1 query	7.1 min	

3.1.2 Session Characteristics

As in the previous studies [29], we detect sessions by grouping together a particular user's successive searches that has a time gap less than a time-out value (30 minutes). Table 3.3 presents several statistics about query sessions. Among the total of 35K sessions, about 59% include only one query.

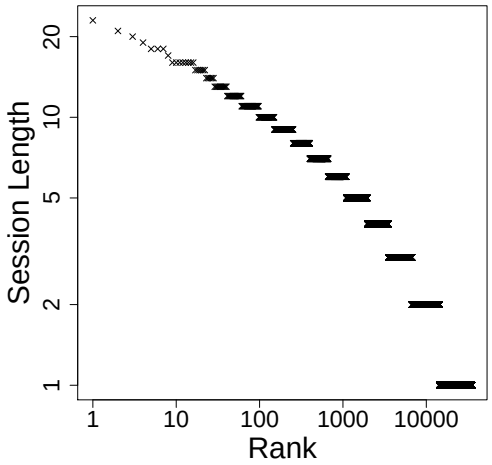

Figure 3.6: Distribution of session lengths. The x-axis represents the rank according to the session length in query count in the plot.

This skewed distribution of session length in number of queries can be seen in Figure 3.6. Users submit around two queries in a session on average (computed by macro-averaging over users). The average number of queries submitted to a commercial search engine is 2.4 [34]. The average session duration in our log is 4.7 minutes and this is slightly longer than the session duration for children (between ages 6-18) reported in [29]. However when it is compared to a general user's query

session in a web search engine (around 7 minutes in [34]), it is shorter. This again indicates that the students can effectively find what they look for in this context of educational search.

3.1.3 User Characteristics

We present the characteristics of users in Table 3.4. Among 18K total users, 40% of them issue only one query during the one month period of our log. This skewed distribution can also be seen in Figure 3.7 (left plot), where a large portion of users asks very few queries but a few users submit large number of queries. The distribution of the number of sessions over users shown in Figure 3.7 (right plot) is even more skewed since 60% of users interact in only one session. On the average, users ask 3.61 queries in 1.92 sessions.

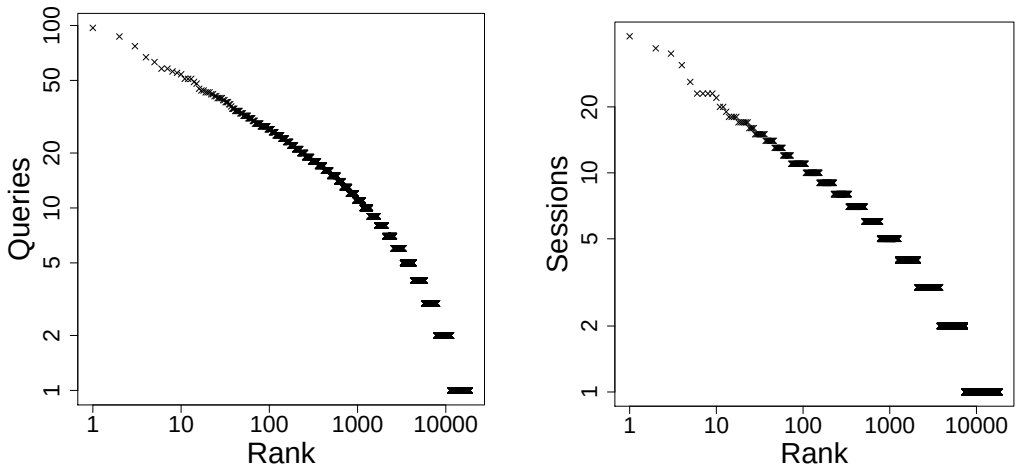

Figure 3.7: Distribution of number of queries (left plot) and sessions (right plot) over users. The x-axis represents the rank according to the number of queries (sessions) per user in the left (right) plot, respectively.

Figure 3.8 shows the distribution of query submissions over time. Monthly analysis (left plot) shows weekly patterns clearly. Students submit the largest number of queries on Sunday and least number of queries on Friday, according to the daily analysis in Figure 3.8 (center). This provides some interesting clues in students' studying habits: the students heavily search for information on Sunday,

Table 3.4: User characteristics

Number of users	18,534	
Number of users with >1 query	11,402	62%
Number of users with >1 session	7,590	40%
Avg. num. of queries per user	3.61	
Avg. num. of queries per user with >1 query	5.24	
Avg. num. of sessions per user	1.92	
Avg. num. of sessions per user with >1 query	3.31	

during when they might be doing homeworks for the upcoming week. Then, their activity in the search engine decreases gradually in the weekdays and reach the minimum on Friday, when most of the students seem to enjoy the weekend. Hourly analysis in Figure 3.8 (right) shows the percentage of queries submitted to the system in different hours of a day separately for weekdays and weekends. It is seen that students prefer to use the system mostly between 18:00-21:00 on weekdays (after school) and between 12:00-21:00 on weekends.

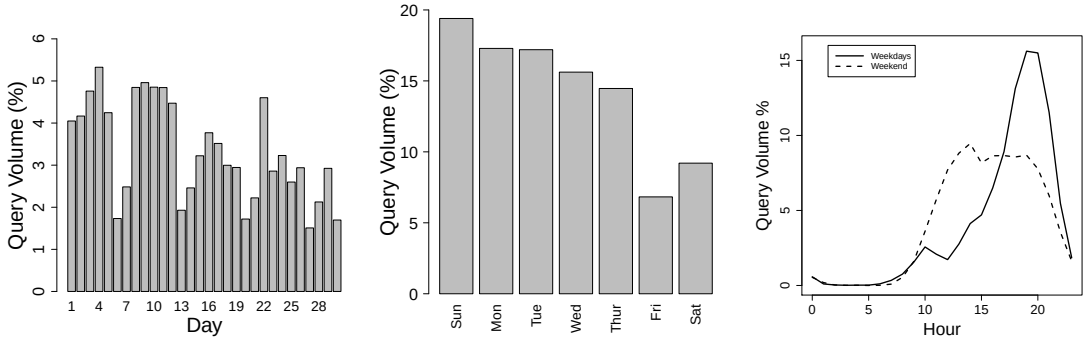

Figure 3.8: Distribution of query submissions over time. Left: Number of query submissions per day in December 2013. Center: Distribution of queries over weekdays. Right: Percentage of queries submitted per hour of the weekdays and weekend days.

3.1.4 Out Click Characteristics

In this part, we analyze the clicks on the query results. Table 3.5 presents basic statistics about clicks. On the average, users click 2.56 results per query and 2.29 clicks are unique. In terms of a session scope, there are 5.33 clicks on the average and 4.80 clicks are unique. The log-log scale plots in Figure 3.9 shows that the

Table 3.5: Out click characteristics

Total number of clicks	155,537
Average number of clicks per query	2.56
Average number of clicks per session	5.33

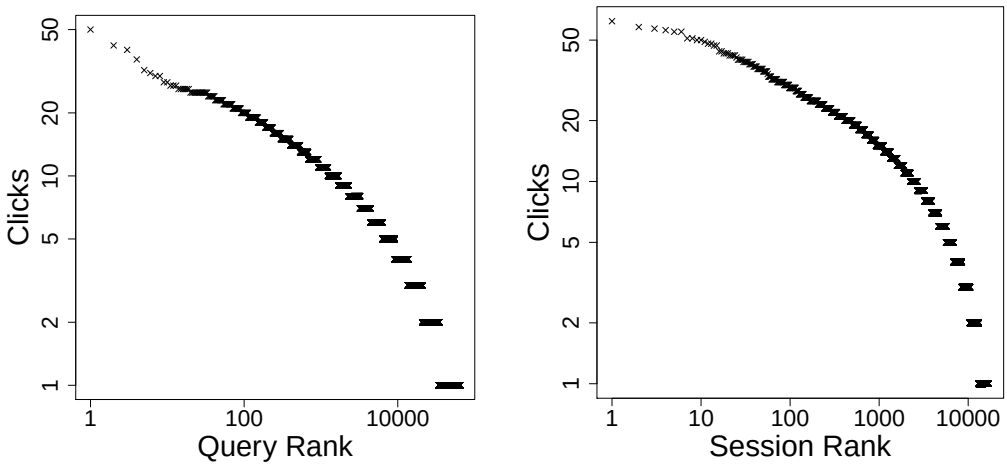

Figure 3.9: Distribution of click counts per query (left plot) and per session (right).

distribution of number of clicks again follows a power law distribution.

Figure 3.10 shows the percentage of clicks for each type of learning objects. It is seen that users mostly prefer "animation" and "interactive exercise" type of contents. Furthermore, "interactive activity" and "lecture" type of contents are also clicked frequently, while textual resources ("Text") are less likely to be clicked. These findings reflect the students preference of interactive content over purely textual material, which actually leads most educational content to be presented in the former format in Vitamin.

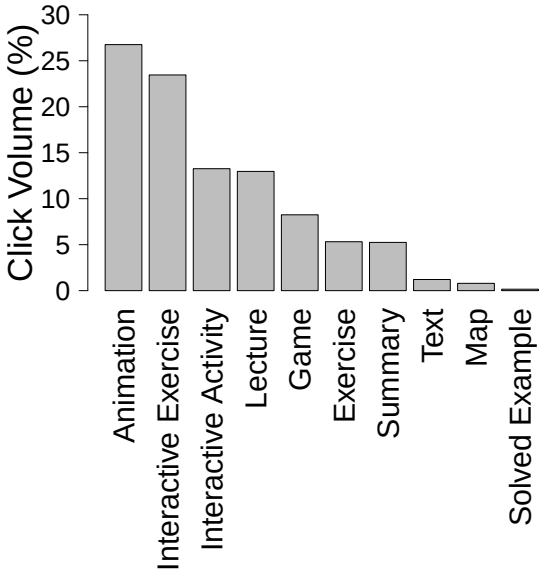

Figure 3.10: Distribution of result clicks by content type.

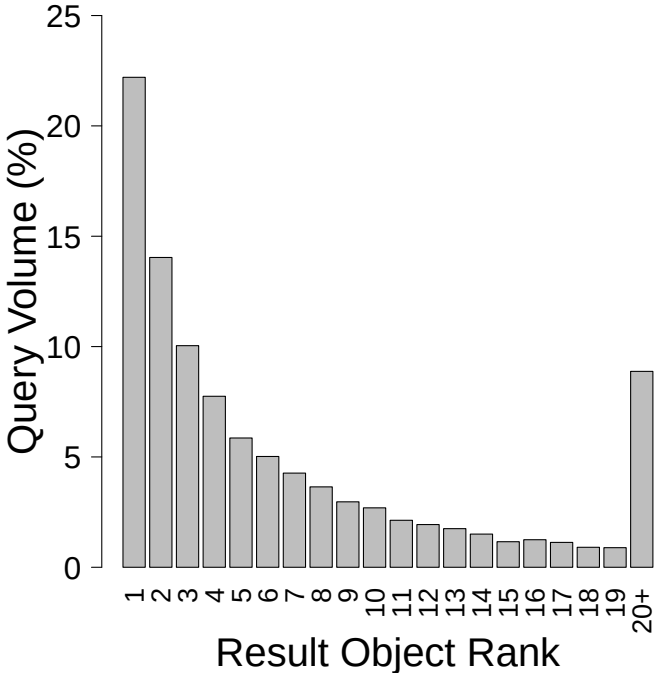

Figure 3.11: Distribution of clicks by rank.

Finally, we focus on ranks of clicked results in Figure 3.11. We see that while top-2 results, non-surprisingly, take the largest share of the clicks, there is a non-negligible fraction of clicks for the results placed at much lower ranks, even after

rank 20. According to a general web search engine log [35], clicks for top-2 results account for 58% of all clicks and only 9% of clicks are below rank 10. However, in our log, top-2 clicks and clicks after rank 10 constitute 36% and 20% of all clicks, respectively. This might either indicate the students' dissatisfaction of the results, or their preference to see several relevant results while learning a topic.

3.1.5 Findings

In this section, we presented an in-depth analysis of a query log from a popular K-12 educational search system with real user queries. Our analysis revealed that the trends in this context differ from general web search in various aspects, which might be exploited for building educational search engines that are better tailored for students' needs and behaviors. In particular, the high fraction of repeated queries indicates that system components that rely on the query history (such as caching and query suggestion) can be made more effective. The students' preferences in using the query filters call for reconsidering the design of the search interface. Finally, our out-click analysis shows that students prefer active content formats (like animations and interactive lectures) over the static content (like text) and can click further lower ranks in the results list other than the first few results. Such findings can help designing better features for the machine-learned ranking algorithms and lead higher user satisfaction.

3.2 Refinding Analysis

In literature, query log analysis has been done on various query logs provided by some commercial search engines or by some vertical search engines. There exist few query log data which are distributed as public data to help researchers to exploit possible works to be done in search problems. One of the issues that arise from search problems is refinding problem. *Refinding* is the behavior of a user that searches particular document and as a result clicks the same document multiple times from different search sessions. In the work [9], it is noted that 17%

of those surveyed people complained about finding the page once they visited and stated that this is one of the biggest problems in using Web.

Using the query log data provided by SEBIT, we tried to explore refinding behavior of students who issue searches. The data is comparable to other statistical datasets used in query log analysis, which is already explained in the previous section in detail. Query log data gives helpful feedback about user actions, yet it is not enough to explain the underlying motivation behind user search. *Refinding* can be considered as one of the motivations users may have during search.

