Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica
Settore Scientifico Disciplinare: INF/01

Ph.D. Thesis

Improving Search Effectiveness through Query Log and Entity Mining

Diego Ceccarelli

Supervisor
Dott. Raffaele Perego

Referee	Referee
Dott. Peter Mika	Prof. Gerhard Weikum

Abstract

The Web is the largest repository of knowledge in the world. Everyday people contribute to make it bigger by generating new web data. Data never sleeps. Every minute someone writes a new blog post, uploads a video or comments on an article. Usually people rely on Web Search Engines for satisfying their *information needs*: they formulate their needs as text queries and they expect a list of highly relevant documents answering their requests. Being able to manage this massive volume of data, ensuring high quality and performance, is a challenging topic that we tackle in this thesis.

In this dissertation we focus on the *Web of Data*: a recent approach, originated from the Semantic Web community, consisting in a collective effort to augment the existing Web with semistructured-data. We propose to manage the *data explosion* shifting from a retrieval model based on documents to a model enriched with *entities*, where an entity can describe a person, a product, a location, a company, through *semi-structured information*.

In our work, we combine the Web of Data with an important source of knowledge: *query logs*, which record the interactions between the Web Search Engine and the users. Query log mining aims at extracting valuable knowledge that can be exploited to enhance users' search experience. According to this vision, this dissertation aims at improving Web Search Engines toward the mutual use of query logs and entities.

The contributions of this work are the following: we show how historical usage data can be exploited for improving performance during the snippet generation process. Secondly, we propose a query recommender system that, by combining entities with queries, leads to significant improvements to the quality of the suggestions. Furthermore, we develop a new technique for estimating the *relatedness* between two entities, i.e., their semantic similarity. Finally, we show that entities may be useful for automatically building explanatory statements that aim at helping the user to better understand if, and why, the suggested item can be of her interest.

Acknowledgments

It is hard to find the words to thank all the people that helped me during my Ph.D. I am really thankful to my advisor Raffaele Perego, for his guidance, support and encouragement during these years at CNR. Raffaele gave me this opportunity and the freedom to choose the topics I liked most; I feel very fortunate to have him as a mentor and friend. In addition to Raffaele, I will be forever in debt to Claudio Lucchese, Fabrizio Silvestri and Salvatore Orlando, also members of the HPCLab. They are great mentors, and are still important guides in my growth as a researcher. In particular, I have to thank Fabrizio for pushing me to spend time abroad and for creating those opportunities; I wish him all the best in his new research position at Yahoo!.

Moreover, thanks to the external reviewers, Peter Mika and Gerhard Weikum, for taking the effort to contribute to this work by reviewing it: I am really honored of having their endorsement. I am also grateful to Roberto Grossi and Anna Bernasconi for their precious advice.

I would like to thank Giovanni Tummarello and the Digital Enterprise Research Institute in Galway, that hosted me for three amazing months. During my internship I had the opportunity to work (and to drink Guinness) with beautiful people like Giovanni, Stéphane Campinas, Renaud Delbru, and Giulio Cesare Solaroli that gave up their time to help me.

I am grateful to Ricardo Baeza-Yates that hosted me at Yahoo! Research in Barcelona, and also to Roi Blanco, Berkant Barla Cambazoglu, Peter Mika and Jordi Atserias for their valuable guidance and fruitful discussions. In Barcelona I also had good times thanks to the Yahooers, and to my flatmates Chiara, Giuseppe, Marcelo and Vittoria.

I had four amazing years in the HPCLab, and this is mostly due to the special people I met in the lab. I spent two years in the C64 room together with Franco, Gabriele C. and Gabriele T., that, like older brothers, were always available for helping me and are still good friends. I had a good time with everybody in the lab, but I want to individually thank Cristina, Daniele, Rossano, and Salvo: people with whom I shared hard work, long deadline nights, travels, and so many laughs. You made all this work less hard.

This thesis is dedicated to all my *famiglia*, where *famiglia* means all the people that in these years have always been very close. I want to thank my mother who always had faith in me, and Sergio for all his valuable advice. Thanks to my father, Tullia, Flavio and Giulio for their unconditional support. Thanks to my musical family: Carlo, Chiara, Giovanni, Stefano and Thomas with whom I shared billions of reels, jigs, sad songs and beers. I never felt I was alone thanks to special friends like Dalia, Elisa, Laura, Marco, Robin, Irene, Alessandro, Riccardo, Tatonzio, Peppe, Donato, Ugo V., Ugo S., Stéphane and Kasia. Also, thanks to the Montesi family for welcoming me in their home. Thanks *famiglia*, I would have never reached this goal without your help.

Last but not least, to my incredible Sara, for her trust, love, hard work, and the beauty she has brought into my life.

Contents

List of Figures

List of Tables

Introduction

Web search engines (WSE) are about twenty years old. Despite this recency, however, it is amazing how quickly they evolve. A WSE is probably nowadays the most complex software ever conceived, implementing an *Information Retrieval* system.

Baeza-Yates and Ribeiro-Neto define Information Retrieval [10] as the task of representing, storing, organizing, and accessing *information items*. Given a *query* expressing a user information need, as for a Database system, the task of an Information Retrieval system is to retrieve information *relevant* to that query. The key difference is that databases retrieve all items matching exactly queries expressed in a formal language, while information retrieval systems manage queries expressed in natural language, and try to do their best in retrieving matching documents and ranking them by some subjective measure of relevance. Queries in natural language are not easy to handle. For example *homonyms* and *polysemes* – terms with the same or ambiguous meanings – have to be dealt with.

A WSE is an IR system on a very *large scale* that store, organize and give access to a particular kind of documents: Web documents. Unfortunately, the documents that are hyperlinked and distributed over all the Internet, are very noisy and heterogeneous for (quality of) content and formats. This makes very hard the task of collecting and managing effectively them. Moreover, queries submitted to WSEs are usually short – about two-terms on average – and the subsumed information need often difficult to understand. The necessity of dealing with these characteristics made WSEs just from their birth very different from the information retrieval systems introduced in early 1960 for searching the clean and homogeneous documents of digital libraries. The peculiarity of the data managed, their size, and the novel interaction with the users asked for deep changes to information retrieval foundations that were strongly contaminated by other disciplines such as natural language processing, data mining, high performance computing, statistics, human computer interaction, complex networks. In addition, several factors – Web 2.0, Web economy, geo-political changes, advances in technology and popularity of smart mobile devices, etc – caused the Web to grow at unprecedented rates in size, complexity, and number of users. In this extraordinary context, (i) quality of retrieved results, and, (ii) time needed to process queries against huge indexes, two key issues affecting even the first WSEs appeared in the '90, have nowadays a paramount importance.

This thesis deals with techniques to improve effectiveness and efficiency of some WSE tasks by exploiting and combining two important sources of knowledge:

Query Log Data: WSEs collect query log information about their use, such as the queries submitted, the results clicked, the users' sessions, etc. Several studies proved that these logs constitute a gold mine that can be analyzed and mined in order to significantly improve WSEs both in terms of efficiency and efficacy [129];

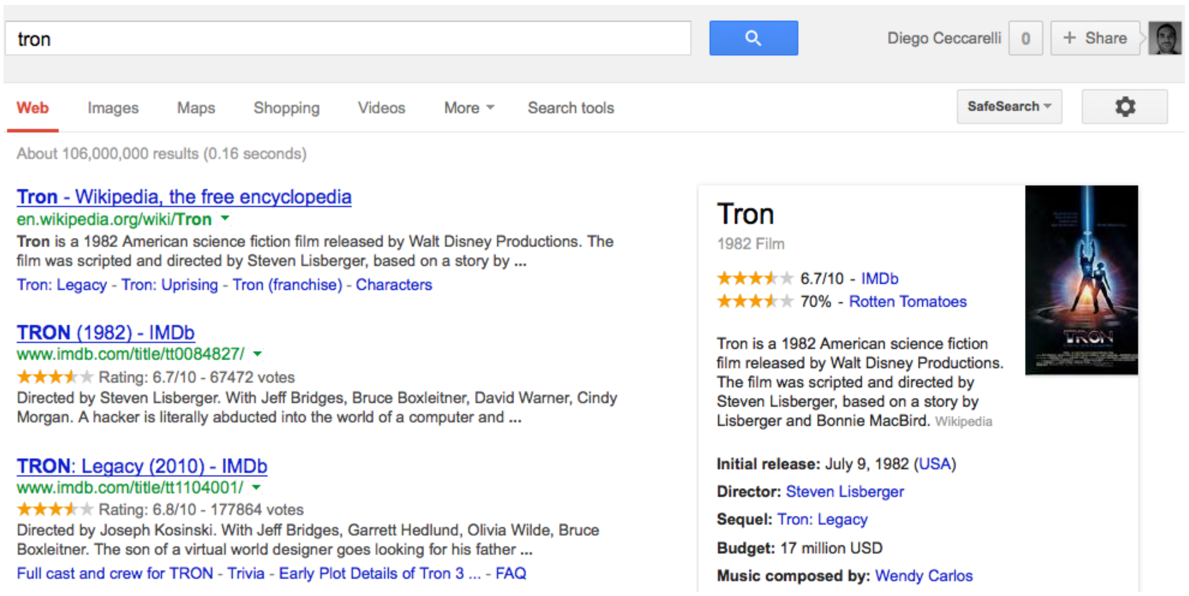
Web of Data: The Web of Data is a recent approach, originated from the Semantic Web [24], and consists in the collective effort to augment the existing Web with semistructured-data, with the ultimate goal to enable automatic intelligent reuse of the data by agents and search engines. In order to reach this target, Web of Data promotes standards and instruments that allow people to publish and retrieve enriched data.

Recently, the major search companies started to show a great interest for the Web of Data: from Figure I.1 it can be seen that when a user performs a query about an item (in the case depicted, the movie `Tron`), Google returns together with the traditional list of ten *blue links*, a box containing semi-structured information about the item. The semi-structured information items available on the Web can describe any type of *entity*: a person, a product, a location, a document, etc. Independently of the type of the entity, they represent a very precious source of information to enhance Web search effectiveness.

In our example, the content in the box is extracted from Wikipedia and *Freebase*, one of largest knowledge bases composed by semistructured-data manually inserted and preserved by volunteers (Freebase was acquired by Google on July 16, 2010[1]). In the top of the box movie ratings provided by authoritative sources for movie reviews such as IMDb and Rotten Tomatoes are also reported. It is worth noting that these sites as many others share their semistructured-data with search engines by annotating their pages with a semantic markup.

More and more structured and semi-structured data sources are becoming available. With the current availability of data-publishing standards and tools,

[1]`http://googleblog.blogspot.com/2010/07/deeper-understanding-with-metaweb.html`



About 106,000,000 results (0.16 seconds)

Tron - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Tron
Tron is a 1982 American science fiction film released by Walt Disney Productions. The film was scripted and directed by Steven Lisberger, based on a story by ...
Tron: Legacy - Tron: Uprising - Tron (franchise) - Characters

TRON (1982) - IMDb
www.imdb.com/title/tt0084827/
★★★★☆ Rating: 6.7/10 - 67472 votes
Directed by Steven Lisberger. With Jeff Bridges, Bruce Boxleitner, David Warner, Cindy Morgan. A hacker is literally abducted into the world of a computer and ...

