

# On Ranking Techniques for Desktop Search

Research Thesis

Submitted in Partial Fulfillment of  
The Requirements for the Degree of  
Master of Philosophy in Information Management Engineering  
by

**Na'ama Aharony**

Submitted to the Senate of the Technion—Institute of Technology  
Adar, 5767 Haifa March 2007

The Research Thesis Was Done Under The Supervision of:  
**Dr. Sara Cohen** and **Dr. Carmel Domshlak**  
in the faculty of Industrial Engineering and Management

#### ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Sara Cohen and Dr. Carmel Domshlak for their devoted, thorough and patient guidance throughout the course of this work. I would also like to thank all volunteers for participating in the experiment and especially Boaz Dotan. Without you this work would not be possible. I am forever indebted to my parents and my husband Kobi for their understanding, endless patience and encouragement when it was most required. Finally, I would like to thank my son Ori, who was born during this thesis work for teaching me greater things.

The Generous Financial Help Of The Technion Is Gratefully Acknowledged.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related Work</b>	<b>5</b>
2.1	Desktop Search Studies . . . . .	5
2.2	Industrial Development . . . . .	7
<b>3</b>	<b>Framework</b>	<b>9</b>
3.1	Queries and Answers . . . . .	9
3.2	Features of Interest . . . . .	10
3.3	Evaluation Criteria . . . . .	14
<b>4</b>	<b>Experimental Setup and Data Gathering</b>	<b>16</b>
<b>5</b>	<b>Basic Ranking Schemes</b>	<b>19</b>
<b>6</b>	<b>Learning to Rank</b>	<b>23</b>
6.1	Learning a Ranking Function . . . . .	23
6.2	Learning A Simple Aggregation . . . . .	26
6.3	Evaluation . . . . .	28
<b>7</b>	<b>Reasoning to Rank and Selectiveness of the Features</b>	<b>32</b>
<b>8</b>	<b>Secondary Ranking schemes</b>	<b>37</b>
<b>9</b>	<b>Conclusions and Future Work</b>	<b>43</b>

## Abstract

Users tend to store huge amounts of files, of various formats, on their personal computers. As a result, finding a specific, desired file within the file system is a challenging task. This thesis addresses the *desktop search* problem by considering various techniques for ranking results of a search query over the file system. First, basic ranking techniques, which are based on various file features (e.g., file name, access date, file size, etc.) are considered and their effectiveness is empirically analyzed. Next, two learning-based ranking schemes are presented, and are shown to be significantly more effective than the basic ranking methods. Then, a novel ranking technique, based on query selectiveness, is considered for use during the cold-start period of the system. This method is also shown to be empirically effective, even though it does not involve any learning. Finally, we study ranking schemes in which the ranking is performed in two phases. We compare the effectiveness of such schemes with our one-phase learning schemes.

# Chapter 1

## Introduction

Due to the increasing storage capabilities of standard personal computers, users are no longer motivated to delete old files. The opposite is true—users tend to save huge amounts of multi-media data, e.g., text documents, pictures, audio and video files, emails, presentations, etc. The practice of “never deleting files” has distinct advantages, since it ensures that important data will never be accidentally removed. However, as an unfortunate side effect, the personal computer often becomes an unwieldy mess. Thus, locating a specific file, within the file system, may become a challenging (and even daunting) task.

To address this problem, numerous research and industrial projects have evolved in the area of personal information management (PIM). These projects aim to develop technologies allowing users to store, access, and effectively search for information in their personal computer, or even about virtually everything in their personal life’s history [27]. Due to the special nature of its goals, PIM brings together researchers from a wide spectrum of scientific and engineering disciplines, bridging cognitive psychology (where the term “personal information management” was first coined [22]) with database management and information retrieval (IR). In particular, [23] considers PIM to be one of the grand research challenges for the upcoming years.

The focus of this thesis is on *desktop search*, i.e., effective search within a personal computer. This is an important problem, since a desktop search

engine should make it easier to locate the information we need, even when it is buried in vast amounts of unrelated information. The commercial and social success of classical and web information retrieval techniques seem to imply that desktop search can be successfully addressed using the tools developed for the former tasks. Desktop search, however, is different in many ways from search in an unknown data repository, such as a digital library or the Web. Perhaps the foremost difference is that users search their personal desktops mainly for *concrete* files they *know* to exist.

Recently, the challenge of developing a tool for effective desktop search has been studied by both academic and commercial research groups (e.g., [14, 13, 9, 25, 15, 6, 26]). While these tools differ in some capabilities and the form of the user interface, most adopt the same intuitive framework of

1. defining a query as a small set of keywords,
2. retrieving a set of documents that are related in some way to the query keywords,
3. presenting the user with these documents ordered according to some (default) primary ranking scheme, and
4. allowing the user to reorder the document list according to secondary ranking schemes, which are generally attributes such as last-access date, filename, purported relevance of the documents' content to the query, etc.

Interestingly, while the last feature is easy to implement, many commercial desktop search tools seem to prefer more laconic forms of user interface, returning to the user a single (possibly structured) ranked list of search results (e.g., [15, 26]). This supports what we believe to be an inherent property of searching personal data-spheres—the typical assumption of desktop-search

users is that they know how to correctly query the system for their desired file, and they expect the system to locate this file successfully.

All these special properties of the desktop search make the choice of a *primary ranking scheme* (Item 3 above) quite crucial. However, while the body of work on personal information management, and in particular, on desktop search, is continuously growing [1, 3, 14, 8, 13, 9, 11, 25, 27, 24], there are currently only limited (published) insights into the question of how to rank desktop-search results. Interestingly, the observations reported in the literature are somewhat surprising from an IR perspective.

In this thesis we focus on the problem of finding an effective primary ranking scheme for the desktop search. We will mainly focus on “known item” queries, i.e., files that the user knows to exist and can also recognize their names. To obtain the essential user data, we developed a simple desktop-search tool that follows the basic format described in Items 1 to 4 above. We provided this tool to a group of volunteer users. After a while, log data was collected from the users. This data was thoroughly analyzed to determine the relative effectiveness of ranking search results according to numerous standard sorting criteria, e.g., last-update date, textual connection between the content and the query, textual connection between the filename and the query, etc. This analysis also provided us with some interesting findings about desktop-search habits (of our users). Based on these findings, we suggest and evaluate the effectiveness of

1. basic ranking schemes that rank results according to individual file features.
2. two learning-based ranking schemes that can be automatically learned from the system’s log data,
3. a novel approach to rank desktop search results that performs (rather simple) reasoning about the search results, but does not require learning,

and thus, it is suitable for the “cold-start” period of the system, and

4. several two-phase ranking approaches.

Our empirical analysis shows that both the learning-based and the reasoning-based approaches significantly dominate all the basic ranking schemes with respect to our evaluation criteria.

The rest of the thesis is organized as follows. In Chapter 2 we describe related work. In Chapter 3 we formally state the problem of desktop search, suitable evaluation criteria, and the type of information about files that we used in our desktop search system. Chapter 4 describes the system, data, and evaluation setup we used in our analysis. In Chapter 5 we present the results of our study of basic ranking methods, that are typically used in desktop search systems. Chapters 6, 7 and 8 are devoted to the learning-based, reasoning-based and two-phase approaches, respectively, and to their evaluation. We conclude and outline a few directions for future research in Chapter 9.

## Chapter 2

# Related Work

### 2.1 Desktop Search Studies

When developing a desktop search engine it is important to “know the audience”, i.e., to know how users organize their files in the file system and how they retrieve data. In addition, it is important to be familiar with the search behaviors of users.

In [3], two independent studies were compared. The studies examined the way users organize and search for files on their computers in three different operating systems (Windows, DOS and Macintosh). Both studies agreed that there are three types of information on the personal computer:

1. Ephemeral: Short term information containing tasks and memos, which function as reminders. This type of information is usually stored at top-level directories and is used very frequently.
2. Working: Information that is frequently used for ongoing work on a current project, which has a shelf life of weeks or months. This type of information is organized in directories according to categories.
3. Archive: Information that has a shelf life of months or even years and contains data on completed work. This information is rarely used and is not organized in a specific manner.

As a rule, both studies agreed that the older the data is, the less often it is used. The studies claim that “file naming was given careful attention by all users.” It also states that users have in mind the goal of being able to find a file when they name the file and when they place it in a specific location. In addition, both studies agreed that users prefer to perform an active search according to the file location since they need to feel in control and since they usually remember more or less where was the file stored.

The last result was also one of the conclusions of [28] which studied how users perform personally-motivated searches in email, files and the web. They discovered that users usually search in small steps. Each step’s results lead to the next step. In most searches, users knew what they were looking for and remembered the type of file (e.g., email, word document, etc.) which held the information they were looking for.

Several approaches for retrieving information from a personal computer have been suggested in the past. One of these is a system for personal retrieval and reuse called Stuff I’ve Seen (SIS) [9], developed in Microsoft Research Laboratories. One of the main assumptions of [9] is that most of the data that a user wants to retrieve was seen by her in the past. Therefore, SIS records and indexes all the data that the user sees, from any source (e.g., mail, web, word, power-point presentations). The system’s GUI provides a text box for the query and presents a preview of the results’ content, title, author, date, rank, mail-to and source. By default, the results are ordered according to date or “rank,” which is a measure of how relevant the content of the document is to the search query. The user is able to sort the results according to any of the fields displayed (title, author, date, rank, mail to and source) or to filter the results according to the values of these fields. By performing such filtering, the user can get to her desired result by small steps, as suggested in [3]. In this study [9], the fields that were most frequently used for sorting were date, then rank, and only afterwards title, author, etc. Therefore, SIS’s ranking function

is not optimal, since it was used less often than sorting by date. Likewise, SIS does not employ any learning methods to improve the search, and does not allow any degree of personalization in the ranking function.

A different approach for organizing and retrieving personal information was taken in [30]. In this study, a tool was developed to help organize and retrieve data from the user's personal web. The user's personal web consists of files stored locally, web pages saved as bookmarks and other web pages that are linked to the bookmarked pages. The tool unifies the following three different hierarchies into one hierarchy according to the content relationship between the documents: (1) the directory structure of the file system, (2) the bookmark directory, and (3) the hidden hierarchy defined by the direct links between documents, via hyperlinks and citations. The tool presented an alternative interface for saving new documents and retrieving documents. The user saves a document not in a specific location in the file system or as a bookmark, but rather with respect to the existing documents. In other words, the user specifies which documents have content that is related to that of the new document. The tool allows users to retrieve data by presenting the hierarchy so that the user can browse to find related documents in the area of interest. The method for information retrieval, presented in [30], is quite different from previous studies. It has two main drawbacks. First, it does not allow the user to control the file location in the file system. This is a potential problem, since [3] and [28] indicate that users prefer to perform search according to the file location. Second, it does not employ any ranking method and therefore, it exposes the user to a large unordered data set.

## **2.2 Industrial Development**

Recently, several industrial companies have developed desktop-search applications. We briefly survey some of the best-known applications.

Copernic [6] was one of the first companies to develop a desktop search application, allowing users to search virtually any type of file, e.g., pdf files, Word documents, power-point presentations, email, audio, bookmarks, etc. For different types of files (e.g., email, music, etc.), Copernic provides specialized GUIs that allow the results to be sorted by any of the properties of the files. By default, the results in Copernic are sorted by date. The system incorporates no general ranking function whatsoever.