We tried to capture *Refinding* behavior by first grouping user issues according to their unique userids derived from query log. Then for each user, search issues are sorted considering their timestamp provided in data. We assumed that if a user clicks for a document and later on in a different search session clicks again that particular document, this behavior is labeled as refinding. Note that, to have *Refinding* behavior, one has to click for a particular document from two different search sessions having different timestamps. Also note that, even if those two search sessions do not have the same query text as an input to the search engine, we considered this as *Refinding* behavior as well. What is important for us is to find out whether the same document is clicked multiple times for a particular user in order to label as *Refinding*.

For instance, a user searches for the query *"Math functions"* and clicks a learning object (page in Web) shown in the result list for that query. Later the same user searches *"functions"* at a different time and clicks the same document again, which we assumed has underlying motivation to find the same document clicked once before. Therefore we consider it *Refinding*.

In this section, we show the results of *Refinding* behavior we observed in our query log data. While doing that, we choose the particular observations stated in [11]. For each analysis we make, we first state the case for general web and provide our results to compare with theirs.

We categorized queries as either *Refinding* or *Newfinding* queries as stated in

[11]. *Refinding* query is the query session in which user clicked a document that is clicked before on another search session by again that particular user. *Newfinding* query is the opposite, having no documents clicked before on any other search session user has. The results along with their comparison to their corresponding results in Web are as follows:

1. In Web, in 40% of all search sessions users have in general web search engines, there is at least a document retrieved in the result list and clicked by the user who clicked that particular document at least one time in any of the past searches. In other words, 40% of users have tendency and motivation to find and therefore click the document they once visited.

   ⇒ In our findings, the results are significantly different. In our query log data, we have 66908 query issues in total. Of those, we found *Refinding* behavior in 17218 queries, which is approximately 25%.

   The reason could be that in educational search environment, students periodically study different materials related to the subjects listed in curriculum. Therefore, they are likely to search for another subject different than what they already searched. Another reason could be that students advertently try to avoid those documents once they already discovered. In terms of learning aspect, it might be rational and wise to choose a document they have never visited to explore what that particular material can offer for them to learn the subject better.

2. If we are to repeat the previous analysis in terms of document perspective, the results again show differences. In other words, among all documents clicked by all users, 28% of them are clicked multiple times by the same user at different search sessions having different timestamps.

   ⇒ In our query log data, we have 165,587 learning objects (documents) clicked at least one time by a user in any search session. Of those, only 20,594 documents are clicked multiple times by the same user at a different time, which roughly corresponds to only 12% of all documents.

   The reasoning behind this behavior is the same as the previous one, that

is, students are less likely to search the same subject due to having multiple subjects listed in their course curriculum and students tend to look for new learning objects to enhance their knowledge about subjects covering different parts mentioned in different documents.

3. Another difference is observed in the number of documents clicked by multiple users in any search sessions. Among all clicked documents, in web only 7% of documents are clicked by multiple users. However, this behavior must be handled regarding the immense number of documents across web.

   ⇒ The results for our query log indicate completely different story. Among all documents clicked, 99% of them are also clicked by another user in a different search session. This is the reflection of our learning object set, which includes only 3500 different objects to be retrieved in result lists. Hence, students are shown similar document lists even if queries written might be different. However, considering the amount of query sessions (66K) we have in our query log, it is not reasonable to explain this behavior only considering the amount of documents we have. Since students share same documents in terms of click information, which can indicate students are likely to find similar documents relevant compared to general web users.

4. Another statistic is to find out the tendency users have to prefer documents that they have never visited before, specifically in search sessions they explore documents that they visited once. In other words, among all *Refinding* queries in Web, 14% of them include user clicks on new documents that are not clicked before by that particular user.

   ⇒ The above analysis given in [11] is one of the behaviors where results show significant difference compared to our results. In our query log, students are more likely to click new documents in search sessions in which they already clicked on a document they once visited. In *Refinding* query sessions, it is observed that 43% students also click new documents that they never visited before.

   We believe that there are two specific reasons behind this behavior from

students according to our observations. The first one is willing of the students to explore new learning objects while trying to cover more aspects of a particular subject. Another reason is more general, that is, as mentioned in previous section, students are more likely to click documents on average compared to general web searches, which essentially results in newly clicked documents.

5. Another aspect of the analysis is to find out the correlation between query texts that resulted in *Refinding* behavior. In web, within the user issues, users generally prefer the same query text while trying to visit a document that they visited once. Among all query sessions including *Refinding* behavior, 71% of them come from the same query texts. The calculation is made by simply taking the ratio of unique query texts over all query issues having *Refinding* click.

   ⇒ The behavior for this particular statistic is similar for the students as well. In our query log, students generally prefer the same query texts that result in *Refinding* click, which accounts for 64% of all query issues having *Refinding* behavior.

6. Similar to the previous analysis, users tend to click documents that they once visited in search sessions where they prefer the same query texts, which account for 87% of all query issues that have the same query text by the same user.

   ⇒ Unlike the previous analysis, there is a difference between Web and our query log in terms of this particular behavior of users. The probability of a search session in which *Refinding* happens to have the same query text as the ones where the users clicked the same document before, is considerably high. However, the probability of a student to click on a document that he visited once before in search sessions where he prefers the same query text is only 44 %, which is relatively low compared to Web.

   The main reason for this result could be that students have less skills to define their queries in terms of their search needs compared to general web users. They have difficulties to express their intent in clear manner,

which results in different query texts even though they have the same intent.

7. Same query text analysis is done this time for search sessions, namely *Newfinding Query Sessions*, where a document that the searching user never visited before gets clicked. As suggested in the paper [11], the results are lower than the *Refinding Query Sessions*, that is, among queries having the same text, 38% of them result in newly clicked documents.

   ⇒ Naturally considering the previous analysis results, this percentage is much higher than Web. While learning, new documents mean more source to cover for a subject, which results in more query sessions where *Newfinding* behavior happens. However, it must be noted that this *Newfinding* behavior occurs even in sessions having the same query texts, meaning the students prefer new learning objects in a consistent manner. Among all queries having the same text, 74% of them result in newly clicked documents.

8. In addition to the previous two analysis, the results for intersection behavior of these two are also calculated. In Web, among query sessions having the same query text, probability of a search session to have both *Refinding* and *Newfinding* behavior is 25%.

   ⇒ The results for this analysis show similarities to those in Web. Yet, the ratio is slightly lower than Web, which is 18%. This difference can be explained by the lack of ability of students expressing their intents clearly. In other words, the ratio clearly shows that even though students type the same query text, those sessions might intend to find both *Refinding* and *Newfinding* documents, which is what makes it difficult for the search engine to predict actual user intent to improve search results.

9. Another analysis made on the paper [11]is whether *Refinding* behavior depends on the number of clicks user made on that particular refinding session. Among all user issues made, 29% of the search sessions which result in single click on a document have *Refinding* behavior.

⇒ The results for this analysis on our query log data are slightly different than the results with the Web. Students are more likely to click on Refinding document where they prefer to click a single document on a search session, which accounts for 42% of all searches including single click. The difference arises from the search habits students have compared to general Web users.

In web, generally search sessions having single click are considered as failed searches. Yet, this is different for students. Having a failed search session including single click generally leads the user to reformulate his query to perform better search to find the page he is looking for in Web. However, students lack the ability of reformulating queries that they think are unsuccessful. In other words, ratio of single click sessions to all search sessions is less than the ratio we have in general Web case. Therefore, in our query log, we believe that, contrary to the behavior users have in Web, sessions having single clicked document means successful *Refinding* search rather than an unsuccessful one.

10. The last analysis we performed for comparison is the opposite of the previous analysis, which is to find *Refinding* behavior in search sessions where multiple clicks occur. In web, among all search sessions including multiple clicks on documents, only 5.3% of them include *Refinding* behavior, which is expected regarding the previous analysis.

    ⇒ This is one of the analysis that behave significantly different for our query log data compared to general Web. 57% of search sessions including multiple clicks lead students to click on a document they once visited.

    The reasons for this behavior we believe are twofold. The first reason is the less number of documents our data have. Hence students tend to click more documents compared to Web, which eventually includes one of the documents they already visited before. The other reason would be that students are less likely to remember the documents they visited before in a successful search, therefore for a *Refinding* intent, they have to click more documents than general Web users.

Chapter 4

LETOR - Learning to Rank

In information retrieval (IR) and natural language processing (NLP), ranking is the central problem for many tasks. These tasks include document retrieval, question answering, personalized search, collaborative filtering, document summarization, and so on.

Ranking problem generally consists of two different types; which are ranking creation [18] and ranking aggregation. Ranking aggregation is to create a list of objects using multiple lists of objects by aggregating them into a single list, while ranking creation is to make a list of objects using feature sets of the objects given another type of the object (query in web search case). Our work in this thesis falls into the latter group.

Document retrieval is one of the main problems in information retrieval for which the ranking problem is the main issue to be solved. In web, although there are limitations for search, to access information available on Web it is by far the most common and practical solution to search for a page. For instance, according to a report by IProspect, 56% of the internet users use web search every day and 88% of the internet users use web search every week [9].

In this scope of document retrieval task, learning to rank refers using machine learning algorithms to rank documents using a trained model given query and

document pairs. The idea of ranking creation is to create a list of documents using extracted features of documents given query so that good and preferable documents will be ranked at top compared to other documents. Learning to rank is therefore concerned with the automatic creation of ranked list of documents using machine learning algorithms.

Learning to rank plays a significant role in document retrieval to rank documents using machine learning techniques to come up with a better list of documents to satisfy user needs and therefore to improve search engine performance. However, this technique can be considered as a new trend following the literature in information retrieval. Before LETOR, traditionally search engines were trying to find a function $f(q, d)$ to score a document d given query q without learning any model using machine learning algorithms.

In [15], BM25 technique is used to derive a conditional probability to construct ranking function $f(q, d)$ where conditional probability is represented as $P(l|q, d)$ in which l denotes the label of the document given query to get either 1 or 0 being relevant or irrelevant respectively to the query.

Another technique used for document retrieval task in literature prior to LETOR is using Language Model for IR (LMIR) [16]. Using LMIR, the ranking model is again defined as conditional probability distribution $P(q|d)$ where q represents the query and d denotes a document. In both techniques, ranking is made using probability distributions that is probability of each document given query is calculated and then documents are sorted according to their probability scores to assess final ranking of documents. Since these two techniques use only probability distributions, no model is learned in either of the methods.

Learning to rank method is a new trend arising in information retrieval to be used in document retrieval such as general web search. In learning to rank, the ranking model $f(q, d)$ is learned through past data, which is basically the search issues by users, called query log data. Using query log data, user behaviors for particular queries such as impressions and clicks are recorded to be exploited in the model to train better ranking function. In learning to rank environment,

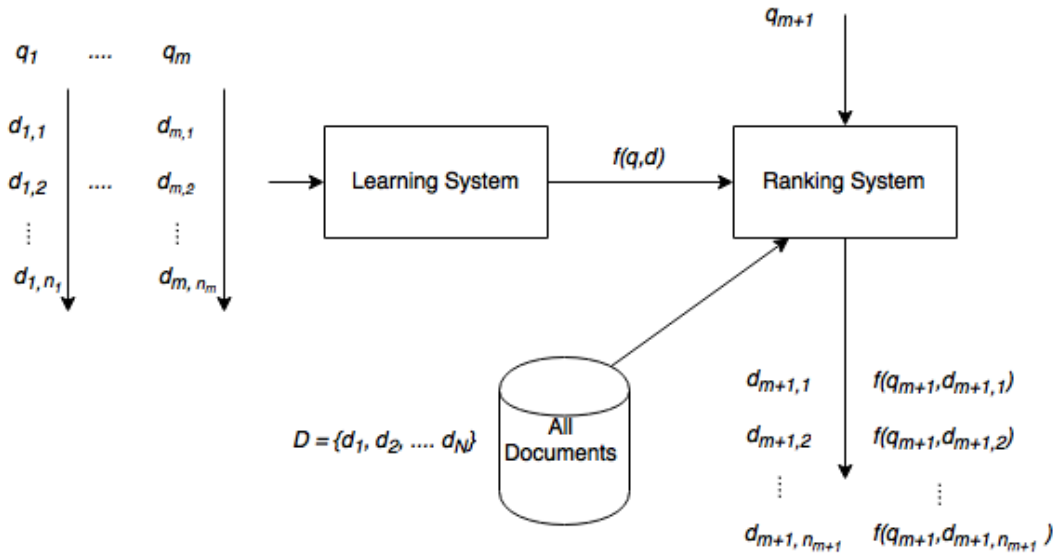

Figure 4.1: Learning to Rank for Document Retrieval

sets of queries $\mathcal{Q}$ and documents $\mathcal{D}$ are constructed. Then for each query q in $\mathcal{Q}$, associated documents are chosen. Learning to rank uses supervised machine learning algorithms, that is, each document d in associated set of q is labeled to denote its relevance to query q. Then using these query-document pairs along with their correspondence labels, ranking function $f(q, d)$ is learned to predict scores. Overall structure of working mechanism of general learning to rank environment can be seen in Figure 4.1. The figure is inspired from the work in [18];

4.1 Training and Testing

Since learning to rank is a supervised learning process, creation of training and test sets is a must for the setup. In this section, we will be presenting the idea of learning to rank environment setup briefly. As explained in [18], each notation that will be used in the explanation is given with their explanations in Table 4.1.