TRON: Legacy (2010) - IMDb
www.imdb.com/title/tt1104001/
★★★★☆ Rating: 6.8/10 - 177864 votes
Directed by Joseph Kosinski. With Jeff Bridges, Garrett Hedlund, Olivia Wilde, Bruce Boxleitner. The son of a virtual world designer goes looking for his father ...
Full cast and crew for TRON - Trivia - Early Plot Details of Tron 3 ... - FAQ

Tron
1982 Film
★★★★☆ 6.7/10 - IMDb
★★★★☆ 70% - Rotten Tomatoes
Tron is a 1982 American science fiction film released by Walt Disney Productions. The film was scripted and directed by Steven Lisberger, based on a story by Lisberger and Bonnie MacBird. Wikipedia

Initial release: July 9, 1982 (USA)
Director: Steven Lisberger
Sequel: Tron: Legacy
Budget: 17 million USD
Music composed by: Wendy Carlos

Figure I.1: The result page for the query `tron`

adding structure and semantic to data published is becoming a mass activity. Nowadays, the Web of Data is no longer limited to a few trained specialists, but it attracts companies, governments and individuals. There are many prominent fields in which semi-structured data publishing efforts became prevalent. Just to make a few examples: the UK government shares open government data, in the editorial world journals such as the New York Times or Reuters publish rich metadata about their news articles, BestBuy publishes product descriptions in a machine-readable format.

Progressively and inexorably, the prediction that Web search would be increasingly semantic and graph-based is becoming true.

Contributions of the Thesis

In this thesis, we illustrate four new contributions in which we exploit Query Log Data and the Web of Data for improving Web Search Engine quality and performance.

Query Biased Snippets This Chapter, based on [44], presents a study on how to improve performance during the result page construction, by dynamically generating document surrogates to be managed by a snippet caching system. Our thesis is that the knowledge stored in query logs can help in building concise surrogates, that we call *supersnippets*. Our experiments show that supersnippets are very effective for efficient retrieval of high quality snippets. Our technique exploits the fact that the set of queries for which a document is retrieved is in most cases small, i.e., a large part of documents are retrieved only for satisfying *one* user information need.

Semantic Query Recommendations This Chapter is based on [42, 43]. We investigate how to combine queries and entities in order to improve the user experience on the WSE: we consider as a case study Europeana[2], a portal that aims to collect metadata about millions of books, paintings, films, museum objects and archival records that have been digitized by European institutions, and make them searchable to users. We perform a deep analysis on the query logs provided by Europeana and we propose a novel method to generate *semantic suggestions*. Our recommender produces a list of entities (represented by URIs) rather than returning to the user a list of possible query suggestions. We show that providing such kind of recommendations has several benefits for the user experience. We also propound a novel technique for evaluating the relevance of semantic suggestions using the concept of *relatedness* [109, 110], a graph-based technique to compute the *semantic distance* between two entities.

Learning Relatedness Measures for Entity Linking This Chapter is based on [45]. Relatedness is an important measure, not only for evaluating suggestions but also for recognizing the entities mentioned in raw text (web queries, web pages, news, tweets . . .); in a nutshell, the process of detecting entities in a document consists of two tasks: *spot* the fragments of text that may refer to an entity, and *disambiguate* the correct entity among several possible candidates. The same mention may in fact refer to more than one entity, e.g., the mention `Tron` could refer to the film, to an arcade game, and even to *Nicolò Tron*, Doge of Venice. The semantic relatedness among the entities mentioned in the textual context is obviously a important measure to drive a correct disambiguation process. We propose a general approach based on learning to rank techniques for improving the precision of the relatedness function that significantly improves state-of-the-art solutions.

On Generating News Recommendation Explanations Finally, in Chapter 5 based on [27], we focus on explaining the relations between the news read online by the user and the ones recommended. News systems, in particular, benefit from recommendations, given the fact that online news systems are exploratory by nature. People browse through the list of daily news usually driven by personal interest, curiosity, or both. Due to the amount of news

[2]`www.europeana.eu`

items available, online news services deploy recommender systems that help the users to find potentially interesting news. In particular, we aim at enhancing the users' experience on news platforms, like Google News, Yahoo! News, by motivating the recommended news shown by means of a tool that automatically generate brief, yet significant, explanations. We introduce different ways of generating explanations for the recommended news, and we develop a machine learning based method whose goal is to rank these explanations in order to maximize the usefulness of the recommendation itself.

Outline

The thesis is organized as follows: Chapter 1 overviews the main topics involved in this work: we briefly describe the architecture of a Web Search Engine, we introduce Query Log Mining techniques, and we present the Web of Data, focussing on Semantic Search, i.e., the information retrieval process that exploits domain knowledge. In Chapter 2 we study how to improve performance during the result page construction, by dynamically generating document surrogates to be managed by a snippet caching system. In Chapter 3 we perform an analysis of the Europeana usage data and we extend a state-of-the-art recommendation algorithm in order to take into account the semantic information associated with submitted queries. Furthermore, we introduce a new technique for evaluating (with low human effort) the *relevance* and the *diversication* of the suggestions. In Chapter 4 we propose a learning to rank approach for improving the precision of the relatedness function. In Chapter 5 we formulate the problem of how to generate and select news recommendation explanations and we present and evaluate several techniques for solving the problem. Finally, in Chapter 6 we present some conclusions and discuss future work proposals.

Chapter 1

Background

Web Search Engines (WSEs) are the primary way users access the contents of the Web. By using a WSE, users are able to look for some information they might need either for work or for leisure: news about the latest football match, or about the last confidence vote of the Parliament. By querying a WSE users obtain a ranked list of Web documents that are hopefully highly related with their information needs. As above mentioned, WSEs can be considered part of a broader class of software systems, namely Information Retrieval Systems. However, the size, diffusion and complexity of the Web, WSEs need to cope effectively with several challenges.

In a paper overviewing the challenges in modern WSE design, Baeza-Yates *et al.* [8] state:

> *The main challenge is hence to design large-scale distributed systems that satisfy the user expectations, in which queries use resources efficiently, thereby reducing the cost per query.*

Therefore, the two key performance indicators in this kind of applications, in order, are: (i) the quality of returned results (e.g. handle quality diversity and fight spam), and (ii) the speed with which results are returned.

In this dissertation we discuss how query logs and the Web of Data may improve the current state of the art in WSEs both for what concerns *quality* and *efficiency*. Our work ranges over several topics in which we exploited knowledge and techniques from the historical usage data recorded in query logs and the Web of Data. This Chapter aims to provide a basic background for the main topics related to the thesis. It is organized as follows: in Section 1.1 we sketch the architecture of a WSE. In Section 1.2 we resume the state of the art in the analysis of the historical usage data. Finally In Section 1.3 we introduce Web of Data and Semantic Search.

1.1 Architecture of a Web Search Engine

A search engine is probably the most complicated software a company may develop. Consisting of tens of interdependent modules, it represents a big challenge in today's computer engineering world. Many papers and books sketch the architecture of web search engines. For example Barroso *et al.* [16] present the architecture of Google as it was in 2003. Other search engines are believed to have similar architectures. A Web search engine consists in three major applications [34, 98]: *crawling*, *indexing*, and *query processing*. Figure 1.1 shows the way the three applications interact and how the main modules of a web search engine are connected.

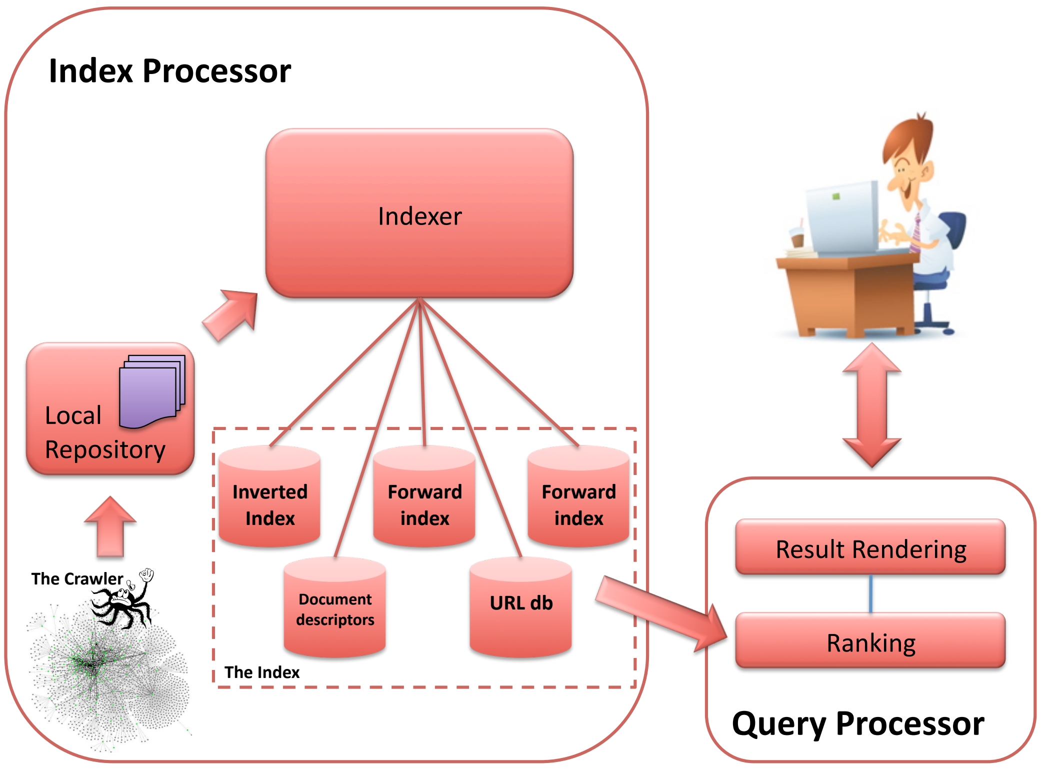


Figure 1.1: The typical structure of a Web search engine. From [129].

Web search engines get their data from different sources: the Web, image and video repositories (e.g. Flickr, or YouTube), social networks, etc. Crawling is the process responsible for finding new or modified pages on the Web and is made by several software agents called *crawlers* or *spiders*. In general, a crawler starts from a list of URLs, called *seeds*; then for each page, copies the page into the repository. Furthermore, the crawler fetches all URLs in the page and adds them to the list of the URLs to visit, called the *crawl frontier*. In particular, a crawler scours through hypertext pages searching for new documents, and detecting stale, or updated content. Crawlers store the data into a repository of content (also known as Web document cache), and structure (the graph representing how Web pages are interconnected). The latter being used, mainly, as a feature for computing static document rank scores (e.g. PageRank [114], or HITS [86]). In modern Web search engines, crawlers continuously run and download pages from the Web updating incrementally the content of the document cache.

The textual (i.e., hyper-textual) content is indexed to allow fast retrieval operations (i.e., query requests). The index (built by the Indexer) usually comprises of several different archives storing different facets of the index. The format of each archive is designed for enabling a fast retrieval of information needed to resolve queries. Indexes to support such text-based retrieval can be implemented using any of the access methods traditionally used to search over classical text document collections. Examples include suffix arrays, inverted indexes or inverted files, and signature files. In Web domain, *inverted indexes* are the index structure used. An inverted index is made by a *dictionary* D of terms. In the following, let D be the data structure, and $V = \{t_1, t_2, \cdots, t_m\}$ the *vocabulary* i.e., the set of m terms of the whole document collection. For each term, we have a list that records in which documents the term occurs. This list is called *posting list* and its elements (*postings*) contain the IDs of the documents containing the term (and often the position of the match in the document).