Google [15], famous for its web search engine, developed a desktop version, called Google Desktop. The GUI of Google desktop is similar to that of the web search engine. The results of a search can be sorted either by date or by the rank of the files, a number which is determined by the Google Desktop application. It is not clear which types of factors are considered in ranking documents in query results. PageRank, a ranking method based on link analysis, is widely believed to be the basis of the ranking function for Google's web search engine. However, PageRank is not suitable for desktop search since most of the files in a personal computer do not have hyperlinks. It is interesting to note that by default the results returned by Google Desktop are sorted by date. This seems to indicate that Google is not confident that sorting according to their ranking function typically yields high-quality results.

Many other companies have created their own desktop search engines, e.g., LookOut, Yahoo, HotBot, Blinkx, FileHand. All these search engines support searching over a variety of file types, yet none of these industrial desktop search engines incorporate a learning functionality or personalization (to the best of our knowledge); when ranking is available, it is performed according to a variation of TF-IDF, a criterion used in classic information retrieval [2].

# Chapter 3

## Framework

### 3.1 Queries and Answers

We use  $f$  and  $\mathcal{F}$  to refer to a file and a set of files, respectively. Each file  $f$  is associated with four multi-sets of words as follows.

- $path(f)$  corresponds to the full path of  $f$ , with “\:.\_-” used as delimiters. For example, the full path multi-set of `c:f\g\my_foo.txt` is  $\{\{c, f, g, my, foo, txt\}\}$ .
- $name(f)$  corresponds to the name of  $f$  with “.\_-” used as delimiters. For example, the name multi-set of the file `c:\f\g\my_foo.txt` is  $\{\{my, foo, txt\}\}$ .
- $content(f)$  corresponds to the bag-of-words content of  $f$ . (If  $f$  has no textual content, e.g.,  $f$  is an image or audio file, then  $content(f) = \emptyset$ .)
- $querylog(f)$  is a concatenation of all previously issued queries that resulted in the user choosing (i.e., clicking on)  $f$ .

Let  $q$  be a query, i.e., a sequence of words, and let  $\mathcal{F}$  be a set of files. The *candidate answers*  $\mathbf{Cand}(q, \mathcal{F})$  for  $q$  over  $\mathcal{F}$  is the set of all files  $f \in \mathcal{F}$  such that there is at least one word in common to  $q$  and to one of the four word multi-sets associated with  $f$ . In our system (described in Chapter 4), when the user issues a query  $q$ , and the file system consists of the files  $\mathcal{F}$ , the user is

provided with the list of files  $\mathbf{Cand}(q, \mathcal{F})$ . The user can then either choose a single file among  $\mathbf{Cand}(q, \mathcal{F})$ , or desist from choosing any file. In our analysis we make two important assumptions:

**Unique File Assumption** There is a *unique* file  $f_{q|\mathcal{F}}^* \in \mathcal{F}$  that the user desires when issuing a query  $q$ .

**Recognizability Assumption** If the user issues a query  $q$ , and chooses a file  $f \in \mathbf{Cand}(q, \mathcal{F})$  from the list of returned answers, then we have  $f = f_{q|\mathcal{F}}^*$ .

Note that these assumptions would not normally hold for a standard, open-repository search applications such as Web search. However, they are quite natural in the context of desktop search [13]. In the sequel, we also assume that the user always chooses a file from  $\mathbf{Cand}(q, \mathcal{F})$ ; all other queries are discarded from our analysis anyway.

## 3.2 Features of Interest

In a standard file system, files are naturally associated with numerous attributes such as filename, size, date of creation, location of the file in the hierarchy of directories, etc. A priori, all these attributes bring information that might influence the relative relevance of the files to the user queries. In addition, previous interactions of the users with the desktop search engine are, as well, potential sources of useful information. In our system we decided to exploit a wide palette of such potentially useful sources of information.

Given a query  $q$  and a set of files  $\mathcal{F}$ , each file  $f \in \mathcal{F}$  is associated with a *feature vector*  $\vec{f}(q) \in \mathbb{R}^n$ . Each feature corresponds to one or another property of the file, and, depending on the nature of the feature, its value might be either query-dependent or query-independent. In what follows, the names of the features are written using the small-caps font, e.g., FILETYPE. For ease of presentation, given a feature FEATURE, a query  $q$ , and a file  $f$ , by

$\text{FEATURE}_q(f)$  we denote the value of  $\text{FEATURE}$  in  $\vec{f}(q)$ . (We allow ourselves to omit the subscript  $q$  if the value of the feature in  $\vec{f}(q)$  is query independent.)

**Query Independent Features** We first consider file features whose values are determined independently from the specific user query.

- $\text{SIZE}(f)$  captures the relative size of  $f$ . We set  $\text{SIZE}(f) = 1$  (respectively, 0.8, 0.6, 0.4, 0.2, 0.0) if the size of  $f$  is among top 5 (respectively 10, 20, 50, 75, 100) percents in  $\mathcal{F}$ . We used discrete values later in the learning techniques in order to improve the generalization power since the data volume is not very large.

- $\text{NORMALIZEDSIZE}(f)$  captures the normalized deviation of the physical size of  $f$  from the average size of all other files in  $\mathcal{F}$  that are of the same type as  $f$ . Specifically,  $\text{NORMALIZEDSIZE}(f)$  is set to a  $[0, 1]$ -normalization of

$$\frac{\text{size}(f)}{\text{avg} \{ \text{size}(f') \mid \text{filetype}(f) = \text{filetype}(f') \}}$$

where  $\text{filetype}(f)$  is the file type of  $f$ , e.g, doc, txt, etc.

- If  $\text{level}(f)$  is the distance of file  $f$  from the uppermost directory, then we set  $\text{LEVEL}(f) = 1/\text{level}(f)$ . For example, we have  $\text{LEVEL}(c:\backslash\text{foo.txt}) = 1$ ,  $\text{LEVEL}(c:\backslash\text{f}\backslash\text{g}\backslash\text{foo.txt}) = 1/3$ , etc. This feature is considered following the observation in [3] on storage habits of users with respect to different types of information.

- We have several binary, mutually-exclusive features of the form  $\text{FILETYPEX}(f)$ , where  $X$  is one of the values doc, txt, tex/bib, pdf, ppt, html, java, c, cpp, h, cs, or other. For instance,  $\text{FILETYPEPDF}(f) = 1$  iff  $f$  is a PDF document.

- Using a non-standard feature  $\text{DIRRANK}(f)$  we aim at capturing the importance of the directories in which  $f$  appears (directly or indirectly) to the specific task of desktop search. As mentioned in [3], ephemeral and working

information types are arranged in directories and are re-used. Therefore, we expect that the probability to open file in a specific directory is proportional to the number of files that were previously opened from this directory. Intuitively, we consider the importance of a directory in this context as proportional to the number of times that the correct result for a query appeared either in that directory, or in one of its subdirectories. However, we normalize this importance by “spreading” out the importance of a directory among all the files in the directory (or in its subdirectories). Note that the value of DIRRANK is updated after each new query  $q$  is issued, and the user choice  $f_{q|\mathcal{F}}^*$  among the query results is obtained.

Specifically,  $\text{DIRRANK}(f)$  is specified as follows. At first, before any queries are issued, we set  $\text{DIRRANK}(f) = 0$  for all files  $f \in \mathcal{F}$ . Now, suppose that a query is issued and the desired file  $f_{q|\mathcal{F}}^*$  is found in a directory  $d_q$ . For a file  $f \in \mathcal{F}$ , let  $d_1, \dots, d_k$  be all the directories that are common ancestors of  $d_q$  and  $f$  when viewing the file-system as a tree. For each  $i \leq k$ , let  $n_i$  be the number of files that are in  $d_i$ , or in any of its subdirectories, i.e.,  $n_i$  is the number of file nodes in the subtree of the file-system rooted at  $d_i$ . Then, we increment DIRRANK of  $f$  by  $\sum_{i=1}^k 1/n_i$ . Note that the value that is added to  $\text{DIRRANK}(f)$  is proportional to the number of directories which are common to  $d_q$  and  $f$ , and is inversely proportional to the number of files in each of the common directories.<sup>1</sup>

**Query Dependent Features** In addition to the query-independent features, some features of a file depend either on the date in which the query was issued, or on the actual content of the query.

First, recall that each file  $f$  is associated with four word multi-sets  $\text{path}(f)$ ,  $\text{name}(f)$ ,  $\text{content}(f)$ , and  $\text{querylog}(f)$ . These sets naturally give rise to four

---

<sup>1</sup>When we performed our experimentation, we also considered two other variants of computing DIRRANK. Rather similar results were obtained, and thus, we omit these details in the thesis.

respective features PATH, NAME, CONTENT, QUERYLOG, where  $\text{PATH}_q(f)$  correlates with the cosine distance between the tf.idf vectors of  $\text{path}(f)$  and  $q$ ; the values of  $\text{NAME}_q(f)$ ,  $\text{CONTENT}_q(f)$ , and  $\text{QUERYLOG}_q(f)$  are similarly determined.<sup>2</sup>

**Time Dependent Features** Next, we adopt and use three features whose values depend on the date in which the query is issued, namely, ACCESSDATE, UPDATEDATE, and CREATEDATE. The value of each of these features is determined in exactly the same fashion, and thus we explicitly explain only for ACCESSDATE. Consider a file  $f$ , which was last accessed on a date  $t$ . Suppose that the query  $q$  was issued on date  $t_q$ . Then, the value of  $\text{ACCESSDATE}_q(f)$  is set to

- 1, if  $t = t_q$ ,
- 0.8, if  $t$  is within the last three days of  $t_q$ ,
- 0.6, if  $t$  is within the last week of  $t_q$ ,
- 0.4, if  $t$  is within the last month of  $t_q$ ,
- 0.2, if  $t$  is within the last two months of  $t_q$ ,
- 0, otherwise.

The value of  $\text{ACCESSDATE}_q(f)$  decreases as the distance in time between  $t$  and  $t_q$  grows. This setting is consistent with the observations reported in [9, 3].

---

<sup>2</sup>We note that tf.idf is the name given to a family of scoring methods in which a higher weight is given to word sequences that contain more occurrences of rare words in  $q$ . To determine how rare a particular word is for one of the features (e.g., PATH), we take as our corpus the set of all word multisets of files in the file-system that are associated with the given feature (e.g., the set of all full path multisets). See <http://lucene.apache.org/java/docs/scoring.html> for the exact version of the tf.idf formula used.

### 3.3 Evaluation Criteria

The goal of this thesis is to find effective ranking methods for desktop search. Towards this aim we must be able to determine the (relative) effectiveness of a particular ranking method. We introduce our evaluation criteria here.

Consider a query  $q$  and a set of files  $\mathcal{F}$ . Recall that our system retrieves the files  $\mathbf{Cand}(q, \mathcal{F})$ , when  $q$  is issued over  $\mathcal{F}$ . A *ranking method*  $\tau : \mathbb{R}^n \rightarrow \mathbb{R}$  is a function that associates feature vectors  $\vec{f}(q), f \in \mathbf{Cand}(q, \mathcal{F})$  with a numeric value, denoted  $\tau_q(f)$ . Note that each feature by itself can be considered as such a ranking method, and additional ranking methods will be discussed later on.