Using query log data, we first find the unique query issues and unique documents to create query set Q and document set D respectively. In learning to rank environment, feature vectors are derived from query-document pairs. Hence, after

Table 4.1: Summary of Notations

Notations	**Explanations**
$\mathcal{Q}$	query set
$\mathcal{D}$	document set
$\mathcal{Y} = \{1, 2, ..., l\}$	label set with order $\succ$
$q_i \in \mathcal{Q}$	i-th query in query set
$D_i = \{d_{i,1}, d_{i,2}, ..., d_{i,n_i}\}$	set of documents related to q_i in training data
$d_{i,j}$	j-th document in D_i
$Y_i = \{y_{i,1}, y_{i,2}, ..., y_{i,n_i}\}$	set of labels of documents in D_i with respect to q_i
$y_{i,j}$	label of the j-th document in D_i with respect to q_i
$x_{i,j} = \varphi(q_i, d_{i,j})$	feature vector derived from $(q_i, d_{i,j})$ pair
$X_i = \vartheta(q_i, D_i)$	feature vector derived from (q_i, D_i) pairs
τ_i	ranking list of the i-th query
$\tau_i(j)$	rank of the j-th document in D_i with respect to q_i
$\mathcal{R} = \{(q_i, D_i), Y_i\}_{i=1}^m$	original training set
$\mathcal{R}' = \{X_i, Y_i\}_{i=1}^m$	transformed training set
$\mathcal{S} = \{(q_{m+1}, D_{m+1}), Y_{m+1}\}$	original test set
$\mathcal{S}' = \{X_{m+1}\}$	transformed test set

finding overall sets, each query is associated with a number of documents. Along with the association, relevance of documents with respect to associated queries is also given. The relevance information may vary depending on the method to annotate the query-document pairs. In this section, the most common scenario is taken into consideration, that is, relevance labels given as integers starting from 1 to g. The labels are at several grades. Specifically, a higher grade a document has with respect to a query, the more relevant the document is associated with the query.

In this section, we follow the work in [18] by Li to explain basic principles of learning to rank environments. Suppose that $\mathcal{Y} = \{1, 2, ...g\}$ is the label set, where the labels represent grades each document can get with respect to a particular query. There exists a relation between the grades in $\mathcal{Y}$ such that, $g \succ g - 1 \succ ... \succ 1$, where $\succ$ states the order of the relation between grades. Further assume that we have training data consisting of query set $\{q_1, q_2, ..., q_m\}$ where m denotes the number of queries that are used in learning. In the set of queries used in learning, i-th query is called as q_i. Then $D_i = \{d_{i,1}, d_{i,2}, ..., d_{i,n_i}\}$

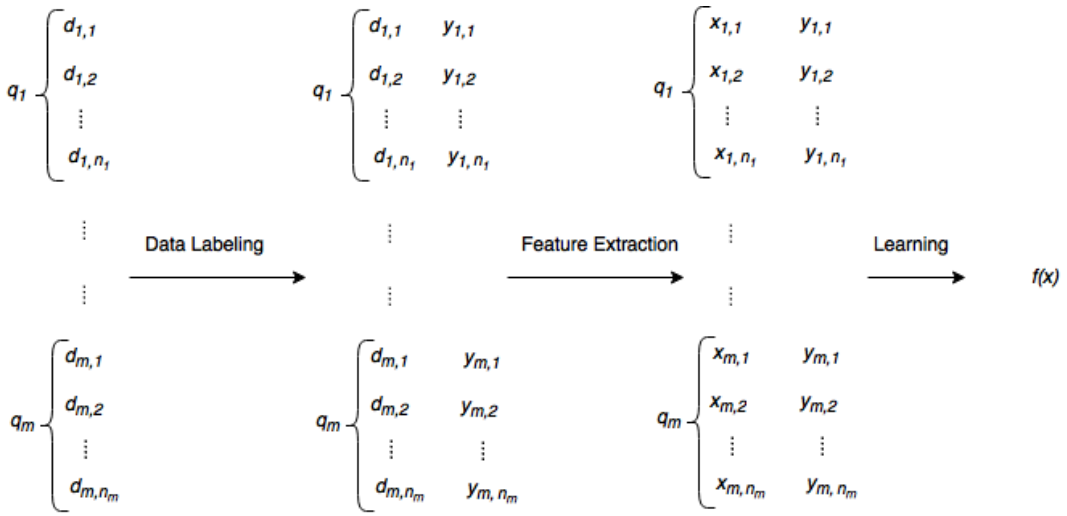

Figure 4.2: Learning to Rank Training Phase

is the set of documents associated with q_i. Along with documents themselves, $Y_i = \{y_{i,1}, y_{i,2}, ..., y_{i,n_i}\}$ is the set of labels of documents in D_i with respect to q_i. Therefore, $y_{i,j}$ represents the label of the j-th document in D_i with respect to i-th query q_i. Overall, training set before feature extraction becomes $\mathcal{R} = \{(q_i, D_i), Y_i\}_{i=1}^{m}$, where m denotes again the number of queries used in training set to learn the model.

Further, the feature vector for each query-document pair is extracted from training set $\mathcal{R}$. For $i = 1, 2, ..., m$ and $j = 1, 2, ..., n_i$, the feature vector $x_{i,j} = \varphi(q_i, d_{i,j})$ is constructed using query q_i and document $d_{i,j}$ where φ represents the functions used to derive the features. In other words, φ is the function which gets two inputs from query q_i and document $d_{i,j}$ and outputs the feature vector $x_{i,j}$. Feature extraction process is done for each query-document pair in each list of documents associated to a particular query, that is, the feature vector $X_i = \vartheta(q_i, D_i)$ for a particular query q_i is calculated using D_i. Next, transformed training data $\mathcal{R}' = \{X_i, Y_i\}_{i=1}^{m}$ is constructed using each X_i along with their correspondence label set Y_i for associated documents.

Suppose that we define the ranking of the result list of documents associated with query q_i as τ_i. Ranking documents is simply nothing but assigning scores to each document to further use them in sorting to finalize their ranking position

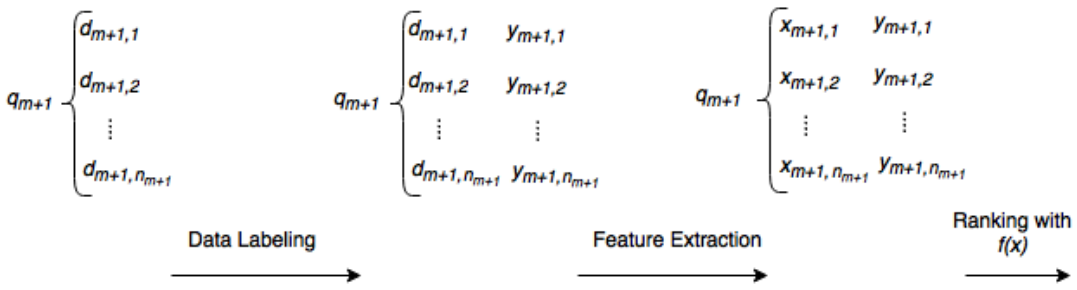

Figure 4.3: Learning to Rank Test Phase

in the result list. Hence as a result, learning to rank algorithms try to find the ranking model $F(q_i, D_i)$, which can give the possible best ranking list τ_i for each query q_i using the transformed training set $\mathcal{R}' = \{X_i, Y_i\}_{i=1}^{n}$. The construction of the training data set is illustrated in Figure 4.2.

Similar to the training phase, the test data is prepared using the same feature function φ used in the learning phase. Suppose a new query q_{m+1} comes into the system as a test query. We are also given the set of documents $D_{m+1} = \{d_{m+1,1}, d_{m+1,2}, ..., d_{m+1,n_{m+1}}\}$ associated with the query q_{m+1}. In test phase, we are not interested in the set of labels of the documents in D_{m+1}, since the learned model is there to predict those grades. Afterwards, the feature vector X_{m+1} that is calculated using φ learned in the learning phase is given as the test data to the learned model $f(x)$. The model $f(x)$ assigns scores to each document in D_{m+1}. Next, the grade (or score) set $Y_{m+1} = \{y_{m+1,1}, y_{m+1,2}, ..., y_{m+1,n_{m+1}}\}$ of test data is sorted to construct τ_{m+1}. The test phase of the learning to rank environment is illustrated in Figure 4.3.

The training and testing data splits are similar to, but slightly different from, the traditional data splits in supervised learning environments to be used in machine learning algorithms such as classification and regression. In learning to rank setup, although each instance is represented as query-document pair, documents associated with the same query form the groups. The groups within the whole training data is independent and identically distributed from each other, whereas the instances within each group are not independant and identically distributed data, which results in the difference between LETOR environment and

conventional machine learning algorithms.

4.2 Annotation

Learning for ranking is a supervised learning process that needs a high quality training data set to perform better. Creation of the training data is not an easy task due to several reasons, one of which is to have grade (or label) for each query-document pair put as an instance to the model. In the literature, there are two ways to annotate the documents associated to a particular query. Both have advantages and drawbacks depending on the environment. In this section, we mention the ways to grade labels to the documents.

The first and the most common way to annotate the documents is *human labeling*. In this approach, a set of queries from the query log is chosen randomly. Next, exploring the query log data, the set of documents related with the queries chosen is determined. As a result, we end up with having a query set along with their correspondence document list to annotate. Then human judges are asked to grade the documents with respect to the given query. Generally, relevance judgments are conducted using multiple levels. Normally, each query-document pair is given to a single judge to label the documents with respect to the query. However, the labeling on query-document pairs can be conducted by multiple judges, and then majority voting can be carried out to have a final score. The advantage of the *human labeling* approach with multiple judges is its robustness due to the voting technique. Yet, having multiple judges to label documents is an expensive process, which leads researches to come up with different techniques to annotate the documents.

The other approach is deriving labels implicitly from click-through data. Click-through data at a web search engine include implicit feedbacks of users as impression and click, which further can be used to assign relevance labels to the documents. For this approach, there are mainly two techniques introduced in the literature that are used to predict relevance information of the documents, which

are *Position Model* and *Cascade Model*. There are various click models built on top of these models, which can be seen in [23, 24, 25].

The two models differ from each other by the assumptions they take while finding the relevant documents. A position model [22, 21, 36] assumes that in order for a click to be considered as relevant, the document clicked must be examined and found out to be relevant by the user. In the position model, each rank in the result list has certain probability of being examined and it decreases as ranks go higher. The model has a drawback in the sense that it treats documents clicked individually, therefore it fails to capture the relation between documents in terms of relevance.

The cascade model [22] assumes that user sequentially examines the documents in the result list and the document user clicked last is the relevant document in the given search session. In other words, the probability of a document being relevant depends on the relevance of the previous documents having lower ranks. However, it is straightforward that cascade model fails in search session where there exist zero or multiple clicks.

The annotation of the query log data using click-through information has pros and cons. Derivation of relevance information from click-through data is of low cost compared to human labeling and it also reflects the real user choices which make them more reliable. Nevertheless, generally click-through data is noisy and it cannot be useful for singleton queries (low frequency queries).

4.3 Feature Extraction

The ranking model $f(q, d)$ is learned through query-document pairs by using the calculated features. In other words, there is a feature vector $f(x)$ based on q and d. Therefore, the performance of the ranking model highly depends on the feature vector. Thus, it is crucial to first find out which features will be used. In web search, there are numerous features provided in the context of learning to

rank algorithms. In this section, we will give information about two of the most popular ones, namely BM25 and PageRank. Most common features used in the learning to rank setups can be found in [13, 37].

4.3.1 BM25

BM25 is a probabilistic model representing the relevance of document d to query q in terms of textual perspective [15]. It tries to capture how many terms appear both in the document and in the query. Specifically BM25 of a query-document pair is calculated as follows

$$BM25(q,d) = \sum_{w \in q \cap d} idf(w) \frac{(k+1)tf(w)}{tf(w) + k((1-b) + b\frac{dl}{avgdl})} \tag{4.1}$$

where w denotes a word appearing both in the document d and in the query q, $tf(w)$ denotes the frequency of w in d, $idf(w)$ denotes the inverse document frequency of w, dl denotes the document length of d, $avgdll$ denotes the average document length and k and b are parameters to be tuned [18].

4.3.2 Page Rank

PageRank demonstrates the importance of a web page in terms of ingoing and outgoing links it has [17]. In order to calculate PageRank score for a particular web page, first a directed graph is constructed to represent the web where vertices represent pages and edges represent hyperlinks in between pages. Page rank of a web page d is defined as $P(d)$

$$P(d) = \alpha \sum_{d_i \in M(d)} \frac{P(d_i)}{L(d_i)} + (1-\alpha)\frac{1}{N} \tag{4.2}$$

where P(d) is the probability of visiting page d, $P(d_i)$ is the probability of visiting page d_i, $M(d)$ is the set of pages linked to d, $L(d_i)$ is the number of outlinks from d_i, N is the total number of nodes on the graph, and α is a weight [18].

Table 4.2: Categorization of Algorithms Used in LETOR

	SVM	**Boosting**	**Neural Net**	**Others**
Pointwise	OC SVM [38]	McRank [39]		Prank [40] Subset Ranking [41]
Pairwise	Ranking SVM [42] IR SVM [43]	RankBoost [44] GBRank [45] LambdaMART [46]	RankNet [47] Frank [48] LambdaRank [49]	
Listwise	SVM [50] PermuRank [51]	AdaRank [52]	ListNet [53] ListMLE [54]	SoftRank [55] AppRank [56]

4.4 Learning Approaches

As we described earlier, learning to rank creation is a supervised learning process, in which machine learning algorithms are used. However, the difference in the machine learning algorithms used for learning to rank creation is that instances are sorted after assigning scores to each of them. As other topics, algorithms that can be used in learning to rank environments are intensively studied recently. There are three approaches commonly used in learning to rank environments, which are pointwise, pairwise and listwise approaches. In [18], Li summarizes the most used algorithms in learning to rank creation setups regarding the category they fall into, which can be seen in Table 4.2. The approaches differ from each other in terms of creation of the instances to be used in learning.

4.4.1 Pointwise Approach

In pointwise algorithms, the group structure of the instances that belong to some particular query is ignored. In other words, the feature vectors created for each query-document pair are then combined into a single set in which there is no further information about the group information a particular instance belongs to.