Usually in real systems the design is tailored to distribute requests through query servers to avoid peaking server response time [16]. In real-world search engines, the index is distributed among a set of query servers coordinated by a broker. Figure 1.2 shows the interactions taking place among query servers and the broker. The broker, accepts a query an user and distributes it to the set of query servers. The index servers retrieve relevant documents, compute scores, rank results and return them back to the broker which renders the result page and sends it to the user. The broker is usually the place where the activities of users (queries, clicked results, etc.) are stored in files called *query logs*. A module dedicated to analyze past queries is also usually available within the architecture components.

Searching is the goal of a Web search engine. When a user enters a query, the user's browser builds a URL (for example `http://www.google.com/search?q=diego+ceccarelli`). The browser, then, looks up on a DNS directory for mapping the URL main site address (i.e., `www.google.com`) into a particular IP address corresponding to a particular data-center hosting a replica of the entire search system. The mapping strategy is done accordingly to differ-

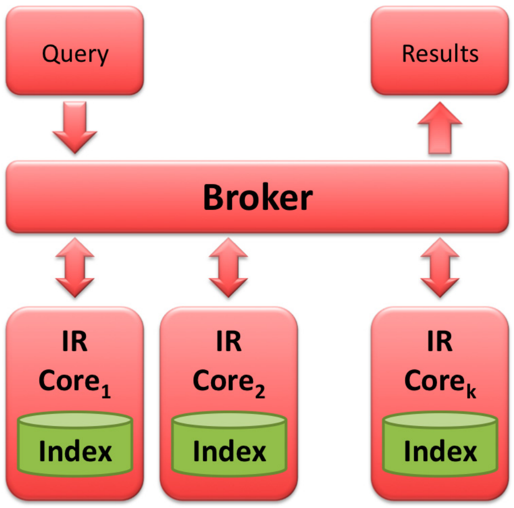


Figure 1.2: The typical structure of a distributed web search engine. From [129].

ent objectives such as: availability, geographical proximity, load and capacity. The browser, then, sends an HTTP request to the selected data-center, and thereafter, the query processing is entirely local to that center. The typical interaction between a user and a WSE thus starts with the formulation of a query q, representing the user's *information need*. Note that the information need is different from the query: the first is the topic about which the user desires to know more, while the second is what the user conveys to the computer in an attempt to communicate the information need. A query consists of a list of r terms

$$q = t_1, t_2, \ldots, t_r$$

Once the user has submitted her query q, document indexes are accessed to retrieve a single, uncategorized list with the most k relevant items appearing first

$$search(q) = \{d_1, d_2, \cdots, d_k\}$$

where k usually is ten.

Sorting documents by relevance requires computing for each document a *relevance score* with respect to the query. Formally, each relevance of each document d_i is evaluated through a scoring function

$$score(q, d_i) = s_i \qquad s_i \in \mathbb{R}^+ \cup \{0\}$$

that returns a score s_i. The highest is the score, the highest is the relevance of the document d_i for the query q.

There are several methods for computing scoring function. A popular scoring function is *Okapi BM25* [123], based on a *bag of words model*: the rank of a document is given by the query terms appearing into it, without taking into consideration the relationships between the query terms within the documents.

After documents are sorted by relevance, the top-k results are returned in the form of an HTML page to the user.

1.2 Query Log Mining

The uncertainty in users' intent is a key problem in Web search engines and, differently from smaller scale IR systems, Web IR systems can rely on the availability of a huge amount of usage information contained within past queries to solve it. Previously submitted queries represent, in fact, a very important mean for enhancing effectiveness and efficiency of Web search systems.

Query log mining is concerned with all the techniques aiming at discovering interesting patterns from query logs of Web search engines with the purpose of enhancing an online service provided through the Web. It can be seen as a branch of the more general Web Analytics [73] scientific discipline. According to the Web Analytics Association [6]:

> "*Web Analytics is the measurement, collection, analysis and reporting of Internet data for the purposes of understanding and optimizing Web usage*".

Also, it can be considered a special type of Web usage mining [136]. Web usage mining, in fact, refers to the discovery of user access patterns from Web usage logs. Furthermore, query log mining is not only concerned with the search service (from which queries usually come from) but also with more general services like, for instance, search-based advertisement, or web marketing in general [74].

Query logs keep track of information regarding interaction between users and the search engine. They record the queries issued to a search engine and also a lot of additional information such as the user submitting the query, the pages viewed and clicked in the result set, the ranking of each result, the exact time at which a particular action was done, etc. In general, a query log is comprised by a large number of records $\langle q_i, u_i, t_i, V_i, C_i \rangle$ where for each submitted query q_i, the following information is recorded: i) the anonymized identifier of the user u_i, ii) the timestamp t_i, iii) the set V_i of documents returned by the WSE, and iv) the set C_i of documents clicked by u_i.

From query log information it is possible to derive *Search Sessions*, sets of user actions recorded in a limited period of time. The concept can be further refined into: i) *Physical Sessions*, ii) *Logical Sessions*, and iii) *Supersessions.*

Physical Sessions: a physical session is defined as the sequence of queries issued by the same user before a predefined period of inactivity. A typical timeout threshold used in web log analysis is $t_0 = 30$ minutes. [118, 140].

Logical Sessions: a logical session [12] or *chain* [120] is a topically coherent sequence of queries. A logical session is not strictly related to timeout constraints but collects all the queries that are motivated by the same information need (i.e., planning an holiday in a foreign country, gathering information about a car to buy and so on). A physical session can contain one or more logical session. Jones *et al.* [82] introduced the concepts of *mission* and *goal* to consider coherent information needs at different level of granularity, being a goal a sub-task of a mission (i.e., booking the flight is one of the goal in the more general mission of organizing an holiday).

Supersessions: we refer to the sequence of all queries of a user in the query log, ordered by timestamp, as a supersession. Thus, a supersession is a concatenation of sessions.

Sessions are, thus, sequences of queries submitted by the same user in the same period of time. This data can be used to devise typical query patterns, used to enable advanced query processing techniques. Click-through data (representing a sort of implicit relevance feedback information) is another piece of information that is generally mined by search engines. In particular, every single kind of user action (also, for instance, the action of not clicking on a query result) can be exploited to derive aggregate statistics which are very useful for the optimization of search engine effectiveness. How query logs interact with search engines has been studied in many papers. Good starting point references are [129, 7, 124].

1.2.1 A Characterization of Web Search Engine Queries

The characteristics of query logs coming from some of the most popular Web search engines have been deeply studied [18, 19, 75, 76, 77, 78, 88, 100, 112, 113, 116, 132, 133, 158]. Typical statistics that can be drawn from query logs are: query popularity, term popularity, average query length, distance between repetitions of queries or terms.

The first contribution in analyzing query logs comes from Silverstein *et al.* [128]. Authors propose an exhaustive analysis by examining a large query log of the AltaVista search engine containing about a billion queries submitted in a period of 42 days. The study shows some interesting results including the analysis of the query sessions for each user, and the correlation among the terms of the queries. Similarly to other works, authors show that the majority of the users (in this case about 85%) visit the first page of results only. They also show that 77% of the users' sessions end up just after the first query. The query log analyzed contains a huge number of queries and account to 285 million users. Furthermore, the results shown in the paper are considered precise and general due to the high number of queries and users analyzed.

Lempel and Moran [93], and Fagni *et al.* [56] study the content of a publicly available AltaVista log. The log consist of 7,175,648 queries issued to AltaVista during the summer of 2001. No information referring the number of users logged is released. This second AltaVista log covers a time period almost three years after the first studies presented by Jansen *et al.* and Silverstein *et al.* Furthermore, the log is smaller than the first one. Indeed it represents a good picture of search engine users.

The distribution of query popularity [99], and term popularity have been shown to follow a power-law. This means that the number of occurrences y of a query or a term is proportional to $x^{-\alpha}$, where x is the popularity rank, and α is a real parameter measuring how popularity decreases against the rank. In a formula, $y = Kx^{-\alpha}$, where K is a real positive constant corresponding to the query with the highest popularity. Power-law distributions have the form of a straight line when plotted on a log-log scale.

Many different topics can be found in query logs. A very first result in categorizing queries is [132]. Authors show the percentage of queries submitted for each topic to the Excite search engine in 1997. Categorizing queries into topics is not a trivial task. Recent papers showing techniques for assigning labels to each query [63, 126, 153, 20, 21, 38] adopts a set of multiple classifiers subsequently refining the classification phase.

Terms are distributed according to a power-law as well (in particular a double-pareto log-normal distribution). In fact, the curve of term distribution fall sharply denoting that the most frequent terms are much more frequent that the rest of the terms. Figure 1.3 shows log-log plots of the term popularity distribution in the case of two query logs: Excite [99], and AltaVista [93].

An interesting statistics obtained from query logs is how terms co-occur. In [135], a follow-up of the work presented in [77], Spink *et al.* present the first fifty most frequently co-occurrent terms. Figure 1.1 shows how terms co-occur in queries without reflecting topic popularity. The majority of term pairs concern non-XXX topics while in the same analysis they found that XXX queries are highly represented. This highlight that, for some topics, people use more terms to search for precise information, while for other topics the same user need can be satisfied by short queries.

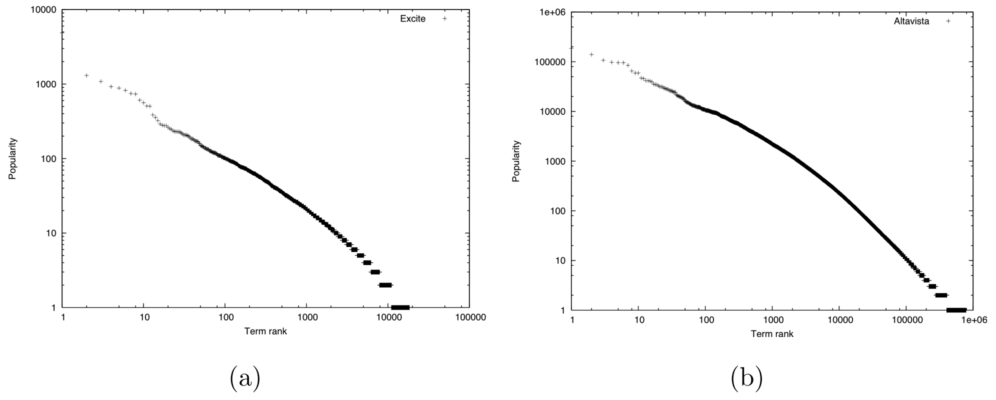


Figure 1.3: Terms popularity of a) the first 1,000 queries in the Excite [99], and b) AltaVista [93] logs.

and-and	6,116	of-and	690	or-or	501	women-nude	382	sex-pics	295
of-the	1,901	pictures-of	637	sex-pictures	496	pics-nude	380	north-carolina	295
pics-free	1,098	how-to	627	nude-pictures	486	of-department	365	free-teen	293
university-of	1.018	and-the	614	for-sale	467	united-states	361	free-porn	290
new-york	903	free-pictures	637	and-not	456	of-history	332	and-nude	289
sex-free	886	high-school	571	and-sex	449	adult-free	331	and-pictures	286
the-in	809	xxx-free	569	the-to	446	of-in	327	for-the	284
real-estate	787	and-free	545	the-the	419	university-state	324	new-jersey	280
home-page	752	adult-sex	508	princess-diana	410	sex-nudes	312	of-free	273
free-nude	720	and-or	505	the-on	406	a-to	304	chat-rooms	267

Table 1.1: List of the fifty most co-occurring terms (term$_1$–term$_2$, frequency) in the Excite log [135].