Now, suppose that we are given a query  $q$ , set of files  $\mathcal{F}$  and a ranking method  $\tau$ . When displaying  $\mathbf{Cand}(q, \mathcal{F})$  to the user according to  $\tau$ , the files in  $\mathbf{Cand}(q, \mathcal{F})$  are listed in decreasing order of  $\tau_q(f)$ , with the ties being arbitrarily broken. Hence, we can associate each file  $f \in \mathbf{Cand}(q, \mathcal{F})$  with its *expected placement* according to  $\tau$ . Let  $n_{\text{gt}}$  be the number of files with a higher ranking than  $f$ , and let  $n_{\text{eq}}$  be the number of other files with the same ranking as  $f$ , i.e.,

$$\begin{aligned} n_{\text{gt}} &= |\{f' \in \mathbf{Cand}(q, \mathcal{F}) \mid \tau_q(f) > \tau_q(f')\}| \\ n_{\text{eq}} &= |\{f' \in \mathbf{Cand}(q, \mathcal{F}) \mid \tau_q(f) = \tau_q(f') \text{ and } f \neq f'\}| \end{aligned}$$

Then, the *expected placement* of  $f$ , according to  $\tau$  is

$$\text{Exp}(\tau, f \mid q, \mathcal{F}) \stackrel{\text{def}}{=} 1 + n_{\text{gt}} + \frac{1}{2} \cdot n_{\text{eq}},$$

where the value 1 is added so that the expected placement will be a number that is greater than or equal to 1.

To determine the *effectiveness* of a ranking method  $\tau$ , we consider the expected placement of  $f_{q_i|\mathcal{F}_i}^*$  (or  $f_i^*$ , for short) with respect to a set of query/fileset pairs

$$\mathcal{S} = \{(q_1, \mathcal{F}_1), \dots, (q_m, \mathcal{F}_m)\}.$$

(Note that the file-system changes over time. Hence, each query is evaluated with respect to a specific instance of the file system.) We define the effectiveness of  $\tau$  by averaging the expected placement of  $f_i^*$  over the pairs  $(q_i, \mathcal{F}_i)$ , i.e.,

$$\text{Score}(\tau, \mathcal{S}) \stackrel{\text{def}}{=} \text{avg}_{(q_i, \mathcal{F}_i) \in \mathcal{S}} \{\text{Exp}(\tau, f_i^* \mid q_i, \mathcal{F}_i)\}. \quad (3.1)$$

Intuitively, the overall effectiveness of  $\tau$  on  $\mathcal{S}$  is inversely proportional to score  $\text{Score}(\tau, \mathcal{S})$ ; a lower score is better as it indicates an expected placement that is closer to 1. This measure corresponds to the *mean reciprocal rank* measure [29] (popular in evaluating question-answering systems), specialized to our assumption of uniqueness of  $f_{q|\mathcal{F}}^*$ .

Other measures of effectiveness might also be of interest in the context of desktop search. As we already discussed, the users in our case are likely to have relatively high expectations from the performance of a desktop search engine. Thus, if  $f_{q|\mathcal{F}}^*$  does not appear within some short top- $k$  prefix of the ordered list  $\mathbf{Cand}(q, \mathcal{F})$ , it is likely not to be discovered by the user at all. To capture this expected property of desktop search applications, given a series of query/fileset pairs  $\mathcal{S}$ , we define the *effectiveness of  $\tau$  at  $k$*  as

$$\text{TopScore}_k(\tau, \mathcal{S}) \stackrel{\text{def}}{=} \frac{100}{|\mathcal{S}^{[>k]}|} \sum_{(q_i, \mathcal{F}_i) \in \mathcal{S}^{[>k]}} \delta(\text{Exp}(\tau, f_i^* \mid q_i, \mathcal{F}_i) \leq k), \quad (3.2)$$

where

$$\mathcal{S}^{[>k]} = \{(q_i, \mathcal{F}_i) \in \mathcal{S} \mid k < |\mathbf{Cand}(q_i, \mathcal{F}_i)|\}$$

and  $\delta$  is the Kronecker step function (which returns 1 if the condition holds and 0 otherwise). A higher value for  $\text{TopScore}_k(\tau, \mathcal{S})$  is preferable as it indicates a greater percentage of highly-ranked query results. Note that  $\mathcal{S}^{[>k]}$  is used to filter out of the set  $\mathcal{S}$  any queries which have at most  $k$  results (and hence,  $f_{q|\mathcal{F}}^*$  must be in the top- $k$ ). Note also that Eq. 3.2 basically corresponds to specializing the standard *recall-at- $k$*  evaluation measure used in IR [2] to our assumption of uniqueness of  $f_{q|\mathcal{F}}^*$ .

## Chapter 4

# Experimental Setup and Data

## Gathering

Our desktop search engine was built as an extension of the open-source search engine Lucene.<sup>1</sup> We implemented the following four main functionalities:

- Users can choose folders of interest and request to *create an index* for these folders. While indexing, our engine stores information about each file, such as its full path, name, create date, etc.
- The user can request the system to *update the index*. By default, the index is updated daily, but users can arrange for a different desired updating schedule.
- *Queries* can be issued to our search engine. Each query constitutes a list of words, possibly with wildcards. The file name, path, type, last access date, last update date and create date are displayed for each file in the result. The system displays the results set  $\mathbf{Cand}(q, \mathcal{F})$  sorted by default in the decreasing order of their last access dates. (See Figure 4.1 for a screen shot of a query result.) The user then selects a single file  $f$  from the query results by clicking on the corresponding file name, at which point the file is opened by the appropriate application. Clicking on a file name also triggered

---

<sup>1</sup><http://lucene.apache.org/java/docs/>

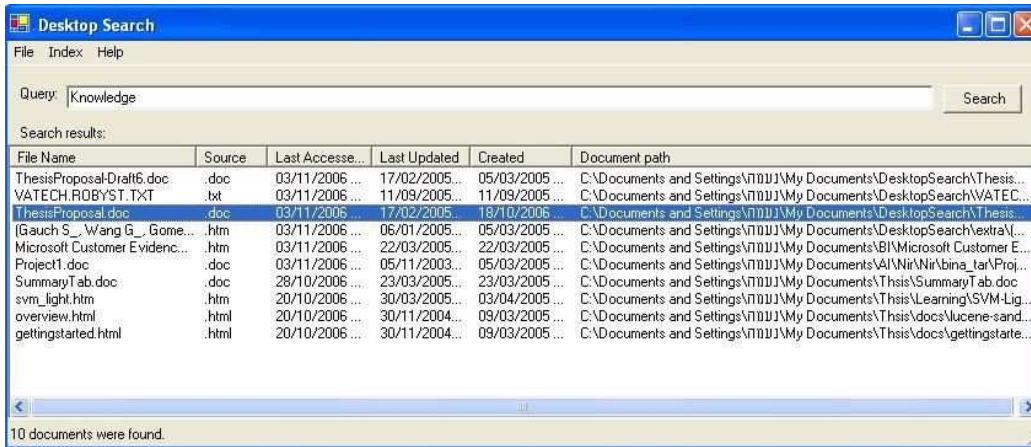


Figure 4.1: A screenshot of our desktop search engine.

an automatic update of the last access date of  $f$ , the values for the feature DIRRANK, as well as the content of the multi-set  $querylog(f)$ .

We supplied our users the ability to sort the results according to any of the displayed fields in order to allow the user to find  $f_{q|\mathcal{F}}^*$  easily. A priori, we did not know which ranking methods would be most effective, and our primary objective was in fact to shed some light on this question. Thus, the effectiveness of various ranking methods was analyzed “post-mortem,” on the basis of the collected log data.

- One of the most important features of our engine is the extensive *logging* that it performs. Specifically, after each query is issued, and a file is chosen, the system stores the list of all files displayed to the user, as well as the file that was chosen, with all of their features. This extensive logging allowed us to perform a post-mortem analysis of ranking mechanisms.

The desktop search engine was distributed to 19 volunteers, 10 of whom were male, and 9 of whom were female. Five of our volunteers were undergraduate students, 8 were graduate students and 6 work in the industry, e.g., as engineers. All were non-native English speakers with a good command of the English language. The users were in the age range of 21–35. The queries

Num. Results	Num. Queries	Notation
1	188	
2–50	916	$\mathcal{S}_{2-50}$
> 50	115	$\mathcal{S}_{>50}$

Table 4.1: Frequency of result size.

were typically performed over a period of several months. Each volunteer performed between 4 and 443 queries. In total 1219 queries were issued (with an average of 64.2 queries per user). The queries varied in their number of results. On average, there were 23.74 results per query, with a very large variance of 3482.99. See Table 4.1 for a more detailed view of the number of results per query. We chose the cutoff point (50) to be the number of results that the user was able to view in full screen without scrolling (in our system).

At the end of the evaluation period, all log files were collected from the users for analysis. We will use  $\mathcal{S}_{\text{all}}$  to denote the set of all pairs  $(q, \mathcal{F})$  that were issued by our users for which  $\mathbf{Cand}(q, \mathcal{F}) > 1$ , that is  $\mathcal{S}_{\text{all}} = \mathcal{S}_{2-50} \cup \mathcal{S}_{>50}$ , where  $\mathcal{S}_{2-50}$  and  $\mathcal{S}_{>50}$  are as defined in Table 4.1. The sets  $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$ , and  $\mathcal{S}_{>50}$  provide the basis for our analysis of different ranking methods. We note that dividing the  $(q, \mathcal{F})$  pairs according to the size of  $\mathbf{Cand}(q, \mathcal{F})$  will prove useful, since the size of the output is an indicator as to the nature of the query. This issue is discussed in more detail in Chapter 5.

## Chapter 5

# Basic Ranking Schemes

Typically, desktop search engines allow the user to sort the search results according to various (query dependent and independent) properties of the files, such as name, last access date, etc. Hence, we first consider the effectiveness of basic ranking methods, each relying upon a single feature of the files.<sup>1</sup> (These ranking methods correspond naturally to different ways that the user can sort data.) Table 5.1 summarizes  $\text{Score}(\tau, \cdot)$  for different such single-feature-based ranking methods with respect to  $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$ , and  $\mathcal{S}_{>50}$ , ordered in the decreasing overall effectiveness, i.e., increasing value of  $\text{Score}(\tau, \cdot)$ . As a reference point, *Random* gives the effectiveness of a random ordering of results.

Let us first consider the results with respect to all search sessions  $\mathcal{S}_{\text{all}}$ . Consistently with the experience reported in [9], the date feature `UPDATEDATE` (as well as other date-related features) has been found clearly more effective than the relevance-based feature `CONTENT`. What was surprising, however, is that `CONTENT` appears to be one of the poorest ranking features—not only most other features have been found more effective than `CONTENT` (with `NAME` being the leader among the date-independent features), but even random ordering appeared to be more effective on average than the `CONTENT`-based

---

<sup>1</sup>For the features `ACCESSDATE`, `UPDATEDATE`, `CREATEDATE`, `SIZE` and `NORMALIZEDSIZE`, we actually ranked the files according to the real values of these features, and not according to the coarser values of 1, 0.8, 0.6, etc., as described in Chapter 3. This provided us with a finer distinction between files, for ranking purposes.