Machine learning algorithms for classification, regression and ordinal classification can be used to predict the grade of the instance in the test set of pointwise algorithms. Then instances are sorted according to scores assigned to them by the model to find out final result lists. The loss function in the learning is pointwise in the sense that it only utilizes single instances without taking group information into the consideration.

4.4.2 Pairwise Approach

In the pairwise approach, the ranking problem is transformed into pairwise classification or pairwise regression [18]. In this approach, group structure of the instances is ignored as well, yet the pairwise approach captures the relation between documents because of its nature of considering preferences between instances rather than single instances.

For $i = \{1, 2, ..., m\}$, the algorithms used in the pairwise approach try to find the preference between each pair of documents. In this approach, instances are indeed pair of documents. For instance, if $x_{i,j}$ has a higher grade than $x_{i,k}$ ($y_{i,j} \succ y_{i,k}$), then $x_{i,j}$ over $x_{i,k}$ becomes a preference pair. The pairwise classification or regression algorithms in the pairwise approach then try to predict the preference of each pair of documents.

4.4.3 Listwise Approach

The listwise approach addresses the ranking problem in a more natural way, trying to capture the group information of the instances as well. The advantage of the keeping group information is to use gain functions particularly used in evaluation metrics to further have better learned models. In this approach each instance that belongs to a particular query q_i creates a unique group identifying the query q_i. Then algorithms try to learn a model that utilizes the order for each group to have better performance.

According to previous studies on evaluation of the algorithms that belong to each group, the listwise and the pointwise approaches generally perform better than the pointwise approach. However, there are certain learning to rank environments in which pointwise approach can perform better compared to the others, which is the nature of the machine learning algorithms. Specifically, LambdaMART algorithm achieved best performance in the Yahoo Learning to Rank Challenge [57].

4.5 Evaluation

In order to evaluate the performance of models learned by learning to rank algorithms, one has to compare the results obtained from the model with the result lists given as ground truth. In information retrieval there are multiple evaluation methods that can be used for performance evaluation. In this section, we will mention two of the most common techniques used in the literature, which are namely NDCG and ERR. Note that, evaluation metrics depend on the type of annotation done for relevance grade of the documents per query. The methods that will be further explained in this section are generally used for multi scale grade levels.

4.5.1 Normalized Discounted Cumulative Gain - NDCG

Discounted Cumulative Gain (DCG) is the first evaluation method that can be used to evaluate the result lists retrieved by the learned model having multi scale annotation scheme. Given query q_i and associated document set D_i, suppose that π_i is the ranking list (permutation) on D_i and y_i is the set of labels associated with each document in D_i. DCG measures the goodness of the ranking list with the labels [58]. DCG at position k for q_i is defined as follows:

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} G(j)D(\pi_i(j)) \tag{4.3}$$

In the equation 4.3, there are two parts, $G(.)$ and $D(.)$,in the calculation, which are gain function and position discount function respectively. Recall that, $\pi_i(j)$ denotes the position of the j-th document $d_{i,j}$ in π_i. $DCG(k)$ score is calculated by the summation of each score for each document whose position is not bigger than k. Therefore, DCG represents the cumulative gain from position one to position k with discounts on the positions. DCG alone cannot be used to compare the results with some particular baseline, since for DCG, the performance of the baseline is disregarded. Therefore, generally $NDCG$ metric is used, which is nothing but the normalized version of DCG metric. Calculation of $NDCG$ metric for query q_i is given as follows:

$$NDCG(k) = DCG_{max}^{-1}(k) \sum_{j:\pi_i(j)\leq k} G(j)D(\pi_i(j)) \tag{4.4}$$

Looking at Equation 4.4, we can easily say that $NDCG$ is the normalized version of DCG by the maximum DCG value that can be achieved for a particular query q_i. In other words, $NDCG$ gives 1 for a perfect ranking of π_i for a query q_i at position k.

The gain function $G(.)$ is generally defined as the logarithmic function on grade levels. The motivation behind the idea is to give more score to the ranking lists who achieve to predict documents having higher grade levels. Since it is more significant to predict well the relevance of the documents of highest grades for a model. Gain function for a document $d_{i,j}$ given query q_i is defined as,

$$G(i,j) = 2^{y_{i,j}} - 1 \tag{4.5}$$

where $y_{i,j}$ denotes the grade of the document $d_{i,j}$.

The position discount function $D(.)$ is defined similar to gain function $G(.)$ in the sense that it also uses logarithmic function on position rather than grade. With discount function, we simply try to give less importance to the documents having lower ranks in the result list. Given the result list π_i of the query q_i, the position discount function $D(\pi(i), j)$ for document $d_{i,j}$ is defined as,

$$D(\pi(i), j) = \frac{1}{\log_2(1 + \pi_i(j))} \tag{4.6}$$

where $\pi_i(j)$ represents the position of the document $d_{i,j}$ in the ranking list π_i. Hence, using Equations 4.5 and 4.6 DCG at position k becomes

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))} \tag{4.7}$$

If we add normalization factor into Equation 4.7, we get $NDCG$ score at position k, which is defined as follows:

$$NDCG(k) = DCG^{-1}_{max}(k) \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))} \tag{4.8}$$

$NDCG$ scores for each query q_i with $i = \{1, 2, ..., m\}$ are then further averaged to calculate final $NDCG$ score to represent the performance of the learned model.

4.5.2 Expected Reciprocal Rank - ERR

The other metric that can be used in evaluation of the learning models by learning to rank algorithms with multi graded levels is *Expected Reciprocal Rank ERR* [59]. This technique uses *cascade model* rather than *position model* compared to $NDCG$ metric, which is the most commonly used metric in learning to rank environments having multi level grades. ERR tries to give a better representative score for each result list compared to $NDCG$ by taking the relevance information of all the examined documents unlike $NDCG$, where the discount function only depends on relevance and position of the current document.

ERR metric uses the strong assumption made by the *cascade model*, which is that the probability of an examined document being relevant depends on the relevance information of the previously examined documents in the result list [60]. The key point in calculating the ERR metric is that probability of an examined document being relevant depends not only the relevance of the document but also relevance of all the previously examined documents. Similar to $NDCG$ metric, the probability function for a document to be relevant depends on the editorial grade made on the document.

Suppose that $y_{i,j}$ represents the editorial grade of the j-th document $d_{i,j}$ in π_i. ERR tries to find a mapping function between the editorial grade and the

probability of the relevance given the document. Given the query q_i and the document $d_{i,j}$ along with its corresponding grade $y_{i,j}$, the mapping scheme is as follows:

$$R(y) :\equiv R(y_{i,j}) \tag{4.9}$$

where R denotes the mapping function between the editorial grade and the probability of the grade of the document. $R(y)$ is defined similar to gain function in $NDCG$, where logarithmic function is used on the grades. Though, it has minor addition to the gain function used in $NDCG$, which is given below:

$$R(y) :\equiv \frac{2^{y_{i,j}} - 1}{2^{y_{max}}}, y \in \{1, 2, ..., y_{max}\} \tag{4.10}$$

ERR defines the discount function slightly different than the one in $NDCG$ metric. Simply, the discount function consists of two parts, which are called *utility function* and probability of previously examined documents being not relevant. Let $\varphi(j)$ be the utility function varying on the position of the document being examined. ERR metric defines the utility function as,

$$\varphi(j) = \frac{1}{j} \tag{4.11}$$

where j represents the position of the document examined. The second part of the discount function is to calculate the probability that the user finds the examined document relevant. Let the probability of the document $d_{i,j}$ being relevant be $P(j)$. Then, $P(j)$ can be defined as,

$$P(j) = \prod_{r=1}^{j-1}(1 - R_r)R_j \tag{4.12}$$

where R_r denotes the probability of the examined document $d_{i,r}$ in π_i being not relevant. R uses the mapping function defined above in Equation 4.10. R_j represents that j-th document $d_{i,j}$ in π_i is found relevant and calculated using mapping function R. Then, ERR score at position k becomes,

$$ERR(k) = \sum_{r=1}^{k} \frac{1}{r} P(r) \tag{4.13}$$

where $P(r)$ denotes the probability that the user stops at position r. Hence, combining Equations 4.12 and 4.13, ERR at position k is calculated as follows:

$$ERR(k) = \sum_{r=1}^{k} \frac{1}{r} \prod_{i=1}^{r-1}(1 - R_i)R_r \tag{4.14}$$

Then, ERR score of each query q_i with $i = \{1, 2, ..., m\}$ is then further averaged to calculate the final ERR score to represent the performance of the learned model. The important difference in ERR can be seen in Equation 4.13, which shows the discount function to be used to decrease the importance of documents having lower ranks. According to the ERR metric, the more relevant the previous documents are, the more discounted the other documents having lower rank are.

Chapter 5

Proposed Educational LETOR Model

In this chapter, we explain the model we use in our experiments with the query log data provided by SEBIT, company which has a search engine product named Vitamin. Our query log contains search issues made by the students who use Vitamin product, which is a commercial platform where thousands of educational materials exist. Therefore, our scope is the educational domain. While constructing the model, we try to exploit educational information included in the query log data. Our feature set includes attributes that represent educational information available in the data. Besides these features, we also include the most common features used in the learning to rank environments [13, 14, 15]. Apart from the feature set, we construct multiple models for each cluster which we derived from the query log. Specifically, we have three clusters that are frequency, course of query and grade of user based. We try to further improve search engine performance by introducing different models for each cluster to have a better average performance than the general model we construct. Finally, we propose a novel algorithm that tries to generate values for features having zero values specifically for queries having low frequencies (singleton queries, which are defined as the queries issued into the search engine exactly once).

5.1 Feature Set

To setup LETOR environment we first create the annotated data in which each query is associated with a list of documents having relevance scores labeled by us. Then, we created a feature vector to be used as instance in LETOR algorithms. We first analyzed the query log to explore potential features to be calculated for LETOR algorithms. According to our observations we created a feature vector for a query-document pair including 50 features consisting of both float numbers and boolean attributes.

While considering potential features, we added well accepted features in LETOR environments in literature, which include tf-idf and BM25 similarity features, query popularity, document popularity, query-document click count, etc. In addition to these features, we generated domain-specific features representing our data, i.e., course of the document, grade of the user issuing the query and type of document (animation, text, quiz etc.). In total, we represented query-document pair as a 50-dimensional feature vector.

In this section, we present the details of these features, by referring to their feature group and calculation issues. We would like to categorize our feature set in 5 different groups, which are as follows;

- Query-document text similarity
- Query specific
- Document specific
- Session based
- Query-document click based.

5.1.1 Query-Document Text Similarity Features

In our data, we have documents each of which has its own title and description part consisting of explanation of the document in text. Therefore, for this group we created two features representing the text similarity between query and document information. In literature, tf-idf and BM25 are the most common similarity metrics to be used in these tasks including LETOR. Hence, we have four features to represent the query-document text similarity. Before using these features, we need to normalize them. For each query session, calculated tf-idf and BM25 scores for each document are normalized into 0-1 range using linear normalization method. The final list of features for this group can be seen in the following list.

	Feature Group	Feature Type	Value Range
1.	Query-Document Text Similarity		
	1.1. tf-idf Title	float	0-1
	1.2. tf-idf Description	float	0-1
	1.3. BM25 Title	float	0-1
	1.4. BM25 Description	float	0-1

5.1.2 Query Specific Features

In the annotated data, we have 900 unique queries. Yet, for each unique query there might be multiple query issues, which are represented by different unique query identifier. We added two features namely query frequency and user frequency to calculate and capture the query popularity among our data.

In addition to these features, we added query text features, token length and character length, which are among the widely used features for LETOR environments. Then, we have unique related top document count and result count to reflect the number of documents originally associated with the given query. Result count is simply the number of documents retrieved by the search, whereas unique top related document is the union of documents which are retrieved in the first page result list. We normalized these features by using the linear normalization

method considering queries across all data.

The last features for this group are novel features we derived from our query log data. We observed that students tend to write queries which are appended by either grade of the user issuing query or the course of the subject that they are looking for. Therefore, we included these two boolean features into our feature set to represent that information.

	Feature Group		Feature Type	Value Range
2.	Query Specific			
	2.5.	Query Frequency	float	0-1
	2.6.	Unique Top Related Document Count	float	0-1
	2.7.	Result Count	float	0-1
	2.8.	Unique User Count Issuing Query	float	0-1
	2.9.	Token Length	float	0-1
	2.10.	Char Length	float	0-1
	2.11.	Having Course Name	boolean	0 or 1
	2.12.	Having Grade Name	boolean	0 or 1

5.1.3 Document Specific Features

Similar to query specific features, we have two features to represent popularity of the document to reflect the mostly used well known feature in general web search, page rank. These features are document frequency and user count and they correspond to the total number of times a document gets clicked and the number of unique users who clicked that document respectively. The values of these features are calculated without taking the query into consideration, and normalized across all data.

In addition to these popularity features, we have category type features to represent and capture the information our query log introduces originally. In

our data, we have information of document types, which in general are of three categories that are document course, document type and document grade. The possible values for each category are as follows;

- Document course: We have five courses associated with documents, which are Math, Turkish, Science, Social Sciences and Revolution History.
- Document grade: We have five grade values to represent the grade of the document, which starts from the 4th and ends with 8th.
- Document type: We have 15 different document type values in our data. These types include animation, text, summary, quiz, etc.

To summarize, we have different features to be included in feature vector. Of these five features, we have three boolean features which have 5, 5 and 15 distinct possible values, respectively. Thus, in total when we convert these categorical type features into 0-1 range, we have 27-dimensional feature vector for document specific features.