1.2.2 Search Sessions

After a first characterization of Web search engines' queries, here we focus on studying how users interact with search engine systems. What happen when a user has submitted a query and results are shown? How can be decided if a query has been correctly answered or if a user is satisfied by the search results? How people change queries if those have not produced satisfying users? The answers to these questions are very important in order to enhance the Web search engine performances.

In one of the first paper devoted to discovery user intent behind queries [36] Andrei Broder studies the goal a user wants to reach when submitting a query to a search engine. In the Broder's taxonomy a query can be either a *Navigational* query – were the immediate intent is to reach a particular destination (e.g. *Yahoo.com*, *America Airlines home page*, *Don Knuth's home page*); an *Informational* query – where the intent is to acquire some information assumed to be present on one or more web pages (e.g. *San Francisco* or *normocytic anemia*); a *Transactional* query – where the immediate intent is to perform some Web-mediated activity (e.g. *online music*, or *online flower delivery service*).

In order to evaluate the quality of search results it is interesting to look at how users interact with the search engine. For instance, it is interesting to extract and analyze user *search sessions* from query logs, and to derive *implicit measures* of quality explicitly tailored on search engine users.

Queries themselves are not always enough to determine the user intent. Furthermore, one of the key objectives of a search engine is to evaluate the quality of their results. Implicit measures that are available to log analysts are: the *click-through rate* – the number of clicks a query attract, *time-on-page* – the time spent on the result page, and *scrolling behavior* – how users interact with the page in terms of scrolling up and down; are all performance indicators search engines can use to evaluate their quality. How are the data recorded? Toolbars and user profiles surveyed directly from users are the main mean through which search engine companies record usage data diverse from those obtained by query logs.

A series of queries can be part of a single information seeking activity. Efforts have been spent on understanding the effects of request chains on the search engine side. The main goals of this analysis are to show how users interact with the search engine, and how they modify queries depending on what they obtain (in terms of results) from the search engine. Last but not least, efforts have been spent on understanding how users use multitasking and task switching on search sessions.

A first interesting result to highlight is how users interact with the search engine from a *page request* point of view. Many studies point out that users rarely visit result pages beyond the first one. Spink *et al.*analyze the Excite query log showing that about 78% of users do not go beyond the first page of results. Similar analysis on different query logs show the same behaviors [93, 56].

In order to obtain the results they need, users often try to reformulate queries (refine or modify) during their search sessions. Lau and Horvitz [91] study this behavior by categorizing queries. They propose seven categories:

New: A query for a topic not previously searched for by this user within the scope of the dataset (twenty-four hours);

Generalization: A query on the same topic as the previous one, but seeking more general information than the previous one;

Specialization: A query on the same topic as the previous one, but seeking more specific information than the previous one;

Reformulation: A query on the same topic that can be viewed as neither a generalization nor a specialization, but a reformulation of the prior query;

Interruption: A query on a topic searched on earlier by a user that has been interrupted by a search on another topic;

Request for additional results: A request for another set of results on the same query from the search service;

Blank queries: Log entries containing no query.

Authors apply the proposed categorization scheme within the Excite query log. Figure 1.4 shows the results obtained. In the majority of the cases most actions are either new queries or requests for additional informations. Furthermore, a large percentage of users (about 30%) issue a modification, (refinement, specification, or a reformulation) of a previously submitted query.

Many other works deal with the identification of users' search sessions boundaries. Work on session identification can be classified into: i) *time-based*, ii) *content-based*, and iii) *mixed-heuristics*, which usually combine both i) and ii). Time-based techniques have been extensively proposed for detecting meaningful search sessions due to their simplicity and ease of implementation. Indeed, these approaches are based on the assumption that time between adjacent issued queries is the predominant factor for determining a topic shift in user search activities. Roughly, if the time gap between two issued queries is lower than a certain threshold then they are also likely to be related.

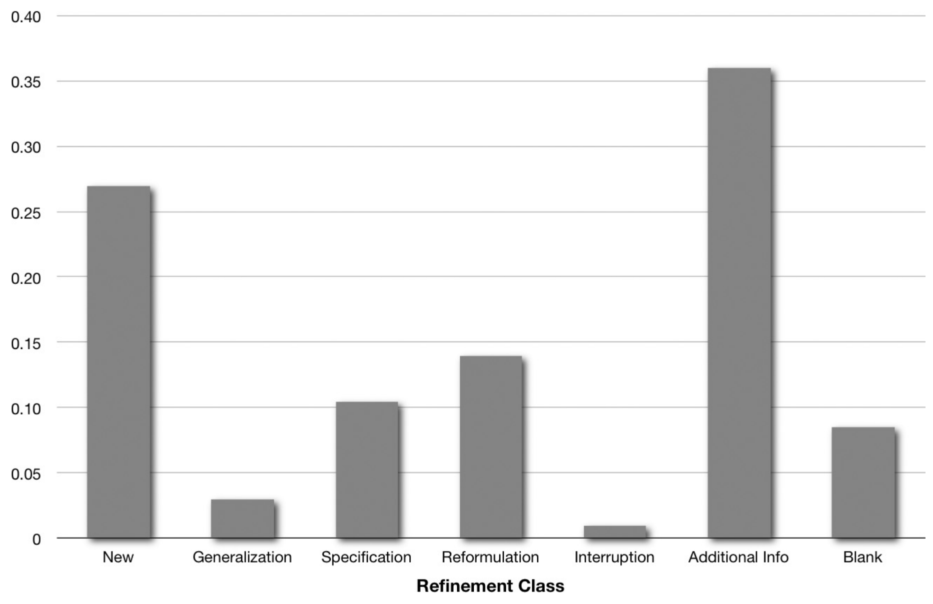


Figure 1.4: Summary of the categorization of 4,960 queries analyzed in [91].

Jansen and Spink [76] make a comparison of nine Web search engines transaction logs from the perspectives of *session length*, *query length*, *query complexity*, and *content viewed*. Here, they provide another definition of session, i.e. *search episode*, describing it as the period of time occurring from the first to the last recorded timestamp on the WSE server from a particular user in a single day, so that session length might vary from less than a minute to a few hours. Moreover, using the same concept of search episode, Spink *et al.* [134] investigate also *multitasking* behaviors while users interacting with a WSE. Multitasking during Web searches involves the *seek-and-switch* process among several topics within a single user session. Again, a user session is defined to be the entire series of queries submitted by a user during one interaction with the WSE, so that session length might vary from less than a minute to a few hours. The results of this analysis performed on a AltaVista query log show that multitasking is a growing element in Web searching. Finally, Richardson [121] shows the value of long-term WSE query logs with respect to short-term, i.e., within-session, query information. He claims that long-term query logs can be used to better understand the world where we live, showing that query effects are long-lasting. Basically, in his work Richardson does not look at term co-occurrences just within a search session that he agrees to be a 30 minutes time-window, but rather across entire query histories.

Boldi *et al.* [29] introduce the *Query Flow Graph* as a model for representing data collected in WSE query logs. They exploited this model for segmenting the query stream into sets of related information-seeking queries, leveraging on an instance of the Asymmetric Traveling Salesman Problem (ATSP).

Jones and Klinkner [82] argue that within a user's query stream it is possible to recognize particular hierarchical units, i.e., *search missions*, which are in turn subdivided into disjoint *search goals*. A search goal is defined as an atomic information need, resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals. Given a manually generated ground-truth, Jones and Klinkner [82] investigate how to *learn* a suitable binary classifier, which is aimed to precisely detect whether two queries belong to the same task or not. Among various results, they realize that timeouts, whatever their lengths, are of limited utility in predicting whether two queries belong to the same goal, and thus to identify session boundaries. Indeed, authors do not explore how such binary classifier could be exploited for actually segmenting users' query streams into goals and missions.

Lucchese *et al.* [95] devise effective techniques for identifying *task-based sessions*, i.e. sets of possibly non contiguous queries issued by the user of a Web search engine for carrying out a given task. In order to evaluate and compare different approaches the authors built, by means of a manual labeling process, a *ground-truth* where the queries of a given query log have been grouped in tasks. The analysis of this ground-truth shows that users tend to perform more than one task at the same time, since about 75% of the submitted queries involve a multi-tasking activity. Furthermore, authors formally define the *Task-based Session Discovery Problem* (TSDP) as the problem of best approximating the manually annotated tasks, and propose several variants of well-known clustering algorithms, as well as a novel efficient heuristic algorithm, specifically tuned for solving the TSDP. These algorithms also exploit the collaborative knowledge collected by *Wiktionary* and *Wikipedia* for detecting query pairs that are not similar from a lexical content point of view, but actually *semantically* related. The proposed algorithms is evaluated on the above ground-truth. Results show that it perform better than state-of-the-art approaches, because it effectively take into account the multi-tasking behavior of users.

1.3 The Web of Data

The *traditional Web* is usually modeled as a hyperlinked collection of *documents*. Each document is uniquely identified by a string of characters called **Uniform Resource Identifier** (URI). We can consider a document as an artifact produced by a human to share knowledge with other humans. The

Web of Data consists in replacing documents with data, in order to allow machines to consume them. The most important property of the Web of Data is that it is naturally organized around entities, and that each of these entities is uniquely identified by a URI.

Switching from documents to data is not trivial, and it implies a lot of challenges. Nevertheless, it deserves the effort. In the last years, the information and the services provided by means of the Web grew up rapidly. At the end of the last century, Internet became popular and many businesses started over it; the Web changed and became the Web 2.0 [53]. In 2005 Tim O'Really [111] stated that *data is the next Intel Inside*. In 2006, Szalay and Gray [138] called the Web an *exponential world* showing that the size of scientific data was doubling every year. The same phenomenon was also called *Data Deluge* [22]. Nowadays, many people publish their data on the web. Many scientific laboratories share the data collected from their instruments (sensors, telescopes, application logs, etc), while most popular social networks collect daily unprecedented amounts of user-generated data. All the data exposed by these stakeholders have a kind of *structure* that often is not strictly defined as in databases. For this reason they are usually referred as *semi-structured data* [1].

The ability to manage and analyze all these data becomes every day so important that the concept of *data science*[1] was created.

1.3.1 Semantic Search

Web of Data originates from the Semantic Web which was conceived as an extension of the Web. The goal of Semantic Web [24] is to add semantics to the documents on the Web in order to simplify the interaction between humans and computers. The focus of the Semantic Web is defining several common formats for encoding the data, in order to be able to easily combine and integrate data provided by different sources. The World Wide Web Consortium (W3C) defines[2] the Semantic Web as:

> *A common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF).*

One of the main issue is the management of the documents and in particular on how to semantically support the document retrieval, i.e., the semantic search. In the following, we focus on the *traditional* search, where the users do not have particular skill in any query language, so the retrieval system needs

[1] http://jobs.aol.com/articles/2011/08/10/data-scientist-the-hottest-job-you-havent-heard-of/
[2] http://www.w3.org/2001/sw/

to adopt simple interfaces and easy accessibility. These systems usually rely on keyword search or natural language interfaces. Despite these interfaces are easy for the users, keywords inserted by the users may be ambiguous. A classical example is a user searching for the term *apple*: what is the user looking for? The Apple Inc. company or the fruit? Understanding the meaning (*query intent*) of the query is then important and would improve the precision of search engines.