FEATURE	Score( $\tau, \mathcal{S}_{\text{all}}$ )	FEATURE	Score( $\tau, \mathcal{S}_{2-50}$ )	FEATURE	Score( $\tau, \mathcal{S}_{>50}$ )
ACCESSDATE	9.85	NAME	5.66	ACCESSDATE	36.20
UPDATEDATE	10.18	PATH	6.32	UPDATEDATE	40.13
CREATEDATE	11.51	UPDATEDATE	6.43	CREATEDATE	49.07
NAME	11.53	ACCESSDATE	6.54	NAME	58.34
DIRRANK	12.56	QUERYLOG	6.59	DIRRANK	58.40
PATH	12.60	CONTENT	6.75	SIZE	61.64
SIZE	12.96	CREATEDATE	6.80	PATH	62.69
QUERYLOG	13.5	DIRRANK	6.80	QUERYLOG	68.63
<i>Random</i>	<i>14.45</i>	SIZE	6.85	<i>Random</i>	<i>73.87</i>
CONTENT	15.43	<i>Random</i>	<i>6.99</i>	NORMALIZEDSIZE	83.86
NORMALIZEDSIZE	16.24	NORMALIZEDSIZE	7.75	CONTENT	84.57
LEVEL	17.90	LEVEL	7.80	LEVEL	98.36

Table 5.1: Effectiveness of basic, feature-based ranking methods.

FEATURE	TopScore $_k(\tau, \mathcal{S}_{\text{all}})$				TopScore $_k(\tau, \mathcal{S}_{2-50})$				TopScore $_k(\tau, \mathcal{S}_{>50})$			
	$k=1$	$k=2$	$k=5$	$k=10$	$k=1$	$k=2$	$k=5$	$k=10$	$k=1$	$k=2$	$k=5$	$k=10$
UPDATEDATE	<b>21.6</b>	<b>31.5</b>	40.7	<b>52.5</b>	<b>23.4</b>	<b>34.6</b>	44.8	57.8	<b>7.8</b>	9.6	20.0	33.9
ACCESSDATE	19.2	30.0	40.9	52.3	19.2	29.0	40.9	52.3	7.0	<b>12.2</b>	<b>22.6</b>	<b>36.5</b>
NAME	16.5	29.2	<b>41.8</b>	51.7	18.1	32.5	<b>47.3</b>	<b>60.5</b>	3.5	6.1	13.9	20.9
CREATEDATE	17.7	27.3	37.6	48.5	19.2	30.0	42.2	54.1	5.2	8.7	14.8	28.7
DIRRANK	5.4	18.0	32.2	47.5	6.1	20.4	36.5	55.3	0.0	0.9	10.4	20.0
PATH	6.9	19.1	33.8	45.9	7.2	20.7	38.2	54.1	4.3	7.8	11.3	17.4
SIZE	16.1	26.6	39.2	42.3	17.8	29.6	44.8	50.6	2.6	5.2	11.3	13.0
QUERYLOG	8.1	18.2	28.9	40.5	8.7	20.0	33.0	49.9	3.5	5.2	7.8	7.8
CONTENT	13.2	23.6	36.9	45.0	14.6	26.4	43.0	55.1	1.7	4.3	6.1	9.6
<i>Random</i>	<i>0.0</i>	<i>10.6</i>	<i>22.6</i>	<i>36.3</i>	<i>0.0</i>	<i>12.1</i>	<i>27.0</i>	<i>46.7</i>	<i>0.0</i>	<i>0.0</i>	<i>0.0</i>	<i>0.0</i>
NORMALIZEDSIZE	14.5	21.6	29.5	36.3	16.0	24.3	34.4	44.4	1.7	2.6	4.3	7.8
LEVEL	4.6	15.7	26.7	34.9	5.1	17.8	31.5	43.2	0.0	0.9	2.6	6.1

Table 5.2: Same as Table 5.1, but for effectiveness at  $k \in \{1, 2, 5, 10\}$ .

ranking (with this dominance being statistically significant<sup>2</sup> on  $\mathcal{S}_{>50}$ , but not on  $\mathcal{S}_{\text{all}}$  and  $\mathcal{S}_{2-50}$ ). Such a poor performance of the CONTENT feature is at least partially due to the fact that many times users search for files with  $\text{content}(f_{q|\mathcal{F}}^*) = \emptyset$ , e.g., image, audio, or movie files. However, we have verified that, even when only considering queries for which the desired file has textual content, ranking on the basis of CONTENT continues to perform similarly.

<sup>2</sup>Statistically significant differences in performance are determined using the two-sided Wilcoxon test at the 95% confidence level.

The poor performance of CONTENT is seemingly inconsistent with the observation in [9], that UPDATEDATE and (a closely related variant of) CONTENT were observed to be by far the most popular sorting schemes. However, this observation does not necessarily contradict the results in Table 5.1. A priori, even for us it was not intuitive why textual connection between the query and the file’s pathname was a more effective search/ranking criterion than textual connection between the query and the content of the file.<sup>3</sup> We may conclude that *despite the fact that desktop search typically targets a concrete data object, the user’s intuition about the relative effectiveness of various ranking methods does not necessarily correspond to their effectiveness in practice.* In fact, the analysis in [9] indirectly supports this hypothesis by indicating that, given a specific search result  $\mathbf{Cand}(q, \mathcal{F})$ , users often order and re-order results according to various features (or ranking schemes) in order to find the desired file. Note that this only stresses the importance of automatically selecting the primary ranking scheme.

Now, we consider the results with respect to the search sessions  $\mathcal{S}_{2-50}$  and  $\mathcal{S}_{>50}$ , and compare them with these for the whole set  $\mathcal{S}_{\text{all}}$ . While the relative effectiveness of ranking methods on  $\mathcal{S}_{>50}$  was almost the same as for  $\mathcal{S}_{\text{all}}$ , the picture is very different in case of  $\mathcal{S}_{2-50}$ . Whereas in general, date-related features appear to be more effective, search sessions  $\mathcal{S}_{2-50}$  with small sets of results are better ranked by the feature NAME that captures the textual similarity between the query and the name of the file (and the difference between the effectiveness of NAME and UPDATEDATE on  $\mathcal{S}_{2-50}$  was statistically significant).

We conjecture that the difference in relative effectiveness of the features between  $\mathcal{S}_{2-50}$  and  $\mathcal{S}_{>50}$  has an intuitive explanation. When querying in a search

---

<sup>3</sup>Some intuitions for why NAME and even PATH were relatively effective can be found in [3] where it was observed that “file naming was given careful attention by the users.” This, however, does not explain why CONTENT was less effective than NAME, PATH, or any other feature.

system of any type, users attempt to provide a query that is *selective*, i.e., that can effectively separate between the result of interest and the remainder of available objects (e.g., files). This perspective on the query formulation process is especially intuitive in the context of desktop search, since the users are presumed to (approximately) know the feature values, e.g., the name or path, of the desired file. Having this perspective in mind, the selectiveness of the content of a query  $q$  with respect to the user's search desires and the given set of files  $\mathcal{F}$ , is inversely proportional to the size of the result set  $\mathbf{Cand}(q, \mathcal{F})$ . If so, *then the larger the resulting size of  $\mathbf{Cand}(q, \mathcal{F})$ , the less selective the query is, and thus, a ranking method that is independent of the content of the query is more likely to be effective.* Date-related features appear to be good candidates for such a query-content independent ranking method as we often search our file system for files that recently were in use. We revisit the issue of selectiveness of features in Chapter 7.

Table 5.2 summarizes the effectiveness of the (best) single-feature-based ranking methods for  $k \in \{1, 2, 5, 10\}$ . In each column, the score of the most effective feature is emphasized. There is no clear winner for any of  $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$  or  $\mathcal{S}_{>50}$ . Thus, based on our results, no recommendation can be made as to a ranking method of choice, when we wish to optimize for the top- $k$  results. Interestingly, we will show in the upcoming chapters that the picture changes dramatically once more sophisticated ranking methods are considered.

## Chapter 6

# Learning to Rank

As we already mentioned, one of the components of our desktop search system is the logger that stores all the data on the past search sessions of the user. Specifically, for each search session  $(q, \mathcal{F})$ , the logger stores information about the result set  $\mathbf{Cand}(q, \mathcal{F})$  and the selected file  $f_{q|\mathcal{F}}^* \in \mathbf{Cand}(q, \mathcal{F})$ . In principle, this information can be used for *learning* a better ranking method. Probably the simplest such learning scheme would go along the lines of our analysis in Chapter 5 by selecting a primary ranking method that would optimize effectiveness measures such as those in Eq. 3.1-3.2, possibly *given* the size of  $\mathbf{Cand}(q, \mathcal{F})$  and/or some other helpful indicators. However, here we show that (even just slightly) more sophisticated learning schemes can provide us with more effective ranking methods for desktop search.

### 6.1 Learning a Ranking Function

Suppose we are given some previous user search sessions,  $(q_1, \mathcal{F}_1), \dots, (q_m, \mathcal{F}_m)$ , or, more specifically, with the corresponding logged information

$$\mathcal{L} = \{\langle f_1^*, \mathbf{Cand}(q_1, \mathcal{F}_1) \rangle, \dots, \langle f_m^*, \mathbf{Cand}(q_m, \mathcal{F}_m) \rangle\}. \quad (6.1)$$

One way to use the information about the past user searches would be (i) adopting a standard machine learning approach of binary classification with

two classes “relevant” and “non-relevant”, and (ii) training a binary classifier over “relevant” training examples  $\bigcup_{i=1}^m \{f_i^*\}$  and “non-relevant” training examples  $\bigcup_{i=1}^m \mathbf{Cand}(q_i, \mathcal{F}_i) \setminus \{f_i^*\}$ . However, as observed in [19, 21], in applications like desktop search this approach has several drawbacks. First, due to a strong majority of “non-relevant” examples, a classifier will typically optimize the predictive classification accuracy by always responding “non-relevant.” Second, and even more importantly, while we know that  $f_i^*$  is explicitly selected by the user as the target of her search, if the content of the file system were to be different, the user might use the same query  $q_i$  to search for a different file. The (unique) “relevance” of  $f_i^*$  to  $q_i$  is not absolute, but relative to the content of  $\mathbf{Cand}(q_i, \mathcal{F}_i)$ . In other words, if  $\succ_i$  stands for a “more relevant to  $q_i$ ” binary relation over the file universe, then the only information that the logger  $\mathcal{L}$  really conveys to us is that

$$\forall 1 \leq i \leq m, \forall f \in \mathbf{Cand}(q_i, \mathcal{F}_i) \setminus \{f_i^*\} : f_i^* \succ_i f \quad (6.2)$$

Considering Eq. 6.2, at first view it seems that learning a classifier cannot address our needs. However, this is not exactly so. Partial rankings such as that of Eq. 6.2 have been used in machine learning community to train binary classifiers over *pairs* of objects, with the target being not a class label, but a binary relation over the objects [5, 16, 21]. Specifically, such learners select, from a family of *ranking functions* over (a representation of) the objects, a ranking function  $\varphi$  that minimizes the classification (= ranking) error on the training data.