	Feature Group	Feature Type	Value Range
3.	Document Specific		
	3.13. Document Frequency	float	0-1
	3.14. Unique User Count	float	0-1
	3.15. Document Course	boolean	0-1(5 distinct values)
	3.16. Document Grade	boolean	0-1(5 distinct values)
	3.17. Document Type	boolean	0-1(15 distinct values)

5.1.4 Session Based Features

LETOR algorithms exploit query log data which capture user patterns to predict relevance of a document given a query with the aim of improving search engine performance. Using query log, one can have two different feedbacks, which are

Long History and Short History. Generally, what user searched, which documents he clicked for each particular search and how he searched are in the category of long history feedback. However, sometimes there is no to little information about long history about user or query user issues. This is where short history feedback becomes important.

The search behavior of the user in previous query sessions tells a lot in terms of what user will do next. Besides, in general web search, 44% of all queries submitted into search engine are singleton queries [33]. The feedback problem arises when the query issued is not popular, called tail queries. Therefore we added session based features. The feature group has its own two subgroups, the first one is the general information of previous query session and the second one is the user behavior for documents retrieved in both previous and current sessions. In other words, features in the first subgroup represent general click and dwell behavior of the user in session, while features in the other subgroup give information about documents retrieved in both sessions to capture user preferences about those documents. The complete feature list for session based group is as follows. The first four features belong to the first subgroup whereas the last ones are in the other subgroup, calculated for each document retrieved in the current session.

	Feature Group	Feature Type	Value Range
4.	Session Based		
	4.18. Total Click Count in Previous Session	float	0-1
	4.19. Unique Click Count in Previous Session	float	0-1
	4.20. Total Dwell Time in Previous Session	float	0-1
	4.21. Result Count in Previous Session	float	0-1
	4.22. isClicked	boolean	0 or 1
	4.23. isSkipped	boolean	0 or 1
	4.24. isMissed	boolean	0 or 1
	4.25. DwellTime	float	0-1
	4.26. ClickCount	float	0-1

5.1.5 Query-Document Click Based Features

The last feature group is used to capture click information of documents with respect to the given query. The information of whether a document is clicked or not for a particular query is invaluable in terms of gathering and therefore predicting the relevance of the document given that particular query, which is what we try to do with LETOR algorithms. In this group, we have two features which are basically the number of times a document is retrieved in the result list and clicked for a particular query. This is in fact the second group which tries to associate documents with queries this time by considering impression and click feedback.

	Feature Group	Feature Type	Value Range
5.	Query-Document Click Based		
	5.27. Impression Count	float	0-1
	5.28. Click Count	float	0-1

In total, we have 28 features calculated for each query-document pair to be used in LETOR algorithms. We normalized each numeric feature into 0-1 range and converted categorical features into bit representations to use in LETOR. Therefore, we ended up having a 50-dimensional feature vector for query-document representation.

5.2 Cluster Models

Apart from the general LETOR model we have learned using our features, we tried to explore performance behavior of different models learned by LETOR algorithms. As in the literature, we tried to group queries considering characteristics of the educational search domain. As a result, we come up with two different clusters namely, course of the query issued into search engine and the user grade who issues the query. In addition to those clusters, we also added head-tail cluster analysis as well to reflect the general web search behavior into our educational search engine domain.

In our query log, we have five different courses, which are Math (Matematik), Turkish (Türkçe), Science (Fen Bilgisi), Social Sciences (Sosyal Bilgiler) and Revolution History (İnkilap Tarihi). We also include another course type into this set, that we call General for the queries we cannot differentiate for a specific group defined above. In total we have six different courses and for each we learned a different model.

Similar to course, we have five different user grade types, starting from 4th grade to 8th grade. These grade types reflect the user who issues the query into the search engine. Another cluster group is query frequency analysis which is highly popular in general web search engine literature. Therefore, we have three different cluster groups, in which there are 13 different models we learned using LETOR algorithms.

5.3 Propagation Algorithm

After having cluster analysis, we thought that we need to do better for singleton queries in terms of NDCG metric (the results can be seen in Chapter 6). The reason why singleton model performs poorly is that for singleton queries there are zero valued features, which makes it difficult for algorithms to learn the model. Specifically, *click* and *impression* are the features whose values have zero values for the singleton queries. In other words, *click* and *impression* represent the number of clicks and the number of impressions a document gets given a particular query. Since singleton queries have a frequency of 1, values of these features for the singleton queries are zero, which does not help the model to learn through these features. Therefore, we tried to improve singleton queries' performance by trying to simulate (propagate) click-through features for singleton queries only. In order to come up with an effective solution, we looked through characteristics of our dataset.

In our query log data, we have approximately 3500 unique documents, which is a considerably small number considering the amount of documents in web. Therefore, we tried to exploit that behavior regarding the intersection of the lists of two different queries, in order to find similar queries to that particular singleton query. We tried to predict values of click features by looking other queries having click for that document included in a singleton query session.

Our algorithm has two preparation phases in order to generate values of the features for singleton queries. First, we need to find out the query set of a document which is retrieved no later than the 5th position. Second, we find the similar query set for each singleton query that shares a common document retrieved in their result lists with all the queries included in the set. The details of the mentioned algorithms can be seen in Algorithms 1 and 2, respectively.

After the construction of the data structures to be used, we have a main algorithm that generates a value for the features, namely *click* and *impression*. We propagate *click* and *impression* features of similar queries to the singleton

Algorithm 1 Inverted Document Structure Algorithm

Input: 2D Matrix, `Pairs`, Containing Query-Document Pairs Found Using Result Lists of Queries
Output: 2D Matrix, `InvertedPairs`, Containing Document-Query Pairs

```
1: procedure INVERTEDDOCUMENT(Pairs)
2:     InvertedPairs ← {}
3:     for each query q in Pairs do:
4:         for each document d in Pairs[q] do:
5:             if position of d is later than 5 then:
6:                 continue
7:             else
8:                 if InvertedPairs contains document d then:
9:                     add query q into InvertedPairs[d]
10:                else
11:                    list ← {}
12:                    add query q into list
13:                    put (d, list) into InvertedPairs
14:                end if
15:            end if
16:        end for
17:    end for
18:    return InvertedPairs
19: end procedure
```

query so that we can reduce the penalization of having zero valued features in learning phase of the model.

In our *Propagation* algorithm, we employ three different ideas which are then combined to have a final score. First, we compare the grade values of the users who issue the similar queries. If the grades are equal to each other, we assign 1 as its score and 0 otherwise, which is defined as

$$G(q_i, q_j) = \begin{cases} 1, & \text{if } g_i = g_j \\ 0, & \text{otherwise} \end{cases}$$

where g_i and g_j represent the grade values of the users who issue the queries q_i and q_j respectively for which the similarity score is calculated.

Second, we calculate the textual similarity of the similar queries using *Cosine*

Algorithm 2 Similar Query Set Algorithm

Input: 2D Matrix, `InvertedPairs`, Containing Document-Query Pairs and 2D Matrix, `Pairs`, Containing Query-Document Pairs

Output: 2D Matrix, `SimilarPairs`, Containing Similar Query Set for each Singleton Query

```
1: procedure SIMILARSET(InvertedPairs, Pairs)
2:     SimilarPairs ← {}
3:     for each singleton query q in Pairs do:
4:         for each document d in the result list of Pairs[q] do:
5:             for each query q' in InvertedPairs[d] do:
6:                 if q' is not singleton then:
7:                     add query q' into SimilarPairs[q]
8:                 end if
9:             end for
10:        end for
11:    end for
12:    return SimilarPairs
13: end procedure
```

Similarity metric, which is defined as

$$\cos(q_i, q_j) = \frac{q_i \cdot q_j}{\parallel q_i \parallel \parallel q_j \parallel} , \tag{5.1}$$

where q_i and q_j represent the similar queries for which cosine similarity score is calculated. The Equation 5.1 gives general definition of *Cosine Similarity*. We use Vector Space Model to create vectors of words in queries to calculate similarity score. The corresponding cosine similarity using Vector Space model is defined as

$$\cos(q_i, q_j) = \frac{\sum_{t=1}^{n} q_{i_t} \times q_{j_t}}{\sqrt{\sum_{t=1}^{n} (q_{i_t})^2} \times \sqrt{\sum_{t=1}^{n} (q_{j_t})^2}} , \tag{5.2}$$

where n represents the number of unique terms appearing either in q_i or q_j. The denominator part in Equation 5.2 simply shows how to calculate the length of vectors of terms in each query.

The last part of our algorithm calculates the similarity of queries in terms of their result lists. Therefore, we employ *Jaccard Similarity* between similar queries

to assign a score for the corresponding part. Suppose that we have result lists τ_i and τ_j for queries q_i and q_j respectively. Then the jaccard similarity score between the result lists is defined as follows:

$$J(q_i, q_j) = \frac{\parallel \tau_i \cap \tau_j \parallel}{\parallel \tau_i \cup \tau_j \parallel} . \tag{5.3}$$

Afterall, given a singleton query q_i with the set of similar queries, the similarity score of q_i with respect to each query q_j in the set is calculated as

$$S(q_i, q_j) = \alpha \times G(q_i, q_j) + \beta \times \cos(q_i, q_j) + \gamma \times J(q_i, q_j) , \tag{5.4}$$

where α, β and γ represent the constants to be used in the calculation. Note that, since each part gives a score in between 0 and 1, the final score is also in the same interval. The optimum value for each constant is finalized using parameter tuning method.

Given the singleton query q_i, we calculate the similarity score of each query in the similar set created before using the Equation 5.4. Then the queries are sorted according to their similarity scores. We take first 10 queries into consideration when generating the feature values of q_i to avoid noise. Then *click* and *impression* values of q_i are generated as

$$q_{i_{c,i}} = \frac{\sum_{s=1}^{n} q_{s_{c,i}} \times S(q_i, q_s)}{n} , \tag{5.5}$$

where $q_{s_{c,i}}$ denotes the click and impression values of the query q_s in the similar set of q_i for which click and impression values are generated. n denotes the number of queries taken into consideration while calculating the values.

Chapter 6

Experiments

6.1 Dataset and Annotation

For our research, we used a commercial query log data provided by SEBIT company which mainly produces educational material that can be viewed on a web portal called Vitamin. Our query log data includes queries submitted to Vitamin in December 2013. Search engine on Vitamin Portal is used by students to get educational material provided by SEBIT. For the one month long data, we have 66.908 queries issued by students. Of all, we have 18.638 unique queries, which is 0.27 of all query volume. The behavior for unique queries differs from web where 0.50 of the queries in a typical web search log are unique. We have 18K unique users who issued the queries in that particular span of time. On the average, users ask 3.61 queries in 1.92 sessions.

To have LETOR environment, we needed an annotated data where each query-document pair is scored in terms of their relevance to each other. To do that, we first sampled 900 queries from our query log without any consideration, which is basically random. For each unique query sampled for annotation, we first found the documents retrieved given that query in order to create document lists to be annotated. Then we asked judges to annotate these documents given query text, document title and document description with 3-scale scoring method. The list of

judges consists of graduate students and professors, all of whose native language is Turkish.

For the annotation part, we carried out two different annotations. The first one is the categorical annotation of query text in terms of course to which that query may belong. We had five different courses initially, which is derived from the query log and we also added another course category named "General Course", which we can use for queries that cannot be categorized among possible course candidates, i.e., queries like "Oyunlar (Games)". In total, we have six different courses that could be matched for a given query, which are Math, Turkish, Science, Social Sciences, Revolution History and General Course.

The second part is usual annotation scheme for LETOR algorithms which is to give relevance score for each document associated to a particular query. We used 3-scale method regarding data behavior we observed from our query log. In detail, 3-scale scores are

- 0 - Irrelevant document
- 1 - Mostly relevant (course and subject of the document matches with query, yet document does not satisfy the user needs according to the query text)
- 2 - Exact match (exactly what query asks for).

In total, we annotated 900 unique queries submitted to Vitamin in December 2013. Therefore, we have 3169 queries annotated which are different issues of 900 unique queries to be used for LETOR algorithms. We also had some experiments on annotated data to explore validity of our annotation and potential improvement that we could achieve using LETOR algorithms on original data.

The first experiment is to figure out whether there is a correlation between user clicks and relevant documents regarding our annotation. The Figures 6.1 and 6.2 show the percentage of documents for each relevance score according to clicks and non-clicks. We tried to analyze the relevance scores for each clicked and non-clicked document independent from document ranks.

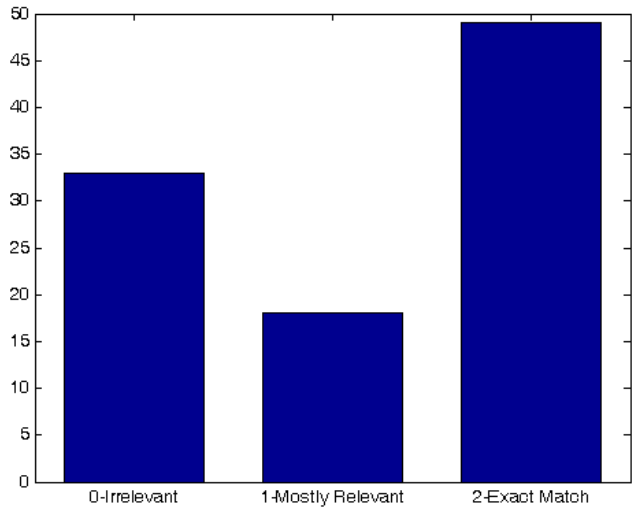

Figure 6.1: Relevance Distribution for Clicked Documents

Looking at Figures 6.1 and 6.2, we can clearly state that there is correlation between relevance and clicked information of documents, which is important to move on LETOR algorithms in order to exploit click feedback from users to improve general performance. The figures also prove the validity of our annotation, since average relevance score for clicked documents is much higher than the score for non-clicked ones. Another aspect of the figures would be potential improvement we could achieve looking at the non-relevant document volume for clicked documents and relevant document volume for non-clicked documents. Using LETOR, we try to minimize percentage of before mentioned volumes.