Mangold [97] defines semantic search as an information retrieval process that exploits domain knowledge. The topic is really broad, and several research communities proposed approaches for improving search by using semantic concepts. Many works have been proposed in Information Retrieval, Machine Learning, Semantic Web and Natural Language Processing communities, each one presenting the problems from its perspective and with its terminology. In particular the approaches differ by *i)* how the data is modeled *ii)* which data resources are used *iii)* how the users express their information need *iv)* how queries are matched against the collection and *v)* how results are ranked. It is worth to observe that Semantic Search is a broad topic and it would not be possible to review all the works in the area, so in the following we will only present works that are relevant for this dissertation, in particular we will not cover literature related to matching and ranking.

1.3.2 Modeling the Web of Data

Semi-structured data can be represented by using different models, that here we briefly resume:

OEM: Object Exchange Model: in this model, objects could be either atomic or complex. An atomic object is like a primitive type such as an integer or a string. A complex object is a set of (attribute, object) pairs;

XML: Extensible Markup Language: XML is very similar to HTML. It has tags, which identify elements. Tags can also contain attributes about elements. Attributes may be represented by new tags or may be atomic data such as text. The XML standard puts no restrictions on tags, attribute names or nesting relationships for data objects, making it well suited for describing semi-structured data;

JSON: JavaScript Object Notation "*The Fat-Free Alternative to XML*"[3] is a lightweight data-interchange format, with the goal of being *easy for humans to read and write and easy for machines to parse and generate.*

[3]http://www.json.org/fatfree.html

It has become popular to code data at Web scale because of its flexibility and minimality. It resembles the OEM model;

RDF: Resource Description Framework [**87**]: a resource description is composed of a *statements* about the resource. A statement is a triple simply consisting in a subject, a predicate, and an object. For example, one way to represent the notion "*Picasso was born in Málaga*" in RDF is by means of a triple having "Picasso" as subject, the predicate "was born in", and an object denoting "Málaga". Note that a set of RDF statements forms a direct labelled graph. Moreover, in this model objects are described by URIs that denote, and can be used to access, actual data on the World Wide Web.

1.3.3 Data Resources

Data resources are many and it is hard to provide a full list of the projects related to their creation. In the following we highlight the most important datasets publicly available. A comprehensive and well-organized list can be find in the work of Weikum and Theobald [155].

The most popular resource is Wikipedia[4], the largest encyclopedia ever conceived. Each page of Wikipedia can be considered a textual description of an entity; moreover Wikipedia pages may expose semi-structured information about entities: in the *infobox* tables (the pull-out panels appearing in the top-right of the default view of many Wikipedia articles, or at the start of the page for the mobile version); in the information about the categories which an entity belongs to; in images, geo-coordinates for places, and in the reference to external web links (e.g., official home pages). The project DBpedia [5], initially developed by the Free University of Berlin and the University of Leipzig, aims at providing all the data contained in Wikipedia in a semi-structured format, i.e., RDF. The English version of the DBpedia knowledge base currently describes 3.77 million entities.

DBpedia is part of the Linked Data [23] project, which aims at providing a "*best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF*". The project collects datasets available under open licenses, publishes them in RDF, and generates relations (*interlinks*) among them [26]. At the time of writing, Linked Data contains about 260 heterogeneous datasets and 50 billions link relations within them; Figure 1.5 shows a graphical representation of the datasets.

Another important collaborative project is Freebase, a large knowledge base where semi-structured data is collected and maintained by the members of a

[4]`http://en.wikipedia.org`

large community. Freebase (available under a Creative Commons Attribution License) contains approximatively 22 million entities and it is used by Google for enriching Web search result pages. The Wikipedia Foundation has a similar project called Wikidata.

YAGO [137] is a rich knowledge base, which contains around 10 million entities and 120 million facts about these entities. YAGO aggregates data coming from Wikipedia, (e.g., categories, redirects, infoboxes), WordNet (e.g., synsets. hyponymy) and GeoNames.

Finally, *WordNet* [108] is a *lexical database* for the English language. WordNet groups words with the same meaning into special sets called *synsets*. Furthermore WordNet provides the polysemy count of any word, and a *frequency* score that helps to identify the most common synset for a word. WordNet provides also the lexical relations among the synsets. For example it specifies if two synsets are *antonyms* (opposite of each other). The latest version of WordNet, released on December 2006) contains 155, 287 words organized in 117, 659 synsets for a total of 206, 941 word-sense pairs.

The largest dataset is however composed by the subset of Web documents on embedding metadata in RDF, RDFa, Microformats, or other formats. Usually, these data specify additional information about the document to which

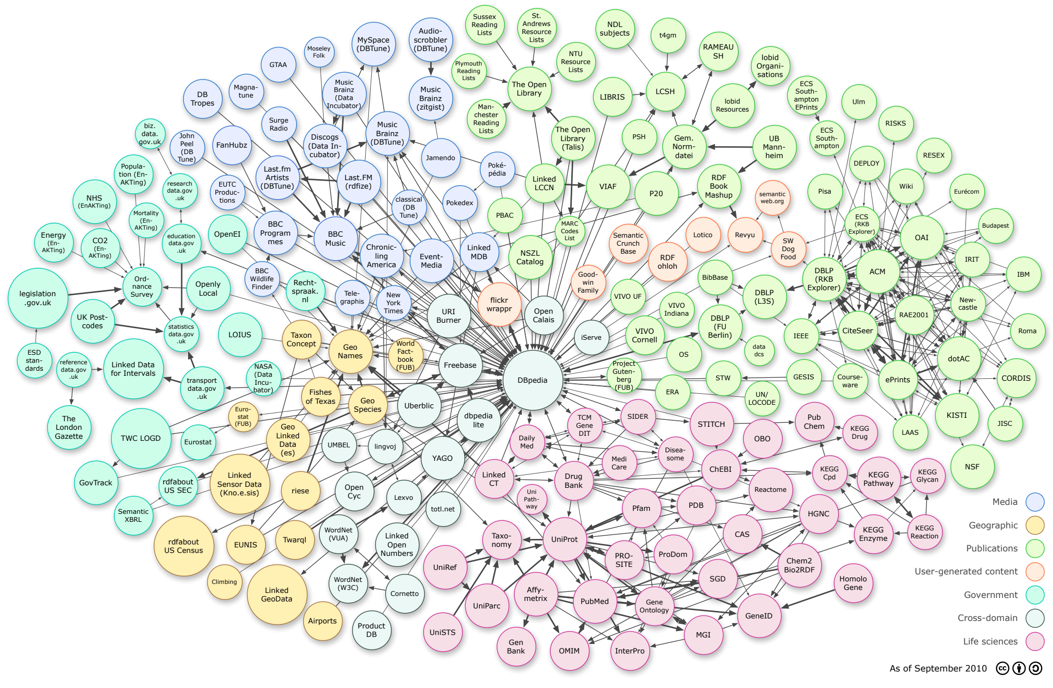


Figure 1.5: The datasets that are published in Linked Data format and are interlinked with other dataset in the cloud

they refer, such as the author or the publication date; they can also contain metadata at the entity-level. For example, Rotten Tomatoes[5] exposes rich annotations about movies and actors. In 2011, we introduced a Dataset for Entity-Oriented Search [40]: the dataset[6] was built from a fresh crawl of the Web performed by Sindice [145]. The crawl covers various domains: e-commerce, social networks, social communications, events, scientific publications, and more, and the data retrieved reflects these topics. Part of he data is coming from Linked Data datasets, such as DBpedia or Geonames. It is worth to note that the number of pages annotated on the Web is increasing: in 2012 Mika and Potter analyzed a large web crawl of Bing and found that about 30% of the pages in the crawl contain microformats [107].

Content providers add metadata to improve the quality of their pages, so that WSEs rank them higher. In order to promote this behavior and to improve the quality of published data, in 2011 the largest search companies (Yahoo!, Google, Bing and Yandex) promoted `schema.org`, a collection of schemas for modeling common entities such as people, organizations, places and so on. Similarly Facebook proposed the Open Graph schema in order to turn content into a social object and enable its multi-point connectivity to the social graph of the Facebook universe.

1.3.4 From raw to semi-structured data

The largest part of Web documents currently does not contain semantic annotations, and they are commonly modeled as a simple bag of words. One of the main approaches for enhancing search effectiveness on these documents consists in automatically enriching them with the most relevant related entities [115]. By enriching in the same way the queries submitted by users, query processing can exploit the semantic relations among the entities involved.

In literature, this automatic enrichment is known as the *Entity Linking*: given a plain text, the entity linking task aims to identify all the small fragments of text (in the following interchangeably called *spots* or *mentions*) referring to any *named entity* that is listed in a given knowledge base, e.g., Wikipedia. The ambiguity of natural language mentions makes it a non trivial task. The same entity can be in fact mentioned with different text fragments, e.g., "President Obama", "Barack Obama". On the other hand, the same mention may refer to different entities, e.g., "President" may refer to the U.S. president or to Alain Chesnais, the president of the Association for Computing Machinery.

[5]see `http://www.rottentomatoes.com/m/tron/`

[6]available at `http://data.sindice.com/trec2011/`

A typical entity linking system performs this task in two steps: *spotting* and *disambiguation*. The *spotting* process identifies a set of candidate spots in the input document, and produces a list of candidate entities for each spot. Then, the *disambiguation* process selects the most relevant spots and the most likely entities among the candidates. The *spotting* step exploits a given catalog of entities, or some knowledge base, to devise the possible mentions of entities occurring in the input. One common approach to address this issue is resorting to Wikipedia [110, 89]: each Wikipedia article can be in fact considered a named entity, and the anchor texts associated with Wikipedia links a rich source of possible mentions to the linked entity. The spotter can thus process the input text looking for any fragment of text matching any of the Wikipedia mentions, and therefore potentially referring to a entity. Indeed, the spotter should detect all the mentions and find all the possible entities associated with such mentions. The coverage of the source knowledge base and the accuracy of the spotter have in fact a strong impact on the recall of the entity linking system: all the entities related to every mention should be possibly detected and returned [47].

In NLP Research, Named Entity Recognition (NER) is a similar problem extensively studied in the state of the art. The main difference with Entity Linking is the fact that entities discovered in NER are represented as labeled phrases and are not uniquely identified by referring them to a knowledge base. Bunescu and Pasca [39] propose to link spots to Wikipedia articles (entities): in their work they exploit Wikipedia categories associated to each article to perform disambiguation. A similar approach is proposed by Cucerzan [50], whose approach creates for each article a vector containing the closest entities and categories. Section 4.3 provides a more detailed description of these approaches [110, 66, 68, 105, 57].

Beside entity linking for documents, an even more challenging issue is entity linking for queries. Annotating keyword queries with entities is difficult mainly for two reasons: i) query terms can have multiple meanings (polisemy), or dually, the same information need can be represented by using different words (synonymy) ii) while in the case of documents one can exploit the text close to the spot in order to perform disambiguation, a keyword query usually does not have an exploitable context. Several approaches to perform query-entity linking have been proposed. Huurnink *et al.* [72] exploit clicks to map queries to an in-house thesaurus. Other proposals [70, 106] perform approximate matching between the query and the *label* of the entity. Meij *et al.* [102] match the query and its n-grams against the entity labels and use machine learning to select the most related entities. Furthermore, in [103] they propose a method still based on machine learning that performs the mapping between the queries and DBpedia entities. Interestingly, they consider in their features set the user history which is important because it provides a context for disambiguation (we also rely on that feature in the approach presented in Section 3.6.1). Entity Retrieval (ER) is a well known problem, strongly related to entity linking: the task consists in ranking the entities in a knowledge base given a keyword query. Many approaches for ER were proposed and evaluated in several tracks proposed at the TREC [14] and at the INEX [52] conferences. In these tracks there are different problems about ranking entities (*e.g.*, rank only entities of a certain type, rank entities that best represent a query, etc.). The research on ER is really extensive [83, 147, 148, 151, 161]. For a complete survey of the methods we remind to Adafre *et al.* [2]. An interesting approach was proposed by Zwol *et al.* [148]. In their work they propose to rank facets (i.e., related entities) by learning a model through Gradient Boosted Decision Trees [58], and they evaluate the quality of their model by considering two different clicks models: i) click through rate, and, ii) click over expected clicks.