Following a variation of the framework suggested in [19], let us consider the class of binary relations  $\succ_{\vec{w}}$  over triplets  $(f; q, \mathcal{F})$  that is defined by the set of linear functions  $\varphi_{\vec{w}} : \mathbb{R}^n \rightarrow \mathbb{R}$  over our  $n$  file features as

$$(f; q, \mathcal{F}) \succ_{\vec{w}} (f'; q', \mathcal{F}') \text{ iff } \varphi_{\vec{w}}(\vec{f}(q)) > \varphi_{\vec{w}}(\vec{f}'(q'))$$

where  $\vec{f}(q)$  is our feature vector of the file  $f$  with respect to query  $q$  and fileset  $\mathcal{F}$ . This can be written equivalently as

$$(f; q, \mathcal{F}) \succ_{\vec{w}} (f'; q', \mathcal{F}') \text{ iff } \vec{w} \cdot \vec{f}(q) > \vec{w} \cdot \vec{f}'(q'),$$

where  $\vec{w}$  is the vector of  $\varphi_{\vec{w}}$ 's coefficients and  $\cdot$  is dot product. Finding such a function  $\varphi_{\vec{w}}$  that minimizes classification error on our training logger data (6.2) is equivalent to finding a weight vector  $\vec{w}$  so that the maximum number of the inequalities in Eq. 6.3 below are satisfied.

$$\begin{aligned} \forall 1 \leq i \leq m, \forall f \in \mathbf{Cand}(q_i, \mathcal{F}_i) \setminus \{f_i^*\} : \\ \vec{w} \cdot \vec{f}_i^*(q_i) > \vec{w} \cdot \vec{f}(q_i) \end{aligned} \quad (6.3)$$

This optimization problem, unfortunately, is known to be NP-hard [19, 17]. However, it is possible to approximate its solution by introducing a non-negative slack variable  $\xi$  for each linear constraint in (6.3), and then minimizing the sum of these slack variables. On the other hand, as suggested by theory and practice of classification Support Vector Machines (SVM) [7], adding regularization for margin maximization to the objective is likely to increase the generalization power of the learned function  $\varphi_{\vec{w}}$  well beyond the training data. A machine with small margin and therefore low generalization power might incorrectly classify queries that do not resemble the training queries, where machine with large margin and therefore high generalization power should succeed better with those queries.

This setting results in the optimization problem (6.4) below.

$$\begin{aligned} \text{minimize: } & \frac{1}{2} \vec{w} \cdot \vec{w} + C \sum \xi_{i,f} \\ \text{subject to:} & \\ & \forall 1 \leq i \leq m, \forall f \in \mathbf{Cand}(q_i, \mathcal{F}_i) \setminus \{f_i^*\} : \\ & \vec{w} \cdot \vec{f}_i^*(q_i) > \vec{w} \cdot \vec{f}(q_i) + 1 - \xi_{i,f} \\ & \forall \xi_{i,f} : \xi_{i,f} \geq 0 \end{aligned} \quad (6.4)$$

where  $C$  is a parameter that allows trading-off margin size, i.e.,  $(\frac{1}{2}\vec{w}\cdot\vec{w})$  against training error, i.e.,  $(\sum \xi_{i,f})$ . The optimization problem (6.4) is strictly convex, has no local minima, and is equivalent to that of classification SVM on *difference vectors*  $\vec{f}_i^*(q_i) - \vec{f}(q_i)$ . Thus, it can be efficiently solved using one of the standard tools for classification SVM (e.g., see [18]). Having generated the weight vector  $\vec{w}$  optimizing (6.4), for each new search session  $(q_{\text{new}}, \mathcal{F}_{\text{new}})$ , the search results  $f \in \mathbf{Cand}(q_{\text{new}}, \mathcal{F}_{\text{new}})$  are then ranked in the decreasing order of  $\vec{w} \cdot \vec{f}(q_{\text{new}})$ .

For our analysis we have implemented and evaluated effectiveness of the learning scheme based on the ranking SVM problem formulation as in Eq. 6.4. The results of this analysis are discussed in Chapter 6.3, after we present yet another type of learning scheme.

## 6.2 Learning A Simple Aggregation

Learning a ranking function using ranking SVM on the basis of a logger  $\mathcal{L}$  as in Eq. 6.1 can be done in time linear in the number of file features  $n$ , and polynomial in  $\sum_{i=1}^m |\mathbf{Cand}(q_i, \mathcal{F}_i)|$  [4]. As the amount of the logged search sessions grows, the latter complexity factor significantly slows down the learner. This slowdown is particularly significant when queries result in a large number of candidate answers. And while learning can be performed offline (at the otherwise idle time of the machine), at some point the system will unavoidably have to cut down the amount of available training data, possibly using only some chronological suffix of the logged data.

To what degree the latter issue is a limitation is not entirely clear. In fact, our evaluation provides some evidence showing that a reasonably small amount of training data already allows learning good ranking functions using the ranking SVM scheme. In any event, we decided to check whether some easy-to-learn compositions of the basic, feature-based ranking schemes can

already buy us improvement in ranking efficiency. For this, we tried two simple ranking schemes: LEXORD and USERBEST.

The first scheme, called LEXORD (for “lexicographic order”), requires learning only the relative efficiency of various basic ranking schemes as in Chapter 5 (that is, learning only information as in Tables 5.1-5.2), and it is defined as follows:

- (a) Let a logger  $\mathcal{L}$  contain information about some past search sessions  $\mathcal{S} = \{(q_1, \mathcal{F}_1), \dots, (q_m, \mathcal{F}_m)\}$  of the user.
- (b) Let  $\text{FEATURE}^{(1)}, \dots, \text{FEATURE}^{(n)}$  be our file features (described in Chapter 3), *numbered in the decreasing order of their efficiency on  $\mathcal{S}$* . That is, if  $\tau_i$  is the ranking method implicitly provided by  $\text{FEATURE}^{(i)}$ , then, for  $1 \leq i < m$ , we have  $\text{Score}(\tau_i, \mathcal{S}) \leq \text{Score}(\tau_{i+1}, \mathcal{S})$ .
- (c) Given a new search session  $(q_{\text{new}}, \mathcal{F}_{\text{new}})$ , rank the result set  $\mathbf{Cand}(q_{\text{new}}, \mathcal{F}_{\text{new}})$  such that, for any two files  $f, f' \in \mathbf{Cand}(q_{\text{new}}, \mathcal{F}_{\text{new}})$ , we have  $f$  ranked higher than  $f'$  if there is an  $i$  such that

$$\text{FEATURE}_{q_{\text{new}}}^{(i)}(f) > \text{FEATURE}_{q_{\text{new}}}^{(i)}(f'),$$

and for all  $j < i$ ,

$$\text{FEATURE}_{q_{\text{new}}}^{(j)}(f) = \text{FEATURE}_{q_{\text{new}}}^{(j)}(f').$$

In other words, LEXORD corresponds to a lexicographic aggregation of feature-based rankings based on their relative efficiency when used in isolation.

The second scheme, called USERBEST, is even simpler. It operates the same as LEXORD. However step (b) is only applied for  $\text{FEATURE}^{(1)}$ . In other words, for each user we choose the feature that was most effective for the previous training queries and apply it on the new query results  $\mathcal{F}_{\text{new}}$ .

FEATURE	Score( $\tau, \mathcal{S}_{\text{all}}$ )	FEATURE	Score( $\tau, \mathcal{S}_{2-50}$ )	FEATURE	Score( $\tau, \mathcal{S}_{>50}$ )
<b>SVM-90</b>	7.84	<b>SVM-90</b>	4.72	<b>SVM-90</b>	32.65
<b>SVM-75</b>	8.15	<b>SVM-75</b>	4.93	<b>SVM-75</b>	33.74
<b>LexOrd</b>	9.19	<b>LexOrd</b>	5.15	ACCESSDATE	36.20
<b>UserBest</b>	9.64	NAME	5.66	UPDATEDATE	40.13
ACCESSDATE	9.85	<b>UserBest</b>	5.73	<b>UserBest</b>	40.80
UPDATEDATE	10.18	PATH	6.32	<b>LexOrd</b>	41.39
CREATEDATE	11.51	UPDATEDATE	6.43	CREATEDATE	49.07
NAME	11.53	ACCESSDATE	6.54	NAME	58.34
DIRRANK	12.56	QUERYLOG	6.59	DIRRANK	58.40
PATH	12.60	CONTENT	6.75	SIZE	61.64
SIZE	12.96	CREATEDATE	6.80	PATH	62.69
QUERYLOG	13.5	DIRRANK	6.80	QUERYLOG	68.63
<i>Random</i>	<i>14.45</i>	SIZE	6.85	<i>Random</i>	<i>73.87</i>
CONTENT	15.43	<i>Random</i>	<i>6.99</i>	NORMALIZEDSIZE	83.86
NORMALIZEDSIZE	16.24	NORMALIZEDSIZE	7.75	CONTENT	84.57
LEVEL	17.90	LEVEL	7.80	LEVEL	98.36

Table 6.1: Relative effectiveness of the learned ranking functions and lexicographic aggregations of the basic, feature-based rankings.

### 6.3 Evaluation

All learning schemes have been evaluated on the log datasets  $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$ , and  $\mathcal{S}_{>50}$ . For each user, the training data consisted of four randomly chosen subsets of 75% of its log data, with the remaining 25% used as the test set. The results were averaged over these four samples and over all users. Learning the LEXORD and USERBEST ranking schemes is straightforward, and does not require any further specification. To learn linear ranking functions, we used the *SVM<sup>light</sup>* software [18] with the ranking SVM setup.<sup>1</sup> No feature selection, parameter optimization, or other tuning was done; all the parameters of the learner have been set to their default values. In addition to 75/25 training/testing data partition, we also learned ranking functions based on the (otherwise similar) 90/10 data partition setup. In what follows, the corresponding results are marked with SVM-75 and SVM-90, respectively.

Tables 6.1 and 6.2 summarize the effectiveness and effectiveness at  $k$  of

<sup>1</sup><http://svmlight.joachims.org>

FEATURE	TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{\text{all}}$ )				TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{2-50}$ )				TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{>50}$ )			
	$k = 1$	$k = 2$	$k = 5$	$k = 10$	$k = 1$	$k = 2$	$k = 5$	$k = 10$	$k = 1$	$k = 2$	$k = 5$	$k = 10$
<b>SVM-90</b>	34.4	45.2	56.3	64.9	36.6	48.4	61.1	71.7	17.4	22.6	32.2	40.9
<b>SVM-75</b>	32.8	42.8	55.0	63.5	34.5	45.8	59.6	70.5	19.1	21.7	32.2	39.1
<b>LexOrd</b>	34.0	41.8	52.2	62.5	37.2	45.8	60.0	68.5	8.7	13.9	27.8	41.7
<b>UserBest</b>	24.3	34.6	46.7	57.1	26.6	38.1	51.3	62.0	6.0	10.4	23.5	40.0
UPDATEDATE	21.6	31.5	40.7	52.5	23.4	34.6	44.8	57.8	7.8	9.6	20.0	33.9
ACCESSDATE	19.2	30.0	40.9	52.3	19.2	29.0	40.9	52.3	7.0	12.2	22.6	36.5
NAME	16.5	29.2	41.8	51.7	18.1	32.5	47.3	60.5	3.5	6.1	13.9	20.9
CREATEDATE	17.7	27.3	37.6	48.5	19.2	30.0	42.2	54.1	5.2	8.7	14.8	28.7
DIRRANK	5.4	18.0	32.2	47.5	6.1	20.4	36.5	55.3	0.0	0.9	10.4	20.0
PATH	6.9	19.1	33.8	45.9	7.2	20.7	38.2	54.1	4.3	7.8	11.3	17.4
SIZE	16.1	26.6	39.2	42.3	17.8	29.6	44.8	50.6	2.6	5.2	11.3	13.0
QUERYLOG	8.1	18.2	28.9	40.5	8.7	20.0	33.0	49.9	3.5	5.2	7.8	7.8
CONTENT	13.2	23.6	36.9	45.0	14.6	26.4	43.0	55.1	1.7	4.3	6.1	9.6
Random	0.0	10.6	22.6	36.3	0.0	12.1	27.0	46.7	0.0	0.0	0.0	0.0
NORMALIZEDSIZE	14.5	21.6	29.5	36.3	16.0	24.3	34.4	44.4	1.7	2.6	4.3	7.8
LEVEL	4.6	15.7	26.7	34.9	5.1	17.8	31.5	43.2	0.0	0.9	2.6	6.1

Table 6.2: Same as Table 6.1, but for effectiveness at  $k \in \{1, 2, 5, 10\}$ .