Another experiment we performed on annotation data is to find out correlation between relevance scores and ranks of documents. The first analysis was done independent of the information that whether the document is clicked or not. Latter figures are to show the correlation for each case; clicked documents and non-clicked documents.

The main outcome of Figure 6.3 is that there is a decreasing trend from position 1 to 25 in terms of relevant documents. Similar but increasing trend for irrelevant documents goes for as well, that is, volume of irrelevant documents are increasing from position 1 to 25. This is consistent with the previous figure. Yet, Figure 6.3

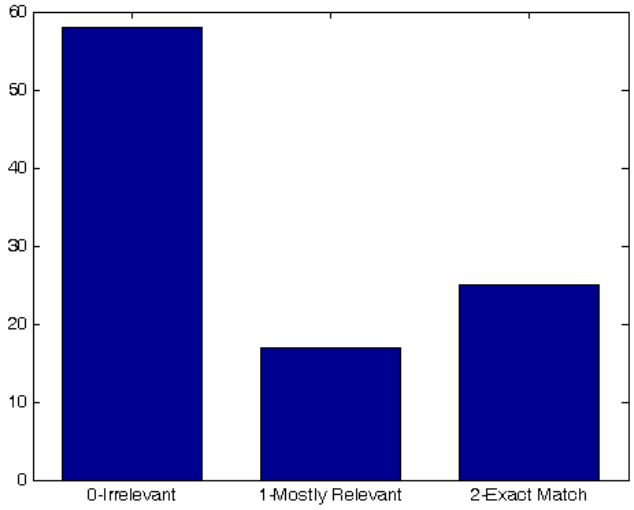

Figure 6.2: Relevance Distribution for Non-Clicked Documents

shows the potential improvement that can be achieved to enhance general search performance.

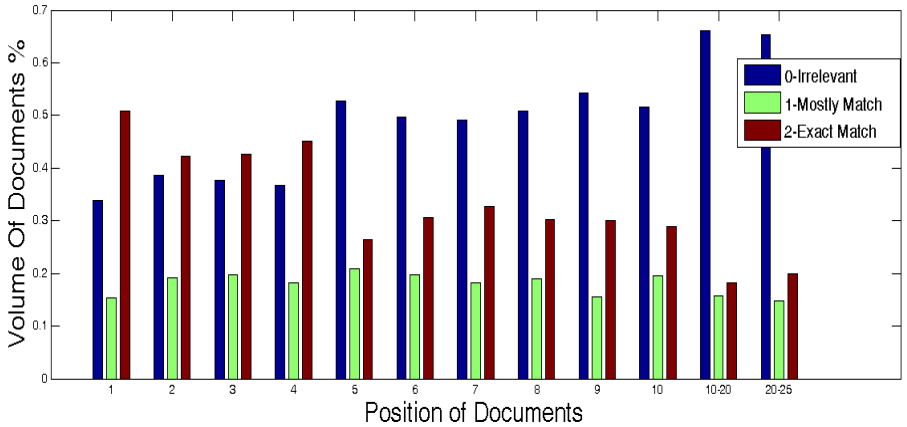

Figure 6.3: Relevance Distribution of Documents for Each Position

The results are as expected in Figure 6.4, that is the volume of relevant documents is much higher than the volume of irrelevant documents for each position. Again, we can easily say that documents clicked at later positions are less likely to be relevant than the documents clicked at first five positions.

The results shown in Figure 6.5, which is the correlation between relevance and

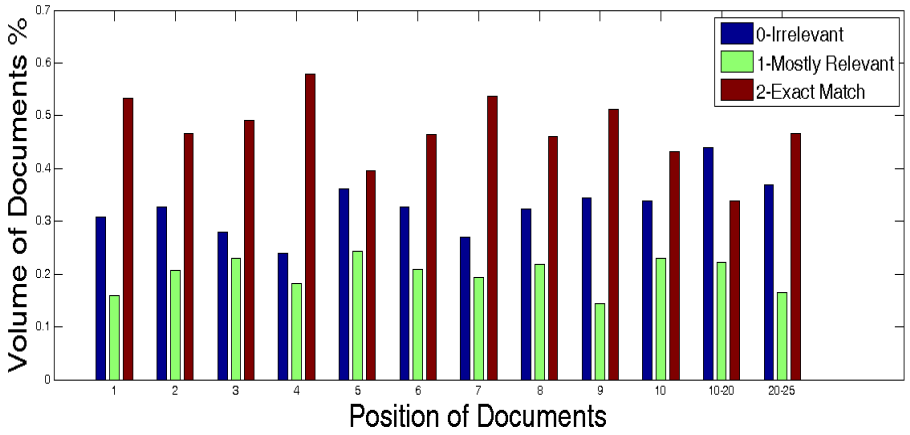

Figure 6.4: Relevance Distribution of Documents for Each Position for Clicked Documents

position for non-clicked documents, are consistent with the previous annotation analysis we have made. This time, volume for irrelevant documents are much higher than the relevant ones and this difference is more dramatic when positions get bigger. Yet, the figure gives simple motivation that especially for the first five positions, there are some relevant documents that are not clicked. This is the general problem SEBT search engine has that we are planning to address to improve.

6.2 Data Preparation and Pre-Processing

6.2.1 Training and Test Sets

After annotation we have 900 unique queries labeled along with their related documents retrieved in their result list. For some unique queries, we have multiple user issues in our query log data. For each issue, we determine unique identifier representing the feature vector. In total, we have 3169 query issues each of which has a unique query identifier. In order to use LETOR algorithms we needed to determine our training and test sets independently not to create any bias. We sorted query issues according to their time stamp values. Recall that, our query

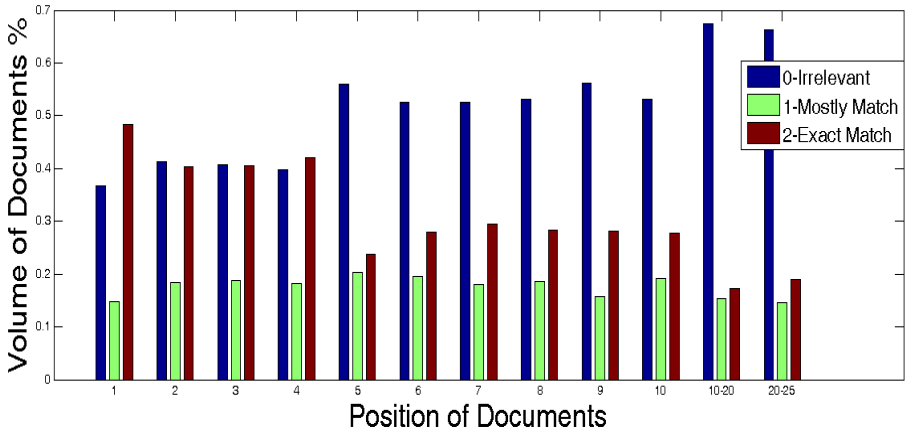

Figure 6.5: Relevance Distribution of Documents for Each Position for Non-Clicked Documents

log data is one month log, specifically from December 2013. In order to have adequate instances for both training and test sets, considering the ratio between training and test sets in terms of instances, we divided our entire query issues into two. While dividing data into two, we set number of instances for training set to be four times as much as the number of instances for test set. Consequently, we have 2536 and 633 instances for training and test respectively. While determining 633 instances for test set, we chose the instances according to their time stamp as mentioned, that is, we chose the queries issued last in the given period of time to constitute test set.

Another issue while preparing the data to be used in LETOR is the calculation of the features, specifically related to long history information. While calculating features like query-document click based features, document click frequency, and user frequency, we only consider instances from training set excluding test instances not to introduce any bias into our data.

6.2.2 Pre-Processing Data

In our data, we have text information available specifically for query text, document title and description. In order to calculate query-document text similarity

features effectively (tf-idf and BM25), we pre-process these texts to have more meaningful and valid scores. In order to calculate these textual features, we implemented inverted file structure to first store the terms occurring in documents, which is to create corpus. The creation of corpus is critical in the sense that, each unique term appearing in corpus will be related to documents according to their occurrences in that particular documents. While relating each unique term to documents, it is important to have healthy list of unique terms. In other words, terms who have the same root must be considered as a single unique one. This is where stemming becomes important. Another issue is as in general web, queries might include some meaningless characters or punctuation marks, which makes it harder to detect query term to match documents. In addition to those, it is important to remove some terms appearing too much, called stop-words to have more valid text similarity scores.

To sum up, for pre-processing data we have 3 steps done following one another. The pre-process steps are explained in detail as follows:

I. Stop Word Removal:

   There is a public list of words containing common stop words and conjunctions used in Turkish, published by Zemberek [61], which is an open-source natural language processing library developed for Turkish researchers. We did not take into consideration words appearing in this list while generating our corpus.

II. Text Cleaning:

   We cleaned query texts, document title and description by removing meaningless characters not coded in Unicode and punctuations to better capture term split by space.

III. Stemming:

   Turkish is a free constituent order language, i.e., according to text flow and discourse context at certain phrase levels, its constituents can change order and still be intelligible [62]. For example, the sentence

"İstanbul Ankara'dan daha güzel." ("İstanbul is more beautiful than Ankara.") and the sentence "Ankara'dan İstanbul daha güzel.", which is an inverted sentence ("devrik cümle" in Turkish), have the same meaning with a slight difference in emphasis [62].

Turkish is an agglutinative language, meaning suffixes are appended to the root to generate new words, which naturally makes it a complex language in terms of word structures. Another important issue is that current stemming algorithms are developed generally for English. However, as stated in [63] to stem root of the words in Turkish texts, getting the first k characters of the term as the root is practical and effective solution, performing similar to stemming algorithms developed for Turkish language. After experimenting multiple k values as the first characters to select to be root, we decided to use the first five characters as the root of the term appearing in both query and documents.

6.3 Baseline and LETOR Model

6.3.1 Baseline Performance

In order to evaluate the performance of our LETOR method, we need to have a baseline approach to compare with. In general, primitive search engines try to rank documents regarding the query-document text similarity scores, mainly tf-idf like scores. When a query is entered, the search engine first calculates tf-idf or BM25 score for each document, then sorts the documents accordingly to show the retrieved list to the user. We have two different query-document text similarity features, which are tf-idf and BM25. We have also two texts (title and description) related with documents, meaning that we have four possible different textual scores, which can be used to sort the documents accordingly as baseline.

In addition to those individual cases, we also introduced linear combination of scores for each textual score for both tf-idf and BM25. While doing linear

Table 6.1: NDCG@K Values for each Baseline

	Training		Test	
	@5	@10	@5	@10
tf-idf Title	0.6422	0.6626	0.6561	0.6710
tf-idf Description	0.6010	0.6306	0.6259	0.6530
BM25 Title	**0.6463**	**0.6670**	0.6578	0.6735
BM25 Description	0.6106	0.6351	0.6455	0.6655
tf-idf Linear (0.7)	0.6401	0.6618	0.6608	0.6722
BM25 Linear (0.7)	0.6375	0.6610	**0.6692**	**0.6804**
SEBIT	0.6210	0.6532	0.6370	0.6710

combination, we normalized each score for document title and description among scores of documents within the query session. For the constant in linear combination, we used parameter tuning method. In our case, 0.7 gives the best result as the constant, therefore we used it as the multiplier for document title score. Lastly, we also put the results produced by the search engine by using LETOR algorithms with these four textual features only.

Table 6.1 shows NDCG scores for each baseline explained above for both training and test sets having K values set to 5 and 10, respectively. The last row presents the search performance of the original SEBIT ranking according to our annotations. Between document title and description, title gives better results for each case, tf-idf and BM25. Besides, individually tf-idf description score calculated using tf-idf metric cannot outperform SEBIT's original ranking for each K value. On the other hand, document ranks according to textual feature of title for both tf-idf and BM25 outperform original ranking of SEBIT. Using both textual features by linear combination gives the best result among all possible baselines. Between tf-idf and BM25, BM25 gives better results so we chose BM25 score with linear combination case to be our baseline to be compared with our general LETOR model learned with our 50-dimensional feature space.

Regarding the results presented in Table 6.1, in general baseline results are similar to the original ranking SEBIT has. Recall that, we did pre-processing for texts of document title and description along with stemming suitable for Turkish language. Although SEBIT's original ranks come from Lucene Based

algorithm, we have better baseline performance thanks to pre-processing of the data. Relative to SEBIT's original ranking, baseline performs slightly better with 3% improvement in NDCG @5 score.

6.3.2 General LETOR Model

Table 6.2 shows the improvement we achieved using LETOR algorithms over the original ranking SEBIT has. From Table 6.2, it is clearly seen that our model learned using LETOR algorithms with our derived feature set outperforms SEBIT's original ranking significantly, by a margin bigger than 14%. Apart from SEBIT's original ranking, we enhanced the baseline performance by almost 11%. Therefore, we can safely state that our suggested model learned by LETOR algorithms using our derived features considerably outperforms SEBIT's original ranking.

Table 6.2: Evaluation of General LETOR Model With Respect to SEBIT and Baseline

	NDCG		ERR	
	@5	@10	@5	@10
Baseline	0.6692	0.6894	0.5762	0.5614
General LETOR	**0.7742**	**0.7888**	**0.6205**	**0.6266**
SEBIT	0.6370	0.6711	0.5521	0.5583

6.4 Feature Group Analysis

Our 50-dimensional feature set has five distinct feature groups related all together constituting the feature vector. We tried to analyze feature groups, that is, which one performs better with respect to others and General Model. Again, we used LETOR algorithms with features only belonging to some particular feature group to see individual performance of each group.