Chapter 2

Query Biased Snippets

2.1 Introduction

Nowadays a Web Search Engine (WSE) is a very complex software system [8]. It is well known that the goal of a WSE is to answer users' queries both *effectively*, with relevant results, and *efficiently*, in a very short time frame. After a user issues a query, the WSE actually runs a chain of complex processing phases producing the Search Engine Results Page (SERP). A SERP contains a list of few (usually 10) results. Each result corresponds to a Web page, and it contains the title of the page, its URL, and a text *snippet*, i.e. a brief text summarizing the content of the page.

The true goal is to *scale-up* with the growth of Web documents and users. The services offered by a WSE exploit a significant amount of storage and computing resources. Resource demand must be however kept as small as possible in order to achieve scalability. During query processing, document indexes are accessed to retrieve the list of identifiers of the most relevant documents to the query. Second, a view of each document in the list is rendered. Given a document identifier, the WSE accesses the document repository that stores permanently on disk the content of the corresponding Web page, from which a summary, i.e. the snippet, is extracted. In fact, the snippet is usually *query-dependent*, and shows a few fragments of the Web page that are most relevant to the issued query. The snippets, page URLs, and page titles are finally returned to the user.

Snippets are fundamental for the users to estimate the relevance of the returned results: high quality snippets greatly help users in selecting and accessing the most interesting Web pages. It is also known that the snippet quality depends on the ability of producing a *query-biased* summary of each document [143] that tries to capture the most important passages related with the user query. Since most user queries cannot be forecasted in advance, these snippets cannot be produced off-line, and their on-line construction is a heavy load for modern WSEs, which have to process hundreds of millions queries per day. In particular, the cost of accessing several different files (containing the Web pages) for each query, retrieved among terabytes of data, under heavy and unpredictable load conditions, may be beyond the capability of traditional filesystems and may require special purpose filesystems [146].

In this Chapter we are interested in studying the performance aspects of the snippet extraction phase and we devise techniques that can increase the query processing throughput and reduce the average query response time by speeding-up snippet extraction phase. In particular, we leverage on a popular technique used to enhance performance of computing systems, namely *caching* [117]. Caching techniques are already largely exploited by WSEs at various system levels to improve query processing, mainly for storing past queries and associated sets of results along with the query-biased snippets [55].

The basic technique adopted for Web search caching is to store the whole result page for each cached query. When a cache hit occurs, the result page is immediately sent back to the user. This approach perfectly fits queries being in the "*head*" of the power-law characterizing the query topic distribution. On the other hand, SERP caching is likely to fail in presence of personalization, that is when the engine produces two different SERPs for the same query submitted by two different users. Furthermore, it fails when a query has not been previously seen or it is a singleton query, i.e. it will not be submitted again in the future. Baeza-Yates *et al.* [9] report that approximatively 50% of the queries received by a commercial WSE are singleton queries.

Unlike query-results caching, snippets caching that is the focus of this Chapter, has received relatively low attention. The two research efforts closest to ours are those by Turpin *et al.* and Tsegay *et al.* [146, 144]. The authors investigate the effect of lossless and lossy compression techniques to generate documents surrogates that are statically cached in memory. They argue that complex compression algorithms can effectively shrink large collection of texts and accommodate more surrogates within the cache. As a trade-off, complex decompression increases the time needed by the cache to serve a hit, and are thus unlikely to decrease the average snippets generation time [146]. Lossy compression techniques produce surrogates by reordering and pruning sentences from the original documents. They reduce the size of the cached documents, still retaining the ability of producing high quality snippets. Tsegay *et al.* measured that in the 80% of the cases snippets generated from surrogates that are the 50% of the original collection size, are identical to the ones generated from the non-compressed documents [144].

To the best of our knowledge, this is the first research studying techniques for dynamically generating document surrogates to be managed by a snippet caching system. Furthermore, the surrogates generation method is completely new, since the sentence selection is based on the past queries submitted to the WSE. Our thesis is that the knowledge stored in query logs can help in building concise surrogates, that we call *supersnippets*, which we prove to be very effective for the efficient retrieval of high quality snippets.

The rest of this Chapter is organized as follows. In Section 2.2 and in Section 2.3 we provide an overview on text summarization techniques and on WSE caching techniques. In Section 2.4, by analyzing a query log, we show that the snippet generation process has a significant locality, which supports the use of caching techniques. Section 2.5 formalizes the architectural framework adopted and introduces two baseline snippet cache organization. In Section 2.6 we introduce the concept of *supersnippet*, and we propose a caching algorithm for building and maintaing supersnippets. Then, in Section 2.9 we report on the effectiveness of our proposed algorithm, while in Section 2.9.2 we show the performance results. Finally, in Section 2.10 we draw some conclusions.

2.2 Text summarization techniques

The techniques to generate good surrogates S_d of a document d (see Table 2.2) are those for summarizing text, and are based on the function $I(s)$ that estimates the document's information contained in each sentence $s \in d$.

There are several ways to compute $I(s)$:

Luhn Method: One of the first approach for text summarization is proposed by Luhn [96]. The method exploits the term frequencies tf in order to assign a weight to each term $t \in s$. A term t is considered as significant if tf overcome a threshold T, whose value depends on the number of sentences of d.

According to Lu and Callan [94], a sentence s is assigned a score $I(s)$ that depends on the so-called *clusters*, which are particular sequences of terms included in s. A cluster starts and ends with a significant term, and does not include long subsequences of insignificant terms. A cluster is scored on the basis of the ratio between the significant and the total number of terms included in the cluster. Finally, the importance of sentence s, i.e. its informative content $I(s)$, is given by the maximum score of all the cluster in s.

TF-IDF Method: Tsegay *et al.* [144] adopts a classical TF-IDF (Term Frequency Inverse Document Frequency) method to score the various sentences s in d. We can consider this method as an extension of the

Luhn one, which only exploits TF. Unfortunately, this technique needs a corpus for computing IDF score.

The formula adopted is the following:

$$I(s) = \sum_{t \in s - stopwords} (\log\ tf + 1) \times \log \frac{N}{df}$$

where tf is the raw count of a term t in the document d where $s \in d$, $N = |D|$ is the number of documents in the collection, while df is the count of documents that contain term t.

Position in text: Edmundson [54] proposes another weight-based method that combines the frequency with other heuristics like the position (usually, first sentences are a natural summary of a document), or the formatting (the titles contain meaningful sentences).

Different approaches based on machine learning and many other techniques have been proposed, we refer to the survey by Radev *et al.* [119] for an extensive review. effective method, based on a specific function learned from a training data, and defined in terms of multiple features extracted from s, has been devised by [104].

Query biased snippets $S_{d,q}$ are a particular form of text summarization, since the selected sentence of d not only depends on the sentence relevance, but also on a query q. In Tombros and Sanderson [143], each sentence has the following score:

$$s\_rel(s,q) = \frac{|s \cap q|^2}{|q|}$$

where $|s \cap q|$ is the number of query terms in the sentence s and $|q|$ the number of query terms: the more query terms a sentence contains, the higher is its score. The relevance of a sentence becomes a function $R(s,q)$ biased on query q:

$$R(s,q) = k_1\ s\_rel(s,q) + k_2\ I(s)$$

where k_1, k_2 are arbitrary weigh parameters.

2.3 WSE caching techniques

Query logs record historical usage information, and are a precious mine of information, from which we can extract knowledge to be exploited for a lot of different purposes [129]. The analysis of common usage patterns to

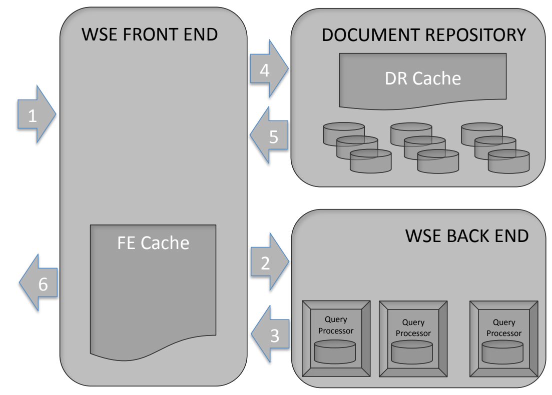


Figure 2.1: Sketch of a WSE architecture.

optimize system performance by means of caching techniques is one of the most important uses of such source of information.

The good efficiency of caching techniques in WSE implementation is motivated by the inverse power law distribution of query topics searched for [157, 129]. This high level of sharing justifies the adoption of a caching system for Web search engines, and several studies analyzed the design and the management of such server-side caches, and reported about their performance [99, 92, 55, 9].

The SERPs returned for frequently submitted queries are cached on the WSE FE (see Figure 2.1 to improve responsiveness and throughput, while index entries of terms commonly occurring in user queries are cached on BE to make faster query processing. Even partial results computed for popular sub-queries can be cached to further enhance performance.

Lempel and Moran propose *PDC* (Probabilistic Driven Caching), a query result caching policy based on the idea of associating a probability distribution with all the possible queries that can be submitted to a search engine [92]. PDC uses a combination of a Segmented LRU (SLRU) cache (for queries regarding the first page of results), and a heap for storing answers of queries requesting pages next to the first. Priorities are computed on the basis of historical data. *PDC* performance measured on a query log of AltaVista is very promising (up to 53.5% of hit-ratio with a cache of $256,000$ elements and 10 pages prefetched)

Fagni *et al.* show that combining static and dynamic caching policies together with an adaptive prefetching policy achieves even a higher hit ratio [55]. In their experiments, they observe that devoting a large fraction of entries to static caching along with prefetching obtains the best hit ratio. They also state the impact of having a static portion of the cache on a multithreaded caching system. Through a simulation of the caching operations they prove that, due to the lower contention, the throughput of the caching system can be doubled by statically fixing a half of the cache entries. This behavior is confirmed also in [9] where the impact of different approaches, such as static vs. dynamic caching, and caching query results vs. caching posting lists are studied.

Caching techniques can also be exploited by BE, by keeping in cache the uncompressed postings lists (or portions of them) associated with terms that are frequently and/or recently used. For a survey about these techniques, and the tradeoff between caching policies adopted by the FE and/or BE components of a WSE please refer to [9].

With regard to DR caching we can mention the idea of the Snippet Engine [144, 146], in which the original DR documents are replaced with their surrogates. This improves DR caching, since a surrogate takes up less space than the original one (size can range from 20% to 60% of the original document). Moreover, producing the snippets from a surrogate is faster than from the original document, because we have less sentences to compare with the query.

Turpin *et al.* [146] additionally compress the surrogate with a semi-static compression model, thus obtaining significant improvement in the performances. Moreover, Tsegay *et al.* [144] proves that query-biased snippets built from surrogate are, in most cases, identical to those built from the whole document. They also provide an approach named *simple go-back* (SGB): if a surrogate does not contain all the terms of the query, SGB builds the snippet from the original disk-stored document.