LEXORD, USERBEST, SVM-75, and SVM-90, and compare them to the basic, feature-based ranking methods (taken from Tables 5.1-5.2.) *The results appear to sharply vote for learning rankings that are based on (some or another) personalized aggregation of different features of the files with respect to the query.* Considering first the general effectiveness measured by  $\text{Score}(\tau, \cdot)$  (Table 6.1):

- (1) Both SVM-75 and SVM-90 ended up clear winners for  $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$ , and  $\mathcal{S}_{>50}$ . The difference between SVM-90 and the third most effective ranking method in each category was statistically significant. The similar difference for SVM-75 was also statistically significant, except for the  $\mathcal{S}_{2-50}$  category (where the third most effective ranking method was LEXORD.)
- (2) For  $\mathcal{S}_{\text{all}}$ , LEXORD was the third most effective ranking and significantly better than the fourth most effective ranking USERBEST. Both simple learning schemes were significantly better than the next best method ACCESSDATE.

- (3) For  $\mathcal{S}_{2-50}$ , LEXORD was again the third most effective ranking method. However, in this case NAME outperformed USERBEST, which was now the fifth best method. LEXORD was significantly better than NAME, however NAME was not significantly better than USERBEST. The next best method PATH was already significantly worse than USERBEST.
- (4) For  $\mathcal{S}_{>50}$ , LEXORD was outperformed by USERBEST and both were outperformed by ACCESSDATE and UPDATEDATE. However, for all four learning methods the difference was not statistically significant. (The next-best method, CREATEDATE, already performed significantly worse.)

The picture is even sharper when considering effectiveness at  $k$  (Table 6.2). For all  $\mathcal{S} \in \{\mathcal{S}_{\text{all}}, \mathcal{S}_{2-50}, \mathcal{S}_{>50}\}$ , and all  $k \in \{1, 2, 5, 10\}$ , SVM-90, SVM-75, and LEXORD have been found to be the three most effective ranking methods (typically in this order). The effectiveness of the three leaders appear to be rather comparable, however, one level lower lies the fourth most effective method USERBEST. In the third level we find our basic features. USERBEST is more effective than the rest of the basic methods for all  $k$  in  $\mathcal{S}_{\text{all}}, \mathcal{S}_{2-50}$ , and for  $k \in \{5, 10\}$  in  $\mathcal{S}_{>50}$ .

*We conclude that both our “multi-feature” learning-based ranking methods have performed significantly better than all of the single-feature-based ranking methods, across all of  $\mathcal{S}_{\text{all}}, \mathcal{S}_{2-50}$  and  $\mathcal{S}_{>50}$ , with respect to both general and top- $k$  effectiveness (for any  $k \in \{1, 2, 5, 10\}$ ). In addition, USERBEST is usually only as good as the best single-based-feature and therefore not very useful.*

It is important to note that:

1. LEXORD was learned to optimize only Score, yet its performance was impressive on both Score and TopScore. Separate learning of LEXORD and USERBEST to optimize Score and TopScore can only improve the results for the latter evaluation criteria.

2. SVM-xx ranking functions were learned to minimize the general ranking error, and not directly **Score** and/or **TopScore**. It is quite possible that learning ranking functions to directly optimize these evaluation criteria (as recently suggested in [20]) will improve the effectiveness of the ranking function methodology even further.

## Chapter 7

# Reasoning to Rank and Selectiveness of the Features

The results in the previous chapter clearly suggest adopting some or another learning-based ranking method as the primary ranking scheme. Using these methods, however, is relevant only in presence of sufficient training data. While in many domains this limitation is less problematic, in our case it can be a real obstacle. Desktop search users tend to issue only a few queries a day, and thus, the *cold-start* period of the system may even last a few months. In attempt to address the cold-start problem more effectively than just (temporarily) returning to some single-feature-based ranking method, we went back to our general intuitions on query selectiveness, discussed in Chapter 5.

Recall that users always aim at formulating a selective query, allowing the desktop search engine to automatically separate between the desired file and the rest of the files in the system. Since a query is a set of words, the selectiveness of the query is meaningful only with respect to one or more of the textual features of the files, i.e., in our case, NAME, PATH, CONTENT, and QUERYLOG. However, the user may not state (and may not even remember) with respect to which of these feature(s) she expects her query to be selective. Targeting this lack of information, we have defined a novel ranking method, called SELECTIVE, that aims at alleviating this problem. This method is based

on a certain form of *reasoning* about the result set  $\mathbf{Cand}(q, \mathcal{F})$ , aimed at inferring the correct way to rank this set of files with respect to the query in question.

Before we formally specify SELECTIVE, let us provide the basic intuition behind this method by a (somewhat extreme) example. Suppose the user poses a query  $q$  resulting in a large set of results  $\mathbf{Cand}(q, \mathcal{F})$ , and we have (i)  $q \subseteq \text{content}(f)$  for all files  $f \in \mathbf{Cand}(q, \mathcal{F})$ , and (ii) there is only one file  $f' \in \mathbf{Cand}(q, \mathcal{F})$  such that  $q \cap \text{name}(f') \neq \emptyset$ . Ignoring for a moment all the other features, and assuming that the user strives to formulate a selective query, it is reasonable to conjecture that the user formulated  $q$  while having in mind (either implicitly or explicitly) the filename of  $f_{q|\mathcal{F}}^*$ . And from this, we have that  $f'$  is more likely to be the desired file  $f_{q|\mathcal{F}}^*$  than any other file in  $\mathbf{Cand}(q, \mathcal{F})$ .

Extending this intuition to typically more fuzzy situations that happen in practice, SELECTIVE combines (i) the information carried by the textual properties of the files in  $\mathbf{Cand}(q, \mathcal{F})$ , and (ii) the (observed on  $\mathbf{Cand}(q, \mathcal{F})$ ) frequency of textual connection between each such property and query  $q$ . Formally, SELECTIVE is computed as follows. Let  $q$  be a query, and  $\mathbf{Cand}(q, \mathcal{F})$  be a result set for  $q$ . We use  $\text{nz}(\text{FEATURE}_q)$  to denote the number of files  $f \in \mathbf{Cand}(q, \mathcal{F})$  that have a non-zero (that is, some non-trivial) value  $\text{FEATURE}_q(f)$ . Given that, for each file  $f \in \mathbf{Cand}(q, \mathcal{F})$  we set

$$\text{SELECTIVE}_q(f) \stackrel{\text{def}}{=} \sum_{\text{FEATURE} \in \{\text{NAME, PATH, CONTENT, QUERYLOG}\}} \frac{\text{FEATURE}_q(f)}{\text{nz}(\text{FEATURE}_q)} \quad (7.1)$$

Intuitively, given  $\mathbf{Cand}(q, \mathcal{F})$ , SELECTIVE ranking aims at adaptively emphasizing the purportedly more selective features. Technically, it does this by following the principle underlying the standard IDF (inverse document frequency) measure used in IR—the contributions of different features to  $\text{SELECTIVE}_q(f)$  are combined via a weighted sum in which less “common” features (in  $\mathbf{Cand}(q, \mathcal{F})$ ) are given larger weights.

Please note that  $\text{SELECTIVE}_q(f)$  is significantly different than just summing the textual features, i.e., without normalization by their selectiveness among the results scores. When comparing between  $\text{SELECTIVE}_q(f)$  and the summation over TF-IDF scores, we found that  $\text{SELECTIVE}_q(f)$  was more effective over all sets ( $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$  and  $\mathcal{S}_{>50}$ ). However, the increase in effectiveness was statistically significant only in 90% confidence level. The score of summation over TF-IDF is omitted since it was less effective than  $\text{SELECTIVE}_q(f)$ .

We note that  $\text{SELECTIVE}$ , as defined in Eq. 7.1, constitutes probably one of the simplest realizations of the basic idea of “reasoning about query selectiveness,” and more complex reasoning procedures, conforming to the same intuition, are possible. Interestingly, despite its simplicity,  $\text{SELECTIVE}$  yields surprisingly good results, especially for  $\mathcal{S}_{2-50}$ . (We did not expect  $\text{SELECTIVE}$  to perform particularly well on  $\mathcal{S}_{\text{all}}$  and  $\mathcal{S}_{>50}$ , since none of the textual-based features considered in  $\text{SELECTIVE}$  is expected to be sufficiently selective in these cases, as discussed earlier.) The relative effectiveness of  $\text{SELECTIVE}$  is

FEATURE	Score( $\tau$ , $\mathcal{S}_{\text{all}}$ )	FEATURE	Score( $\tau$ , $\mathcal{S}_{2-50}$ )	FEATURE	Score( $\tau$ , $\mathcal{S}_{>50}$ )
SVM-90	7.84	SVM-90	4.72	SVM-90	32.65
SVM-75	8.15	<b>Selective</b>	4.86	SVM-75	33.74
LEXORD	9.19	SVM-75	4.93	ACCESSDATE	36.20
USERBEST	9.64	LEXORD	5.15	UPDATEDATE	40.13
ACCESSDATE	9.85	NAME	5.66	USERBEST	40.8
UPDATEDATE	10.18	USERBEST	5.73	LEXORD	41.39
<b>Selective</b>	10.89	PATH	6.32	CREATEDATE	49.07
CREATEDATE	11.51	UPDATEDATE	6.43	NAME	58.34
NAME	11.53	ACCESSDATE	6.54	<b>Selective</b>	58.91
DIRRANK	12.56	QUERYLOG	6.59	DIRRANK	58.40
PATH	12.60	CONTENT	6.75	SIZE	61.64
SIZE	12.96	CREATEDATE	6.80	PATH	62.69
QUERYLOG	13.5	DIRRANK	6.80	QUERYLOG	68.63
<i>Random</i>	<i>14.45</i>	SIZE	6.85	<i>Random</i>	<i>73.87</i>
CONTENT	15.43	<i>Random</i>	<i>6.99</i>	NORMALIZEDSIZE	83.86
NORMALIZEDSIZE	16.24	NORMALIZEDSIZE	7.75	CONTENT	84.57
LEVEL	17.90	LEVEL	7.80	LEVEL	98.36

Table 7.1: Relative effectiveness of  $\text{SELECTIVE}$ .

presented in Table 7.1. For all three of  $\mathcal{S}_{\text{all}}$ ,  $\mathcal{S}_{2-50}$ , and  $\mathcal{S}_{>50}$ , SELECTIVE is better than (or at least as good as) its most effective component (among NAME, PATH, CONTENT, QUERYLOG). Notably, SELECTIVE is more effective than its most effective component on  $\mathcal{S}_{\text{all}}$  and  $\mathcal{S}_{2-50}$  (with statistical significance), and is statistically indistinguishable with FILENAME on  $\mathcal{S}_{>50}$ . Somewhat surprisingly, on  $\mathcal{S}_{2-50}$  SELECTIVE even successfully competes with our learning-based ranking methods—it appears to be more effective than LEXORD (with statistical significance) and is, in fact, statistically indistinguishable with SVM-75 and SVM-90. Thus, SELECTIVE appears highly effective for  $\mathcal{S}_{2-50}$ , even though it does not involve any learning.