Table 6.3 shows performance results for individual feature groups using

Table 6.3: The Performances of Each Feature Group

	NDCG		**ERR**	
	@5	@10	@5	@10
Baseline BM25	0.6692	0.6804	0.5762	0.5614
(Baseline LETOR) 1	0.6868	0.7095	0.5793	0.6030
(Query Specific) 2	0.6370	0.6711	0.5522	0.5584
(Document Specific) 3	0.6747	0.7101	0.5648	0.5646
(Session Based) 4	0.6349	0.6686	0.5481	0.5549
(Document Click) 5	0.7382	0.7583	0.6055	0.6123
All LETOR	**0.7742**	**0.7888**	**0.6205**	**0.6266**
SEBIT (Baseline)	**0.6370**	**0.6711**	**0.5521**	**0.5583**

LETOR models learned by only features belonging to that group. In addition, we added performance scores for the General Model and the baseline model to make the comparison more meaningful. Results clearly indicate that the best feature group for performance is the 5th group, which is query-document click features, basically consisting of number of clicks and impression of documents given query. We used these two features as separate, while in some works these are combined as a single feature called CTR (Click Through Rate). For LETOR models, the best feedback to be used in algorithms as features seems to be click related ones for each query-document pair.

From Table 6.3, we can assert that textual features, composed of tf-idf and BM25, perform better with the model learned by LETOR algorithm than the one in which they are simply sorted. Query Specific and Session Based Features alone can outperform neither Baseline nor SEBIT's original ranking. Yet, it is clear regarding the margin between General model and best feature group in terms of performance that these feature groups add some information learned by LETOR to further improve the search engine performance. Another outcome is that learning object types including course, grade and type carry helpful information to enhance search engine performance using LETOR regarding the positive difference between Document Specific Feature Group and other baselines, which are the baseline with textual features and SEBIT.

6.5 Cluster Based Analysis

The training and test instance counts vary for each group, which can be seen in Table 6.4 in detail relative to the cluster group they belong to.

Table 6.4: Training and Test Instance Counts for Each Cluster Model

Cluster - Value	**Instance Count**	
	Training	Test
Grade - 4	875	201
Grade - 5	300	90
Grade - 6	374	82
Grade - 7	874	205
Grade - 8	113	55
Course - Math	432	92
Course - Turkish	424	107
Course - Science	344	121
Course - Social Sciences	137	38
Course - Revolution History	116	33
Course - General Course	1038	242
Head	2024	510
Tail	512	123
TOTAL	2536	633

We tried to figure out whether we can improve search engine performance by having different models for each cluster group. We again used NDCG and ERR metrics for 5 and 10 as k values. These two values are chosen to reflect our query log having 5 relevant documents per query, and the literature respectively. Results show that, having different models for each cluster type improves the general LETOR model's performance besides improving the performance for most types separately.

6.5.1 Course Cluster Results

We have 3169 instances in total. Of 3169, we use 2536 instances for training data and 633 instances for test data. Of those training and test sets, we separated

Table 6.5: Search Engine Performance for Course Cluster in NDCG Metric

NDCG	**SEBIT**		**General Model**		**Course Model**	
	@5	@10	@5	@10	@5	@10
Math	0.7429	0.7534	0.7459	0.7547	**0.7854**	**0.7839**
Turkish	0.6659	0.7114	**0.8034**	**0.8122**	0.7564	0.7886
Science	0.5632	0.6086	0.6889	0.7155	**0.6890**	**0.7193**
Social Sciences	**0.6798**	**0.6954**	0.6075	0.6492	0.6568	0.6953
Revolution History	0.7560	0.7711	**0.8404**	**0.8448**	0.8377	0.8232
General Course	0.5984	0.6361	0.8283	0.8261	**0.8389**	**0.8447**
AVG	0.6370	0.6711	0.7742	0.7888	**0.7775**	**0.7923**

different training and test data for each course type. Therefore, we have 6 different models learned separately for each course type.

As we can see from Table 6.5, average performance of course models is slightly better than the general model performance in which all instances are used without any grouping. Apart from general performance improvement, there is also improvement for individual courses that are of Math, Science and General Course types. Although the best performance for social sciences group seems to be obtained with SEBIT's original rank, we also improved this course type's performance from 0.60 to 0.65, which is relatively significant.

Another observation is that the general model outperforms cluster models for the course types which are Turkish and Revolution History. This result is due to the fact that these courses are text oriented courses, therefore documents related to those courses have much more meaningful texts, which automatically improves the textual features we have, which are tf-idf and BM25. Therefore, since we have more instances with the general model, it behaves better than the cluster models. However, we believe that if we had enough number of instances for each query course type, then we might have expected this behavior to change.

Similar to the values obtained with the NDCG metric, the results for the ERR metric for Math and Science courses show that course models outperform the general model. Yet, in terms of ERR, the average performance of course models is in general slightly worse than that of the general model.

Table 6.6: Search Engine Performance for Course Cluster in ERR Metric

ERR	**SEBIT**		**General Model**		**Course Model**	
	@5	@10	@5	@10	@5	@10
Math	0.6407	0.6452	0.6386	0.6438	**0.6570**	**0.6700**
Turkish	0.5604	0.5697	**0.6486**	**0.6527**	0.6123	0.6199
Science	0.4068	0.4162	**0.4519**	**0.4599**	0.4370	0.4324
Social Sciences	0.5438	0.5496	0.4906	0.5024	**0.5109**	**0.5320**
Revolution History	0.6011	0.6065	**0.6537**	**0.6592**	0.6069	0.6409
General Course	0.5820	0.5861	0.7051	0.7071	**0.7082**	**0.7106**
AVG	0.5521	0.5583	**0.6205**	**0.6266**	0.6156	0.6218

6.5.2 Grade Cluster Results

We have five different values for grade cluster indicating grades of users who issue the query into the search engine. Therefore, we have learned five different models for this cluster analysis, and the results show that clustering user issues according to the user grade values enhances the search engine performance. The results also indicate that this cluster group gives the best result in terms of the average general search engine performance compared to other clusters and the general model.

Looking at Table 6.7, we can clearly see that the cluster for user grade enhances the search engine performance by almost 1% relative to the general model performance. Besides, we can state that for almost each grade model, there is further improvement except for the 4th grade. This cluster model outperforms the course cluster model and the general model in terms of NDCG metric. Looking at these numbers, we can conclude that, students who use search engine for educational purposes can be grouped according to their grades, since students in each grade type have different search characteristics which makes the difference for grade models among other cluster groups. Hence, considering the improvement relative to SEBIT's performance, we can safely say that grouping LETOR instances for each user grade seems a good choice to further improve the search engine performance.

Evaluation scores with the ERR metric show similar results to those with the

Table 6.7: Search Engine Performance for Grade Cluster in NDCG Metric

NDCG	**SEBIT**		**General Model**		**Grade Model**	
	@5	@10	@5	@10	@5	@10
4th Grade	0.6473	0.6851	**0.7996**	**0.8054**	0.7857	0.8030
5th Grade	0.6452	0.6640	0.6194	0.6540	**0.6504**	**0.6759**
6th Grade	0.4722	0.5257	0.7419	0.7464	**0.7672**	**0.7619**
7th Grade	0.6732	0.7005	0.8463	0.8476	**0.8512**	**0.8599**
8th Grade	0.6968	0.7410	0.7453	0.7594	**0.7633**	**0.7629**
AVG	0.6370	0.6711	0.7742	0.7888	**0.7833**	**0.7945**

Table 6.8: Search Engine Performance for Grade Cluster in ERR Metric

ERR	**SEBIT**		**General Model**		**Grade Model**	
	@5	@10	@5	@10	@5	@10
4th Grade	0.6107	0.6162	**0.6650**	**0.6696**	0.6514	0.6531
5th Grade	**0.5596**	**0.5661**	0.5294	0.5367	0.5486	0.5375
6th Grade	0.2949	0.3041	0.4274	0.4327	**0.4475**	**0.4480**
7th Grade	0.5948	0.6004	0.7122	0.7154	**0.7129**	**0.7192**
8th Grade	0.5506	0.5562	0.6010	0.6060	**0.6095**	**0.6181**
AVG	0.5521	0.5583	0.6205	0.6266	**0.6266**	**0.6284**

NDCG scores. Results can be seen from Table 6.8. Although, we could not improve the general search engine performance by using course clusters in terms of the ERR metric, with the models learned for each different grade, we have approximately 0.6% improvement relative to the general model performance. To conclude, for both NDCG and ERR metrics, grade cluster performs the best and gives the best results for average overall search engine performance.

6.5.3 Frequency Cluster Results

For LETOR setups, the best feature group which adds most to LETOR model is click through features which are basically document impression and click count per query. It directly gives feedback about whether that document is related to the given query. Having click information for a specific document given query tells a lot about relevance of that document. Therefore, as in the literature, we also added these features, also known as CTR (Click-Through Rate), into our

Table 6.9: Search Engine Peformance for Frequency Cluster in NDCG Metric

NDCG	**SEBIT**		**General Model**		**Frequency Model**	
	@5	@10	@5	@10	@5	@10
Non-singleton	0.6446	0.6757	0.8180	0.8186	**0.8249**	**0.8280**
Singleton	**0.6054**	**0.6521**	0.5923	0.6386	0.5795	0.6301
AVG	0.6370	0.6711	0.7742	0.7888	**0.7772**	**0.7895**

Table 6.10: Search Engine Peformance for Frequency Cluster in ERR Metric

ERR	**SEBIT**		**General Model**		**Frequency Model**	
	@5	@10	@5	@10	@5	@10
Non-singleton	0.5688	0.5740	**0.6586**	0.6614	0.6571	**0.6645**
Singleton	0.4814	**0.4932**	0.4743	0.4869	**0.4864**	0.4906
AVG	0.5521	0.5583	0.6205	0.6266	**0.6239**	**0.6307**

feature set. And the results show that the best performing feature group for our data set consists of these features (Section 6.4). However, the problem is, there are instances or query issues which we have no click feedback about prior to user issue. These queries are called singleton queries.

Generally, queries issued few number of times are also considered as tail ones in Web. Yet, considering our data environment and behaviors, we called queries issued exactly once (singleton) tail queries. The problem arises with these queries, since in the feature set click-through features of tail queries are zero. Therefore, we cannot use our best performing features for that special group of queries. Thus, we tried to cluster queries according to their frequencies, mainly non-singleton or singleton, to explore potential improvements we could gain having different LETOR models. Another problem with our singleton dataset is that, our training and test instance count is relatively small compared to other cluster groups. Hence, as expected, there is huge difference between non-singleton and singleton models in terms of search engine performance.

For this cluster, NDCG and ERR results show slightly different scores, which can be seen from Tables 6.9 and 6.10, respectively. For singleton queries, in terms of the NDCG metric neither the general model nor the singleton model

outperforms SEBIT's original ranking. Yet, for the ERR metric the singleton model outperforms both the general model and SEBIT's original ranking. Results for non-singleton queries are satisfying for both metrics with the non-singleton model performing better for the NDCG metric while the general model behaves better for the ERR metric. However, in terms of overall average performance we have still slight improvement relative to the general model for both metrics.

6.5.3.1 Singleton Queries Improvement

For singleton queries (queries issued into search engine exactly once), the values of the certain features are zero, which are specifically the number of times a document is seen as a result of a query issue and a document is clicked as a result of search. The values of the above features are zero, since the system sees the query issue for the first time. Having a zero value for the features causes the model to perform poorly for singleton queries. Therefore, we propose an algorithm to generate synthetic values for the features to enhance the performance of the model.

We also implemented a methodology similar to the one used in [1] to compare our algorithm with theirs as well as our baselines. To compare the results, we set up two different environments. After generating the features for a particular method, we employ a model using all the instances including the non-singleton ones that we call the general model with generated features. The second model is the one learned using the singleton queries only.

Gao et al. [1] introduce a set of features called *clickstream* features that could be useful for queries having little information in query log. For each query document pair, they calculate a set of features independent from both contextual and click information. They basically find a set of queries for which a certain document is clicked. Given a singleton query, using the queries in the aforementioned set, they calculate the values of the features. Apart from clickstream features, they also introduce a technique called *Discount Method* to further help generating the values when *clickstream* features are zero as well. We implement

Table 6.11: The Results of Models Learned Using Only Singleton Queries

	NDCG		ERR	
	@5	@10	@5	@10
Singleton Queries Model	0.5795	0.6301	0.4864	0.4906
ClickStream Features	0.5828	0.6335	0.4885	0.4911
ClickStream Features + Discount	0.5782	0.6245	0.4826	0.4904
Propagation Algorithm	**0.6061**	0.6367	**0.4829**	**0.5028**
SEBIT	0.6054	**0.6521**	0.4814	0.4932

both methods and compare the results of them with our algorithm in each approach defined above. We choose these methods due to usage of similar queries to generate values for zero valued features, which is related to what we propose.

The results of the first approach that is the learned models using only singleton queries are shown in Table 6.11. The results clearly present that our proposed algorithm performs the best among all methods along with the original ranking SEBIT has. While *clickstream* features slightly improves the model, the discount method used on top of it does not seem to be helpful in terms of performance. The reason would be that discount method gives the same value for each instance and since the number of instances are small compared to all model, this might introduce noise to the model. Our proposed algorithm improves the singleton query model by approximately 3%.

Table 6.12 shows the results of the second approach. In this approach, *Discount Method* performs better compared to the previous approach, improving the performance of the model learned using *clickstream* features. However, our *Propagation Algorithm* gives the best result again in terms of the metrics calculated @5, which better reflects the characteristics of our query log. The performance improvement however is lower compared to the previous approach. The reason is that in this model head instances dominate the model due to higher number of instances.