2.4 Motivations

In this Chapter we propose a new approach for generating query-biased snippets from concise surrogates of documents cached in main memory. Our thesis is that the knowledge stored in query logs can help in building concise surrogates that allow high-quality query-biased snippets to be efficiently generated. We thus start our study by analyzing a real-world query log, with the aim of understanding the characteristics and the popularity distribution of URLs, documents and snippets returned by a WSE.

In our experiments we used the MSN Search query log excerpt (RFP 2006 dataset provided by Microsoft). Such query log contains approximately 15 million queries sampled over one month (May 2006), and coming from a US Microsoft search site (therefore, most queries are in English). In particular, we consider two distinct datasets extracted from the query log: the first (which we call `D1`) contains $1,500,000$ queries and it is used for analysis purposes and

Dataset	Queries	Distinct Queries	Distinct Urls (top 10)
$\texttt{D1}_{train}$	$1,000,000$	$601,369$	$4,317,603$
$\texttt{D1}_{test}$	$500,000$	$310,495$	$2,324,295$
$\texttt{D2}_{train}$	$9,000,000$	$4,447,444$	$25,897,247$
$\texttt{D2}_{test}$	$4,500,000$	$2,373,227$	$12,134,453$

Table 2.1: Main characteristics of the samples of the MSN query log used for the experiments.

to motivate our approach, while the second (`D2`) contains $14,000,000$ queries and it is used to experimentally assess the performance of our technique.

Both datasets are further split in a training and a testing segment. Table 2.1 reports the main characteristics of the samples of the two datasets used for the experiments. For each query in the two datasets we retrieved the corresponding SERP via the Yahoo! Search BOSS API. Thus, for each query we stored the top-10 most relevant URLs and query-biased snippets as provided by Yahoo!. Moreover, we downloaded also the documents corresponding to all the URLs returned for the distinct queries occurring in the smallest dataset `D1`.

2.4.1 Analysis of the query logs

The first analysis conducted by using the `D1` dataset tries to answer a very simple question: "*is there temporal locality in the accesses to documents that are processed for extracting snippets to be inserted in the SERPs?*" If the answer to this question is positive, we can think to minimize the cost of such accesses by means of some caching technique. The question is somehow trivial to answer if we consider the high sharing of query topics studied in several papers (e.g., [55]). If the same query topics are shared by several users, the same should happen for the top results returned by the WSE for these queries.

We measured directly on our dataset the popularity distribution of documents occurring within the top-10 URLs retrieved for user queries in `D1`: as expected, the plot in Figure 2.2 shows that document popularity follows a power-law distribution. By looking with attention at the plot, we can already claim a presence of locality in the accesses to documents that goes beyond the locality in the user queries. We can in fact note that the top-10 most frequently accessed documents do not have the same popularity. This means that some of the top-10 documents returned for the most frequent query present in the log are returned among the results of some other query. An high sharing of

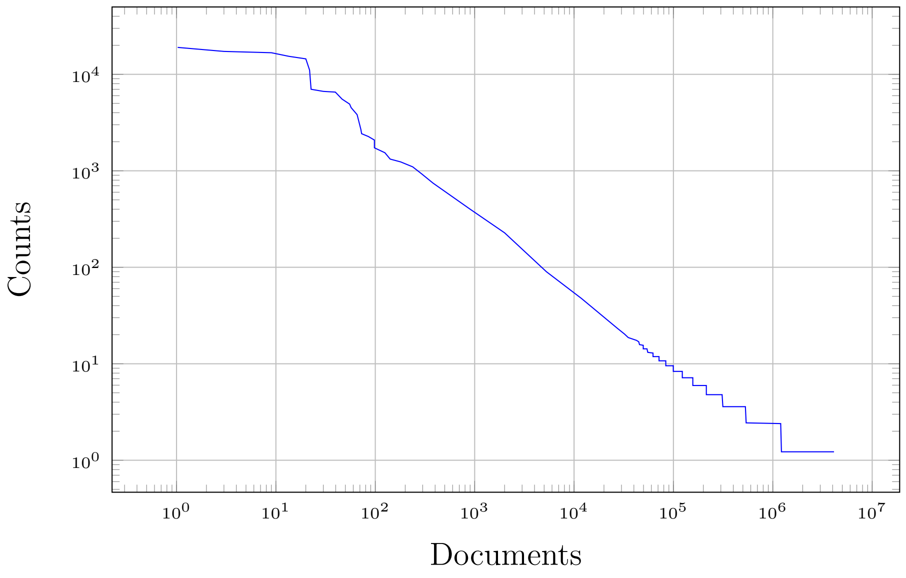


Figure 2.2: Popularity distribution of documents retrieved among the top-10 results (log-log scale).

URLs retrieved is thus present due to the well-known power-law distribution of query topics popularity, but also due to the sharing of the same URLs in the results' sets of different queries. From Table 2.1 we can see that `D1` contains about 912k distinct queries out of 1.5 millions, but only 6,640k distinct URLs occur among the 15 millions documents returned as top-10 results for these queries. Note that if all the results returned for the distinct queries would be completely disjoint, the distinct URLs should be 9,120k, about 27% more.

This high sharing in the URLs returned to WSE users surely justifies the adoption of caching techniques. Moreover, the sharing of URLs also among the results of different queries motivates the adoption of caching at the document level for speeding-up the generation of snippets, in addition to the query results cache commonly exploited in WSEs. Unfortunately, caching documents is very demanding in term of amount of fast memory needed to store even the most frequently accessed document.

However, other researchers already investigated the exploitation of lossless or lossy compression techniques for reducing memory demand, and proved that effective snippets can be generated also from document surrogates that retain less than half of the original document content [146, 144]. In this work we proceed further in the same direction, and propose an effective, usage-based technique for generating and managing in a cache much more concise document surrogates.

Having this goal in mind, next question we must answer is the following:

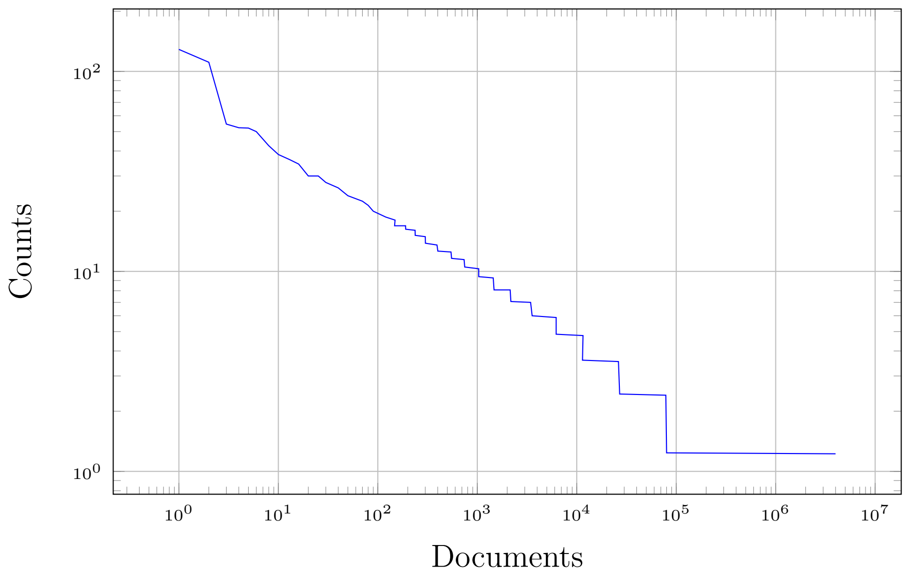


Figure 2.3: Number of distinct snippets (δ_u) per document distribution (log-log scale)

"how many different snippets have to be generated for a single document?" The more the number of different snippets generated by a single document, richer the content and larger the document surrogate from which these snippets can be generated.

Let us denote by δ_u the number of different snippets associated with URL u. Figure 2.3 shows that also δ_u follows a power-law distribution: a few URLs have several different snippets being generated and returned to the user. Indeed, about 99.96% of URLs have less than 10 snippets associated with them, and about 92.5% of URLs have just a single snippet associated with them. This means that the large majority of documents satisfies a single information need, and therefore just one snippet is usefully retrieved for them. On the other side of the coin, we have that about 8% of documents retrieved originate more than one snippet. These documents are potentially retrieved by different queries addressing different portions of the same document. To generate efficiently high-quality snippets for these documents may thus require to cache a richer surrogate.

Our thesis is that in most cases, even when the same URL is retrieved by different queries, the snippets generated are very similar. This happens, for example, in the case of query specializations and reformulations, synonyms, spell corrections, query-independent snippets such as those returned for navigational queries. To further investigate this point, in Figure 2.4 we show, for the 1,000 most frequently retrieved URLs, the number of distinct queries that retrieved them, and the corresponding number of distinct snippets generated. It is apparent that while many distinct queries retrieve the same document, only a small number of snippets, typically less than 10, is generated. This proves that when the same URL answers distinct queries, most of these different queries share exactly the same snippet.

Our proposal tries to exploit this interesting fact by introducing a novel snippeting strategy that allows to devise concise and effective document surrogates based on the past queries submitted to the WSE, and to exploit similarities among queries retrieving the same URL.

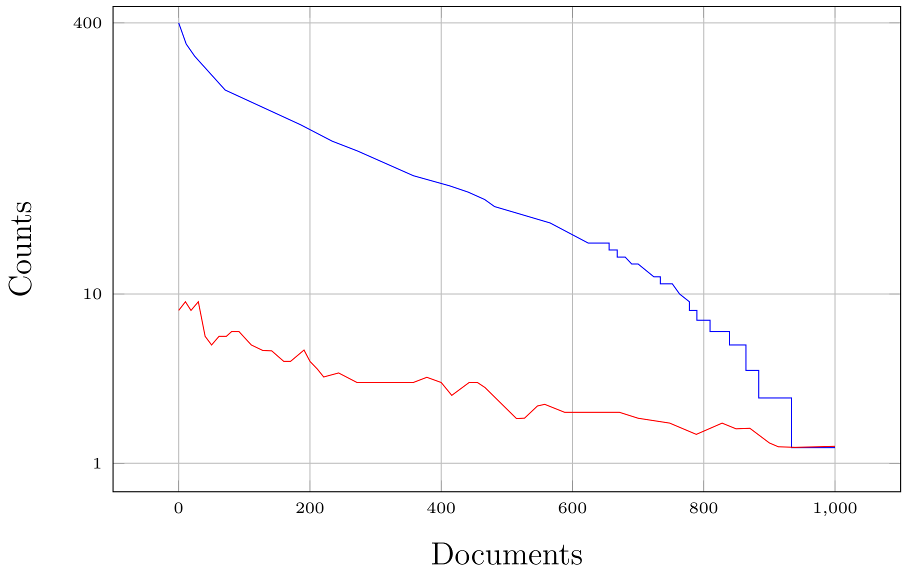


Figure 2.4: For the top-1,000 popular documents: number of distinct queries retrieving each document and number of distinct snippets associated with that document.

2.5 Architectural framework

The WSE subsystem responsible for query answering, as sketched in Figure 2.5, is usually made up of three cooperating components [17]:

WSE Front-End (FE) is in charge of managing the interactions with users: it receives a stream of queries, and returns the result pages built by exploiting the two other components.

The WSE Back-End (BE) for each query received from FE, extracts the top-k most relevant documents in its possibly distributed indexes, and returns to FE the corresponding document identifiers (docIDs).