FEATURE	TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{\text{all}}$ )				TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{2-50}$ )				TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{>50}$ )			
	$k = 1$	$k = 2$	$k = 5$	$k = 10$	$k = 1$	$k = 2$	$k = 5$	$k = 10$	$k = 1$	$k = 2$	$k = 5$	$k = 10$
SVM-90	34.4	45.2	56.3	64.9	36.6	48.4	61.1	71.7	17.4	22.6	32.2	40.9
SVM-75	32.8	42.8	55.0	63.5	34.5	45.8	59.6	70.5	19.1	21.7	32.2	39.1
LEXORD	34.0	41.8	52.2	62.5	37.2	45.8	60.0	68.5	8.7	13.9	27.8	41.7
<b>Selective</b>	27.1	37.5	53.3	63.1	29.3	41.1	59.4	72.7	9.6	12.2	22.6	29.6
USERBEST	24.3	34.6	46.7	57.1	26.6	38.1	51.3	62.0	6.0	10.4	23.5	40.0
UPDATEDATE	21.6	31.5	40.7	52.5	23.4	34.6	44.8	57.8	7.8	9.6	20.0	33.9
ACCESSDATE	19.2	30.0	40.9	52.3	19.2	29.0	40.9	52.3	7.0	12.2	22.6	36.5
NAME	16.5	29.2	41.8	51.7	18.1	32.5	47.3	60.5	3.5	6.1	13.9	20.9
CREATEDATE	17.7	27.3	37.6	48.5	19.2	30.0	42.2	54.1	5.2	8.7	14.8	28.7
DIRRANK	5.4	18.0	32.2	47.5	6.1	20.4	36.5	55.3	0.0	0.9	10.4	20.0
PATH	6.9	19.1	33.8	45.9	7.2	20.7	38.2	54.1	4.3	7.8	11.3	17.4
SIZE	16.1	26.6	39.2	42.3	17.8	29.6	44.8	50.6	2.6	5.2	11.3	13.0
QUERYLOG	8.1	18.2	28.9	40.5	8.7	20.0	33.0	49.9	3.5	5.2	7.8	7.8
CONTENT	13.2	23.6	36.9	45.0	14.6	26.4	43.0	55.1	1.7	4.3	6.1	9.6
Random	0.0	10.6	22.6	36.3	0.0	12.1	27.0	46.7	0.0	0.0	0.0	0.0
NORMALIZEDSIZE	14.5	21.6	29.5	36.3	16.0	24.3	34.4	44.4	1.7	2.6	4.3	7.8
LEVEL	4.6	15.7	26.7	34.9	5.1	17.8	31.5	43.2	0.0	0.9	2.6	6.1

Table 7.2: Same as Table 7.1, but for effectiveness at  $k \in \{1, 2, 5, 10\}$ .

Table 7.2 presents the effectiveness of SELECTIVE for top- $k$ . Observe that SELECTIVE is fourth-best in almost all cases, that is, next best after our learning-based ranking methods. The two exceptions are in the cases of  $\mathcal{S}_{>50}$  for  $k = 5$  and  $k = 10$ , where, as we discussed above, our expectations from SELECTIVE were the lowest. Furthermore, similarly to what we observed with

the learning-based ranking methods, in general here as well there is a sharp drop in effectiveness between `SELECTIVE` and the best single-feature-based ranking method.

We conclude this chapter by observing that, despite its simplicity, `SELECTIVE` yields quality results. Hence, we believe that `SELECTIVE` (or some possibly more sophisticated variation) is an excellent choice for ranking desktop search results during a cold-start period when sufficient training data for learning is not available.

## Chapter 8

# Secondary Ranking schemes

In Chapters 6 and 7, we have shown the effectiveness of two types of primary ranking schemes: the SELECTIVE scheme for the cold-start period of the system, and the learning schemes (SVM-90, SVM-75 or LEXORD) for routine use. Unfortunately, though effective, these schemes are not helpful unless users trust them and feel comfortable enough to use them.

In Chapter 2, we mentioned that numerous research works show that users prefer to sort search results according to date or a version of ranking scheme (i.e., similar ranking feature as our CONTENT feature). In [9] it was shown that most users choose to sort the search results according to Update Date and then by Content. [3] also supports the notion that users would prefer sorting according to file date or name. Most industrial projects choose to offer users the same ranking capabilities. The outcome of this may be that users prefer to rank according to CONTENT or UPDATEDATE and feel more comfortable using these methods, e.g., than a learning scheme.

Presuming that no matter how effective the primary ranking is, the user may still be more comfortable ranking according to CONTENT or UPDATEDATE, we decided to explore a different approach. In this new approach ranking takes place in two phases. In the first phase the user ranks according to one of the primary ranking schemes (CONTENT or UPDATEDATE). If the ranking scheme in the first phase was not helpful, i.e, the user did not chose any file, we move to

the next phase. The second phase offers the user a secondary ranking scheme, abstractly denoted as  $\text{SECONDARY}\langle\tau\rangle$  that will be specified later in this chapter. This two-phase ranking is consistent with [28], which claimed that users prefer to search in small steps and need to “feel in control” during the search.

The advantage of two-phase ranking is in the following assumption. In our new approach we assume that if the user did not choose a document in the first phase, then we conclude that the desired document  $f_{q|\mathcal{F}}^* \in \mathcal{F}$  is not placed in the top  $k$  documents when ranking according to  $\tau$ .

In order to specify our secondary ranking scheme  $\text{SECONDARY}\langle\tau\rangle$  we first define  $\mathcal{S}^\tau$ .  $\mathcal{S}^\tau$  is the subset of query/fileset pairs for which the  $\tau$  ranking scheme did not place the desired document among the top  $k$  documents. Formally:

$$\mathcal{S}^\tau = \{(q, \mathcal{F}) \in \mathcal{S} \mid \text{Exp}(\tau, f_i^* \mid q_i, \mathcal{F}_i) > k\}$$

According to [12], most users view only the top 5 documents from the result list and therefore in this thesis we will always assume that  $k = 5$ .

Note that we later use  $\mathcal{S}_{2-50}^\tau$  to denote the subset of  $\mathcal{S}^\tau$  which contains only queries with 2-50 results and  $\mathcal{S}_{>50}^\tau$  to denote the subset of  $\mathcal{S}^\tau$  which contains only queries with over 50 results.

Based on these subsets we now formulate secondary ranking schemes. We define  $\text{SVM-xx}\langle\tau\rangle$  as the value that was given to a file  $f$  by SVM classification, based on  $\mathcal{S}^\tau$  and not on the entire training set  $\mathcal{S}$ . We prefer to omit those queries from our training set since we would like to focus the classification effort on queries that were not ranked high according to  $\tau$ . Specifically we consider the following ranking schemes  $\text{SVM-xx}\langle\text{C}\rangle$  and  $\text{SVM-xx}\langle\text{U}\rangle$ , derived by running  $\text{SVM}^{\text{light}}$  only for the subsets  $\mathcal{S}^{\text{C}}$  and  $\mathcal{S}^{\text{U}}$ , respectively, for both 75 and 90 percent of the data.

We similarly define  $\text{LEXORD}\langle\text{C}\rangle$  and  $\text{LEXORD}\langle\text{U}\rangle$  as learning  $\text{LEXORD}$  only on queries that did not rank the desired document among the top 5 documents according to  $\text{CONTENT}$  or  $\text{UPDATEDATE}$ , respectively.

FEATURE	Score( $\tau, \mathcal{S}_{\text{all}}^C$ )	FEATURE	Score( $\tau, \mathcal{S}_{2-50}^C$ )	FEATURE	Score( $\tau, \mathcal{S}_{>50}^C$ )
SVM-90(C)	13.24	SVM-90(C)	7.32	SVM-75(C)	31.03
SVM-75(C)	13.28	SVM-75(C)	7.48	SVM-90(C)	31.37
SVM-90	14.11	SVM-90	7.79	SVM-90	33.48
SVM-75	14.51	SVM-75	8.24	SVM-75	33.73
LEXORD(C)	15.86	LEXORD	8.71	LEXORD(C)	36.59
LEXORD	16.12	LEXORD(C)	9.07	LEXORD	38.81

Table 8.1: Relative effectiveness of SVM-75(C), SVM-90(C) and LEXORD(C).

FEATURE	TopScore $_k(\tau, \mathcal{S}_{\text{all}}^C)$				TopScore $_k(\tau, \mathcal{S}_{2-50}^C)$				TopScore $_k(\tau, \mathcal{S}_{>50}^C)$			
	$k=1$	$k=2$	$k=5$	$k=10$	$k=1$	$k=2$	$k=5$	$k=10$	$k=1$	$k=2$	$k=5$	$k=10$
SVM-90(C)	22.6	31.7	49.2	61.2	25.7	35.3	55.3	69.5	13.0	20.4	30.6	39.8
SVM-75(C)	23.2	31.9	49.2	62.5	25.7	35.6	55.9	72.0	15.7	20.4	28.7	38.0
SVM-90	23.0	31.2	47.4	60.2	25.4	34.4	52.6	68.1	15.7	21.3	31.5	39.8
SVM-75	22.3	29.4	46.7	57.9	23.9	32.3	51.7	65.6	17.6	20.4	31.5	38.0
LEXORD(C)	18.2	27.1	43.7	58.7	22.4	32.6	49.2	64.9	5.6	10.2	26.9	42.6
LEXORD	18.5	27.3	41.7	58.7	21.8	32.0	46.2	65.2	8.3	12.9	27.7	41.7

Table 8.2: Same as Table 8.1, but for effectiveness at  $k \in \{1, 2, 5, 10\}$ .

The relative effectiveness of SVM-90(C), SVM-75(C) and LEXORD(C) is presented in Table 8.1. In both  $\mathcal{S}_{\text{all}}^C$  and  $\mathcal{S}_{2-50}^C$ , SVM-90(C) and SVM-75(C) performed significantly better than SVM-90 and SVM-75. However, LEXORD(C) was statistically insignificant with LEXORD. In  $\mathcal{S}_{>50}^C$ , though they scored higher, SVM-90(C) and SVM-75(C) were not significantly better than SVM-90 and SVM-75. However, LEXORD(C) was significantly better than LEXORD.

Table 8.2 presents the effectiveness of SVM-xx(C) and LEXORD(C) for top- $k$ . For both  $\mathcal{S}_{\text{all}}^C$  and  $\mathcal{S}_{2-50}^C$ , SVM-75(C) is the best scheme followed by SVM-90(C). For  $\mathcal{S}_{>50}^C$ , SVM-90 and SVM-75 seem to perform better. When comparing the effectiveness of LEXORD(C) and LEXORD it is hard to say which one is better.