We also demonstrate the results of the general model learned using generated synthetic features, which can be seen in Table 6.13. Although, the difference in performance compared to the general LETOR model is lower, the improvement

Table 6.12: The Results of Models Learned All Queries With Generated Features

	NDCG		ERR	
	@5	@10	@5	@10
Singleton Queries Model	0.5923	0.6386	0.4743	0.4869
ClickStream Features	0.5800	0.6289	0.4630	0.4754
ClickStream Features + Discount	0.5861	0.6204	0.4530	0.4618
Propagation Algorithm	**0.6073**	0.6425	**0.4898**	**0.4995**
SEBIT	0.6054	**0.6521**	0.4814	0.4932

Table 6.13: The General Results of Models Learned All Queries with Generated Features

	NDCG		ERR	
	@5	@10	@5	@10
General LETOR Model	0.7742	0.7888	0.6205	0.6266
ClickStream Features	0.7713	0.7848	0.6185	0.6191
ClickStream Features + Discount	0.7748	0.7931	0.6238	0.6305
Propagation Algorithm	**0.7772**	**0.7942**	**0.6253**	**0.6343**
SEBIT	0.6370	0.6711	0.5521	0.5583

is positive. Recall that, we only generate the values of the features for singleton queries. In other words, in the general model case, we only change the values of the features for instances representing singleton queries without touching the instances for non-singleton queries which dominate the model in terms of instances. Therefore, we believe that if we have a more balanced training set, the change would be more dramatic.

Chapter 7

Conclusion

There has been a growing interest in the field of web search engine research due to immense data available to work with. The idea of using query logs as a source of implicit feedback about user behaviors leads researches to figure out useful search patterns about users. Analyzing the query logs, researchers have exploited the patterns to further improve search engine performance.

We exploited a query log provided by an educational vertical search engine. First, we analyzed the search characteristics of the students who do search and compared our findings with the general web search behaviors. Our analysis demonstrates that there is difference in terms of search characteristics between the general web search engine and an education search engine. Students tend to explore more documents compared to general web users. However, our findings present that students have hard time to express their search intent in query terms. Students are more likely to repeat the queries they once entered into search engine than the general web users.

In our query log analysis, we also tried to find the *Refinding* behavior, which is a well-known problem in the literature. The results again differ from the ones in the case of Web search. While 40% of the queries in web result in a click for a document which is visited before by the user issuing the query, only 25% of the queries in our query log result in *Refinding* behavior. Another difference

comes in search sessions where students prefer same query texts. In web, 87% of search sessions started with the same query text result in *Refinding* behavior. However, this number in our case is only 44%. Similar to this analysis, the ratio of the search sessions with the same query text having *Newfinding* behavior is 38%, while the corresponding result in our query log is 74%. The reason behind these differences is that students periodically study different subjects, which leads them to explore new documents that they never visited before. Another reason is that learning is a continuous process. While learning for a particular subject, students tend to look for new educational material to capture more aspects of the subject.

Using the search patterns found in the query log analysis, we proposed an educational learning model. While creating the model, we introduced novel features best suited for the educational domain. Using the learning to rank algorithms, our proposed model outperformed both the original ranking of SEBIT and the baselines created by the state of the art techniques by up to 14% and 11%, respectively. We also employed the cluster idea of grouping the queries to be used in a different model for each cluster. Regarding the educational domain as well, we mainly introduced three different clusters, which are namely course of the query, grade of the user issuing the query, and the frequency of the query. Our results indicate that for each cluster the average performance of the models learned outperforms the general model. Specifically, the grade cluster further improves the general model by up to 1%.

In order to further enhance the performance of frequency models, we proposed an algorithm to generate values of the features for singleton queries where the model performs poorly according to the results due to having zero values for certain features. We compared the results of our algorithm with a similar approach introduced in [1]. We showed that our algorithm performs best among all models in our query log. Using our algorithm, we also managed to improve both the performance of the singleton model and the general LETOR model.

Bibliography

[1] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie, "Smoothing clickthrough data for web search ranking," in *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, (New York, NY, USA), pp. 355–362, ACM, 2009.

[2] Alexa, "The top 500 sites on the web." Available at `http://www.alexa.com/topsites` (2015/06/18).

[3] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large web search engine query log," *SIGIR Forum*, vol. 33, pp. 6–12, Sept. 1999.

[4] A. Broder, "A taxonomy of web search," *SIGIR Forum*, vol. 36, pp. 3–10, Sept. 2002.

[5] A. Spink, B. J. Jansen, I. Taksa, and A. Spink, *Handbook of Web Log Analysis*. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2008.

[6] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman, "Defining a session on web search engines: Research articles," *Journal of the Association for Information Science and Technology*, vol. 58, pp. 862–871, Apr. 2007.

[7] S. Lawrence, K. Bollacker, and C. L. Giles, "Indexing and retrieval of scientific literature," in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, pp. 139–146, ACM, 1999.

[8] W. Weerkamp, R. Berendsen, B. Kovachev, E. Meij, K. Balog, and M. de Rijke, "People searching for people: analysis of a people search engine log,"

in *Proceedings of the 34st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pp. 45–54, ACM, 2011.

[9] Iprospect, "Search engine user behavior study." Available at `http://district4.extension.ifas.ufl.edu/Tech/TechPubs/WhitePaper_2006_SearchEngineUserBehavior.pdf` (2015/05/12).

[10] A. Cockburn, S. Greenberg, S. Jones, B. McKenzie, and M. Moyle, "Improving web page revisitation: Analysis, design, and evaluation," 2003.

[11] J. Teevan, E. Adar, R. Jones, and M. Potts, "History repeats itself: Repeat queries in yahoo's logs," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, (New York, NY, USA), pp. 703–704, ACM, 2006.

[12] S. Mark and S. Dumais, "Examining repetition in user search behavior," in *Proceedings of the 29th Annual European ECIR Conference on Information Retrieval*, ECIR '07, 2007.

[13] T. Qin, T.-Y. Liu, J. Xu, and H. Li, "Letor: A benchmark collection for research on learning to rank for information retrieval," *Information Retrieval*, vol. 13, pp. 346–374, Aug. 2010.

[14] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[15] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval," in *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, (New York, NY, USA), pp. 232–241, Springer-Verlag New York, Inc., 1994.

[16] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, (New York, NY, USA), pp. 275–281, ACM, 1998.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[18] H. Li, *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers, 2011.

[19] G. Giannopoulos, U. Brefeld, T. Dalamagas, and T. Sellis, "Learning to rank user intent," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, (New York, NY, USA), pp. 195–200, ACM, 2011.

[20] M. Shokouhi, "Learning to personalize query auto-completion," in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, (New York, NY, USA), pp. 103–112, ACM, 2013.

[21] G. E. Dupret and B. Piwowarski, "A user browsing model to predict search engine click data from past observations.," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, (New York, NY, USA), pp. 331–338, ACM, 2008.

[22] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey, "An experimental comparison of click position-bias models," in *Proceedings of the ACM International Conference on Web Search and Data Mining*, WSDM '08, (New York, NY, USA), pp. 87–94, ACM, 2008.

[23] O. Chapelle and Y. Zhang, "A dynamic bayesian network click model for web search ranking," in *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, (New York, NY, USA), pp. 1–10, ACM, 2009.

[24] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White, "Evaluating implicit measures to improve web search," *ACM Transactions on Information Systems*, vol. 23, pp. 147–168, Apr. 2005.

[25] Y. Kim, A. Hassan, R. W. White, and I. Zitouni, "Modeling dwell time to predict click-level satisfaction," in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, (New York, NY, USA), pp. 193–202, ACM, 2014.

[26] A. Hassan, X. Shi, N. Craswell, and B. Ramsey, "Beyond clicks: query reformulation as a predictor of search satisfaction," in *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, CIKM '13, (New York, NY, USA), pp. 2019–2028, ACM, 2013.

[27] E. Aktolga and J. Allan, "Reranking search results for sparse queries," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, (New York, NY, USA), pp. 173–182, ACM, 2011.

[28] A. Usta, I. S. Altingovde, I. B. Vidinli, R. Ozcan, and O. Ulusoy, "How k-12 students search for learning?: Analysis of an educational search engine log," in *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '14, (New York, NY, USA), pp. 1151–1154, ACM, 2014.

[29] S. D. Torres and I. Weber, "What and how children search on the web," in *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pp. 393–402, ACM, 2011.

[30] E. Foss, A. Druin, R. Brewer, P. Lo, L. Sanchez, E. Golub, and H. Hutchinson, "Children's search roles at home: Implications for designers, researchers, educators, and parents," *Journal of the Association for Information Science and Technology*, vol. 63, no. 3, pp. 558–573, 2012.

[31] C. Eickhoff, P. Dekker, and A. P. de Vries, "Supporting children's web search in school environments," in *Proceedings of the 4th Information Interaction in Context Symposium*, IIIX'12, pp. 129–137, 2012.

[32] M. E. Bakanlığı, "Meb istatistikleri örgün eğitim 2012-2013." Available at `http://sgb.meb.gov.tr/istatistik/meb_istatistikleri_orgun_egitim_2012_2013.pdf` (2013/06/21).

[33] R. A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "Design trade-offs for search engine caching," *ACM Transactions on the Web*, vol. 2, no. 4, 2008.

[34] I. Weber and A. Jaimes, "Who uses web search for what: and how," in *Proceedings of the 4th Annual ACM International WSDM Conference on Web Search and Data Mining*, WSDM '11, pp. 15–24, ACM, 2011.

[35] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *Proceedings of the International Conference on Scalable Information Systems for Big Data*, InfoScale '06, 2006.

[36] M. Richardson, E. Dominowska, and R. Ragno, "Predicting clicks: Estimating the click-through rate for new ads," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 521–530, ACM, 2007.

[37] E. Agichtein, E. Brill, and S. Dumais, "Improving web search ranking by incorporating user behavior information," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, (New York, NY, USA), pp. 19–26, ACM, 2006.

[38] A. Shashua and A. Levin, "Ranking with large margin principle: Two approaches," in *Advances in Neural Information Processing Systems 15* (S. Thrun and K. Obermayer, eds.), pp. 937–944, Cambridge, MA: MIT Press, 2002.

[39] P. Li, C. Burges, and Q. Wu, "Learning to rank using classification and gradient boosting," in *Advances in Neural Information Processing Systems 20*, no. MSR-TR-2007-74, p. 0, MIT Press, Cambridge, MA, January 2008.

[40] K. Crammer and Y. Singer, "Pranking with ranking," in *Advances in Neural Information Processing Systems 14*, pp. 641–647, MIT Press, 2001.

[41] D. Cossock and T. Zhang, "Subset ranking using regression," in *Proceedings of the 19th Annual Conference on Learning Theory*, COLT'06, (Berlin, Heidelberg), pp. 605–619, Springer-Verlag, 2006.

[42] R. Herbrich, T. Graepel, and K. Obermayer, *Large Margin Rank Boundaries for Ordinal Regression*, ch. 7, pp. 115–132. MIT Press, January 2000.

[43] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon, "Adapting ranking svm to document retrieval," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, (New York, NY, USA), pp. 186–193, ACM, 2006.

[44] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, pp. 933–969, Dec. 2003.

[45] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun, "A general boosting method and its application to learning ranking functions for web search," in *Advances in Neural Information Processing Systems 20* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 1697–1704, Curran Associates, Inc., 2008.

[46] Q. Wu, C. Burges, K. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Information Retrieval*, vol. 13, pp. 254–270, June 2010.

[47] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, (New York, NY, USA), pp. 89–96, ACM, 2005.

[48] M. feng Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "Frank: A ranking method with fidelity loss," Tech. Rep. MSR-TR-2006-155, Microsoft Research, November 2006.

[49] C. Burges, R. Ragno, and Q. Le, "Learning to rank with non-smooth cost functions," in *Advances in Neural Information Processing Systems 19*, MIT Press, Cambridge, MA, January 2007.

[50] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, (New York, NY, USA), pp. 271–278, ACM, 2007.

[51] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma, "Directly optimizing evaluation measures in learning to rank," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, (New York, NY, USA), pp. 107–114, ACM, 2008.

[52] J. Xu and H. Li, "Adarank: A boosting algorithm for information retrieval," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pp. 391–398, ACM, 2007.

[53] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, (New York, NY, USA), pp. 129–136, ACM, 2007.

[54] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: Theory and algorithm," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), pp. 1192–1199, ACM, 2008.

[55] M. Taylor, J. Guiver, S. Robertson, and T. Minka, "Softrank: Optimizing non-smooth rank metrics," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, (New York, NY, USA), pp. 77–86, ACM, 2008.

[56] T. Qin, T.-Y. Liu, and H. Li, "A general approximation framework for direct optimization of information retrieval measures," Tech. Rep. MSR-TR-2008-164, Microsoft Research, November 2008.

[57] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview.," in *Yahoo! Learning to Rank Challenge*, pp. 1–24, 2011.

[58] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, (New York, NY, USA), pp. 41–48, ACM, 2000.

[59] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan, "Expected reciprocal rank for graded relevance," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, (New York, NY, USA), pp. 621–630, ACM, 2009.

[60] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey, "An experimental comparison of click position-bias models," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, (New York, NY, USA), pp. 87–94, ACM, 2008.

[61] A. A. Akın and A. M. Dündar, "Turkish NLP libraries." Available at `https://github.com/ahmetaa/zemberek-nlp` (2015/04/25).

[62] G. Lewis, *Turkish grammar*. Oxford New York: Oxford University Press, 2000.

[63] F. Can, S. Kocberber, E. Balcik, C. Kaynak, H. C. Ocalan, and O. M. Vursavas, "Information retrieval on turkish texts," *Journal of the American Society for Information Science and Technology*, vol. 59, no. 3, pp. 407–421, 2008.