The relative effectiveness of SVM-90(U), SVM-75(U) and LEXORD(U) is presented in Table 8.3. In  $\mathcal{S}_{\text{all}}^U$  only SVM-75 significantly dominates SVM-75(U).

FEATURE	Score( $\tau, \mathcal{S}_{\text{all}}^{\text{U}}$ )	FEATURE	Score( $\tau, \mathcal{S}_{2-50}^{\text{U}}$ )	FEATURE	Score( $\tau, \mathcal{S}_{>50}^{\text{U}}$ )
SVM-90	14.77	SVM-90⟨U⟩	7.81	SVM-90	37.39
SVM-75	15.47	SVM-75⟨U⟩	8.02	SVM-75	39.0
SVM-90⟨U⟩	15.81	SVM-90	8.28	SVM-90⟨U⟩	43.76
SVM-75⟨U⟩	17.24	SVM-75	8.73	LEXORD⟨U⟩	44.02
LEXORD⟨U⟩	17.33	LEXORD	9.50	LEXORD	48.09
LEXORD	18.10	LEXORD⟨U⟩	9.65	SVM-75⟨U⟩	49.4

Table 8.3: Relative effectiveness of SVM-75⟨U⟩ and SVM-90⟨U⟩ .

FEATURE	TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{\text{all}}^{\text{U}}$ )				TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{2-50}^{\text{U}}$ )				TopScore <sub>k</sub> ( $\tau, \mathcal{S}_{>50}^{\text{U}}$ )			
	k = 1	k = 2	k = 5	k = 10	k = 1	k = 2	k = 5	k = 10	k = 1	k = 2	k = 5	k = 10
SVM-90	18.9	25.7	44.3	57.2	20.9	28.7	50.2	65.2	12.0	15.2	23.9	33.7
SVM-75	17.9	22.8	41.6	55.5	18.7	24.9	47.0	64.1	15.2	15.2	22.8	30.4
SVM-90⟨U⟩	18.6	26.2	43.8	61.0	21.5	30.2	49.8	69.3	8.7	12.0	22.8	37.0
SVM-75⟨U⟩	18.9	27.1	43.6	57.7	21.5	30.5	49.5	65.9	9.8	15.2	22.8	33.7
LEXORD⟨U⟩	12.8	20.1	35.8	54.7	15.0	23.1	40.8	63.0	5.4	9.8	18.5	30.4
LEXORD	14.0	20.6	34.1	53.0	16.8	24.3	39.6	60.7	4.3	7.6	15.2	30.4

Table 8.4: Same as Table 8.3, but for effectiveness at  $k \in \{1, 2, 5, 10\}$ .

The rest of the results are not significant. In  $\mathcal{S}_{2-50}^{\text{U}}$  however, SVM-90⟨U⟩ and SVM-75⟨U⟩ are significantly better than SVM-90 and SVM-75, respectively. In  $\mathcal{S}_{>50}^{\text{U}}$  only LEXORD⟨U⟩ is significantly better than LEXORD.

Table 8.4 presents the effectiveness of SVM-xx⟨U⟩ for top- $k$ . It is hard to determine which method is better. However, LEXORD⟨U⟩ seems better than LEXORD in  $\mathcal{S}_{>50}^{\text{U}}$  and as good as LEXORD in  $\mathcal{S}_{\text{all}}^{\text{U}}$  and  $\mathcal{S}_{2-50}^{\text{U}}$ .

The above results indicate that when trying to find a good secondary ranking for UPDATE\_DATE, LEXORD⟨U⟩ is better than LEXORD. However, SVM-xx⟨U⟩ is not significantly better than SVM-xx. On the other hand, when trying to find a good secondary ranking to CONTENT, SVM-xx⟨C⟩ is better than SVM-xx however LEXORD⟨C⟩ is not significantly better than LEXORD. The root cause is described below.

As explained in Chapter 6.2, LEXORD sorts basic features according to their effectiveness and then ranks each file according to this order. In this

type of aggregation the most effective features are the ones at the top of the list, whereas the features at the bottom of the list hardly have any effect. Removing queries from the training set might change the effectiveness of each basic feature, and therefore the order. In this case, changing the placement of features that were located at the top of the list, such as `UPDATEDATE`, have a great effect on the new ordering of features and therefore  $\text{LEXORD}\langle\text{U}\rangle$  is significantly better than  $\text{LEXORD}$ . However `CONTENT` was not located among the most effective basic feature and therefore  $\text{LEXORD}\langle\text{C}\rangle$  is similar to  $\text{LEXORD}$ , since its feature ordering was similar (because `CONTENT` is rather poor on its own).

When comparing the performance of  $\text{SVM-xx}\langle\tau\rangle$  and  $\text{SVM-xx}$  we can see that the results are quite different, since SVM gives each feature a weight rather than ranking by the order that the most effective features set. In  $\mathcal{S}_{2-50}^\tau$  we can see that both  $\text{SVM-xx}\langle\text{C}\rangle$  and  $\text{SVM-xx}\langle\text{U}\rangle$  are significantly better than  $\text{SVM-xx}$ . This is because both `UPDATEDATE` and `CONTENT` were influencing features in  $\mathcal{S}_{2-50}$  (but were not the most effective features). Removing queries that ranked the desired document among the top 5 documents from the training set has improved the learning model on  $\mathcal{S}_{2-50}^\tau$ .

In  $\mathcal{S}_{>50}$  `CONTENT` was one of the least influencing features. Therefore, SVM gave it a very low weight and using  $\mathcal{S}^{\text{C}}$  instead of  $\mathcal{S}$  as a training set hardly changed the model. Hence we can see that though  $\text{SVM-xx}\langle\text{C}\rangle$  is ranked higher than  $\text{SVM-xx}$ , the difference is insignificant. On the other hand, in  $\mathcal{S}_{>50}$  `UPDATEDATE` was one of the most influencing features. Using  $\mathcal{S}^{\text{U}}$  instead of  $\mathcal{S}$  as a training set did not help to build a better model, but rather the other way around. The explanation for this behavior may be that `UPDATEDATE` is a very effective feature, even in queries in which it did not rank the desired document among the top 5 documents. Perhaps using  $k$  larger than 5 could have improved the results.

We conclude this chapter by observing that  $\text{SVM-xx}\langle\text{C}\rangle$  is a better method

than SVM-XX as a secondary ranking method. Hence, we recommend using it with the primary method CONTENT or with another content related feature such as SELECTIVE. LEXORD⟨U⟩ was also better than LEXORD and therefore we would recommend using it as well as a secondary ranking to UPDATEDATE.

## Chapter 9

# Conclusions and Future Work

In this paper we studied the ranking problem for desktop search. Based on our user study, we evaluated the effectiveness of basic (single-feature-based) ranking schemes, learning-based ranking and ranking based on selectiveness reasoning and two phase learning-based ranking schemes. Our findings show that learning-based ranking is significantly more effective than the basic ranking schemes, both in terms of expected placement and in terms of effectiveness at  $k$ . We have also shown that our novel reasoning based ranking, *SELECTIVE*, is effective (despite its simplicity), and hence, is a good choice as a primary ranking scheme for the cold-start period of the system. In addition, we showed the effectiveness of two phase ranking schemes in comparison to our learning schemes.

As future work, we will investigate whether ranking based on selectiveness reasoning can also be effective in general ad hoc information retrieval. We also plan to extend *SELECTIVE* to take into consideration additional file features, such as the date-based features. Another important extension to our framework is to develop a conceptual treatment of missing feature values (e.g., of missing content), since currently these features are given a default value of zero. In addition, we will try using qualitative rank aggregation techniques, as suggested in [10], to determine whether they can lead to more effective ranking. Finally, we would investigate more the justification for the two-phase

ranking scheme in terms of how much users are interested in ranking according to `CONTENT` or `UPDATEDATE`.

# Bibliography

- [1] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of CHI'92*, pages 619–626, 1992.
- [2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press Series. Addison-Wesley, May 1999.
- [3] D. Barreau and B. A. Nardi. Finding and reminding - File organization from the desktop. *ACM SIGCHI Bulletin*, 27(3):39–43, 1995.
- [4] D. Bertsekas, A. Nedic, and A. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.
- [5] W. Cohen, R. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.
- [6] Copernic desktop search. [www.copernic.com/](http://www.copernic.com/).
- [7] C. Cortes and V. Vapnik. Support–vector networks. *Machine Learning Journal*, 20:273–297, 1995.
- [8] X. Dong and A. Halevy. A platform for personal information management and integration. In *Proceedings of CIDR'05*, pages 119–130, 2005.
- [9] S. Dumais, E. Cutrell, JJ Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff I've seen: a system for personal information retrieval and re-use. In *Proceedings of SIGIR'03*, pages 72–79. ACM Press, 2003.

- [10] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proceedings of the WWW'01*, pages 613–622, Hong Kong, 2001.
- [11] T. Erickson. The design and long-term use of a personal electronic notebook: A reflective analysis. In *Proceedings of CHI'96*, pages 11–18, 1996.
- [12] Eyetools study. <http://www.enquiro.com/eye-tracking-pr.asp>.
- [13] R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. Searching the workplace web. In *Proceedings of WWW'03*, pages 366–375. ACM Press, 2003.
- [14] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. MyLifeBits: Fulfilling the Memex vision. In *Proceedings of ACM Multimedia '02*, pages 235–238, 2002.
- [15] Google desktop. <http://desktop.google.com/features.html#search>.
- [16] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. 2000.
- [17] K. Höffgen, H. Simon, and K. van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50:114–125, 1995.
- [18] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, chapter 11. MIT Press, 1999.
- [19] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of KDD'02*, pages 154–161, 2002.
- [20] T. Joachims. A support vector method for multivariate performance measures. In *Proceedings of the International Conference on Machine Learning*, pages 377–384, 2005.

- [21] T. Joachims, L. Granka, B. Pang, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of SIGIR'05*, pages 154–161, 2005.
- [22] M. Lansdale. The psychology of personal information management. *Applied Ergonomics*, 19(1):55–66, 1988.
- [23] The Lowell database research self assessment. Attendees at Lowell Workshop, May 2003.
- [24] H. Marais and K. Bharat. Supporting cooperative and personal surfing with a desktop assistant. In *Proceedings of UIST'97*, pages 129–138, 1997.
- [25] W. Nejdl and R. Paiu. Desktop search - how contextual information influences search results and rankings. In *Proceedings of the IRiX Workshop at SIGIR'05*, 2005.
- [26] Spotlight. <http://www.apple.com/macosx/features/spotlight/>.
- [27] J. Teevan, W. Jones, and B. B. Bederson, editors. *Special Issue on Personal Information Management*, volume 49 of *Communication of the ACM*, 2006.
- [28] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger, editors. *The perfect search engine is not enough: a study of orienting behavior in directed search*, Proceedings of the SIGCHI conference on Human factors in computing systems, 2004.
- [29] E. Voorhees. The TREC-8 question answering track report. In *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, pages 77–82. NIST, 1999.

- [30] D. Wolber, M. Kepe, and I. Ranitovic. Exposing document content in the personal web. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, pages 151–158. ACM Press, 2002.