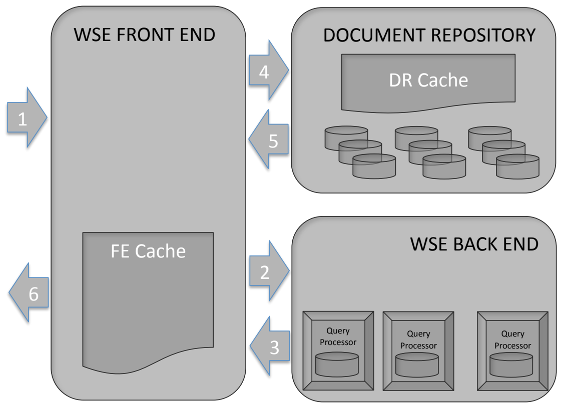


Figure 2.5: Sketch of the architecture of our testbed.

The WSE Document Repository (DR) is assigned the task of producing the result page by enriching and making user-understandable the ranked lists of docIDs returned by FE. More specifically, for each docID in the result set, DR retrieve the corresponding URL, and generates the snippet that summarizes, as shortly and clearly as possible, the query-biased content of the associated document.

In Figure 2.5, we illustrate the data flow. The user query q submitted to the WSE is received by FE (step 1). This forwards the query to BE (step 2) which is in charge of finding, via the distributed index, the most relevant documents to the query, and returns to FE the corresponding document identifiers (step 3). FE sends the query and the docID list to DR (step 4) which returns the corresponding query-biased snippets (step 5). Finally, the SERP is built by FE and returned to the user (step 6).

In order to improve throughput and save redundant computations, all the three components may exploit caching. FE caches SERPs of frequent queries, BE may cache portions of the postings lists of terms frequently occurring in user queries, and DR may cache frequently requested documents. In this work we focus on issues deriving from the use of caching on DR. For its evaluation we also consider the presence of a query result cache on FE, but the caching of postings lists, exploited by the various Query Processor in BE, is out of the scope of this work.

2.5.1 FE Cache

FE usually hosts a large cache storing results of recently submitted queries. This cache was proved to be very effective and capable of answering more than one third of all the queries submitted to commercial WSEs without involving the BE [9, 55, 92, 99]. The cache is accessed by supplying the current query as the key. Whenever a hit occurs, the FE may avoid to forward the query to BE, since the most relevant document are already present in the cache. The steps 2 and 3 are skipped, thus reducing the load at the BE. Depending on the organization of the FE cache, the DR may or may not invoked for generating the snippets. We distinguish between two possible caches types hosted on FE:

RCache$_{\mathsf{DocID}}$ stores in each entry the docIDs of the most relevant documents to a given query;

RCache$_{\mathsf{SERP}}$ uses larger entries to store all the information contained in a SERP, that is both the URLs and query-biased snippets.

In case of a miss in a RCache$_{\mathsf{DocID}}$ cache, the most relevant documents are requested from BE (steps 2,3), and inserted into the cache with key q for future reference. Since snippets and URLs are not stored, both in case of a hit and in case of a miss, these must be requested from DR (steps 4,5).

If FE adopts the RCache$_{\mathsf{SERP}}$, when a hit occurs, FE can retrieve the SERP for the query, and promptly return it to the user (step 6). The DR is not invoked, since the cache entries also stores URLs and snippets. In case of a miss, both the BE and DR must be invoked for producing the result page, which is also cached into RCache$_{\mathsf{SERP}}$. The type of result cache hosted on FE clearly affects the stream of requests processed by DR. If FE hosts an RCache$_{\mathsf{DocID}}$ cache, DR must process all the incoming WSE queries. It is worth noting that, from the point of view of the DR workload, this case is exactly the same as an architecture where no result caches are present. On the other hand, the presence of an RCache$_{\mathsf{SERP}}$ cache on FE strongly reduces the volume of requests issued to DR, since only the queries resulting in misses on RCache$_{\mathsf{SERP}}$ generate requests for URLs and snippet extractions.

2.5.2 DR Cache

Given a query q and a document identifier $docID$, DR retrieves the URL of the corresponding document and generates a query-based snippet.

The DR cache, which aims at reducing the number of disk accesses to the documents, is accessed by $docID$, while query q is used to select the best sentences from the document associated with $docID$. In literature, there are two main cache organizations:

DRCache$_{\mathsf{doc}}$ each cache entry contains an integral copy of a document;

$\mathsf{DRCache}_{\mathsf{surr}}$ each entry stores a surrogate of a document, which includes its most relevant sentences precomputed offline;

The $\mathsf{DRCache}_{\mathsf{doc}}$ works as a sort of OS buffer cache, speeding up the access to the most popular documents present on disk. Once the full document is retrieved a snippetting algorithm may easily generate a proper summary for the given query. However, the document size is significantly larger than the required snippet. Indeed, as shown in the previous section, for most documents only a single snippet is generated, and for the most popular documents, less than 10 snippets are sufficient to answer any query. Therefore, most of the cache memory is actually wasted.

$\mathsf{DRCache}_{\mathsf{surr}}$ tries to increase the number of entries cached in a given amount of memory by exploiting document surrogates which are shorter of the original document still retaining most of the informative content. Document surrogates are generated offline for each document, and are accessed and stored in $\mathsf{DRCache}_{\mathsf{surr}}$ in place of the original documents. The size of the surrogates induces a trade-off between cache hit-ratio and snippet quality. On the one hand, having small surrogates allow to cache more of them, thus allowing to increase the hit ratio. On the other, surrogates have to be sufficiently large in order to produce high quality snippets for all the possible queries retrieving a document.

This Chapter proposes a novel DR cache, namely $\mathsf{DRCache}_{\mathsf{Ssnip}}$, whose entries are particular document surrogates, called *supernippets*, whose construction and management are based on the past usage of the WSE by the user community. The idea behind $\mathsf{DRCache}_{\mathsf{Ssnip}}$ is to exploit the past queries submitted to the WSE in order to produce very concise document summaries, containing only those document sentences being actually useful in generating snippets. Also, the size of each summary may be very small when only a few sentences were needed to generate the snippets for the past queries, or it may be larger when there are many useful sentences, and thus several topics of interest for users, in each document. We prove that this cache organization allows for high hit ratio and high-quality query-biased snippets.

2.5.3 Answering queries

We can now discuss the flows of control and data occurring in the WSE subsystem depicted in Figure 2.5. In particular, the interactions among the components FE, BE, and DR, highlighted with numbered arrows in the figure.

Let q be a generic user query submitted to the WSE. q is received by FE (1), which looks up in the query result cache. We have to distinguish between the types of cache hosted on FE, either $\mathsf{RCache}_{\mathsf{SERP}}$ or $\mathsf{RCache}_{\mathsf{DocID}}$.

$\mathsf{RCache}_{\mathsf{SERP}}$ If FE hosts an $\mathsf{RCache}_{\mathsf{SERP}}$ cache, when a hit occurs, FE can promptly built the complete page of results answering q by using the information stored in the entry of the cache. Then FE can soon return the page to the requesting user (6).

If a miss occurs, the FE forwards q to BE (2), and waits for the set of the top-k docIDs that best match q (3). These k docIDS are then communicated to DR along with query q (4) to retrieve the k URLs and the associated query-biased snippets generated by DR. Such data are finally received by FE (5), and are used to prepare the result page returned to the user (6). The same page is inserted in the $\mathsf{RCache}_{\mathsf{SERP}}$ cache if dictated by cache policy.

$\mathsf{RCache}_{\mathsf{DocID}}$ In case FE hosts an $\mathsf{RCache}_{\mathsf{DocID}}$ cache, FE must interact with DR even when a hit occurs in the result cache. Only the list of the top-k docIDs that answers q is in fact stored in cache, and thus the associated URLs and snippets must always be requested from DR (4 and 5).

When a miss occurs, the list of the top-k docIDs answering q are still requested from BE (2 and 3), but, unlike the previous case, this list can be directly inserted with key q in $\mathsf{RCache}_{\mathsf{DocID}}$ for future reference.

The type of result cache hosted on FE clearly affects the stream of requests processed by DR. If FE hosts an $\mathsf{RCache}_{\mathsf{DocID}}$ cache, DR must process all the incoming WSE queries. It worth noting that, from the point of view of the DR workload, this case is exactly the same as an architecture where no result caches are present on the FE component.

On the other hand, the presence of an $\mathsf{RCache}_{\mathsf{SERP}}$ cache on FE strongly reduces the volume of requests issued to DR, since only the queries resulting in misses on the $\mathsf{RCache}_{\mathsf{SERP}}$ cache generate requests for URLs and snippet extractions.

2.6 DR Cache Management

As discussed in Section 2.5 and illustrated in Figure 2.6, the DR can host different types of caches, either $\mathsf{DRCache}_{\mathsf{doc}}$, $\mathsf{DRCache}_{\mathsf{surr}}$, or $\mathsf{DRCache}_{\mathsf{Ssnip}}$. To discuss the various cache organizations, it is worth introducing some notation, summarized in Table 2.2.

Let D be the set of all the crawled documents stored in the repository. Each $d \in D$ is composed of a set of sentences, i.e. $d = \{s_1, s_2, \cdots, s_n\}$. We denote by S_d, $S_{d,q}$, and $SS_{d,Q}$, the surrogate, snippet, and supersnippet of a document $d \in$

Symbol	**Meaning**
D	The collection of the crawled documents
d	A generic document ($d \in D$) associated with a distinct *DocID* and a distinct URL, composed of a set of sentences s_i, i.e., $d = \{s_1, s_2, \cdots, s_n\}$
s	A sentence composed of several terms
t	A term included in a sentence s
S_d	A surrogate that summarizes document d, and includes its most important sentences
$S_{d,q}$	The query-biased snippet of document d, which includes the sentences of d that are the most relevant to q
$SS_{d,Q}$	The supersnippet of document d, computed over the set Q of past queries
$I(s)$	The informative content of a sentence s
$R(s,q)$	The relevance of sentence s to query q

Table 2.2: Table of Notation

D. The generation of these summaries from d actually consists in selecting a suitable subset of its passages/sentences according to some relevance criterium. Note that the notation used recalls that a surrogate S_d is a query-independent excerpt of d, while a snippet $S_{d,q}$ depends on the query q. Finally, the content of a supersnippet $SS_{d,Q}$, $SS_{d,Q} \subseteq d$, depends on the set Q of past queries submitted to the WSE which retrieved document d. The lookup keys of all the three caches $\mathsf{DRCache}_{\mathsf{doc}}$, $\mathsf{DRCache}_{\mathsf{surr}}$, and $\mathsf{DRCache}_{\mathsf{Ssnip}}$ are the unique identifiers of the various documents (docIDs). Given a key docID associated to document d, then the possible contents of the entries of caches $\mathsf{DRCache}_{\mathsf{doc}}$, $\mathsf{DRCache}_{\mathsf{surr}}$, and $\mathsf{DRCache}_{\mathsf{Ssnip}}$ are, respectively, d, S_d, and $SS_{d,Q}$.

About the memory hierarchy of DR, below the cache level, which is located in *main memory*, we have the *disk*-stored document repository, which keeps the collection of all the indexed documents D and the associated URLs. Both the integral documents [1] and the URLS are directly accessible by supplying the corresponding docIDs. Figure 2.6 shows these disk-based levels of memory hierarchy in DR.

[1]In case DR hosts a $\mathsf{DRCache}_{\mathsf{surr}}$ cache, below the cache level we have surrogates S_d of all the documents, which are statically generated from each $d \in D$.

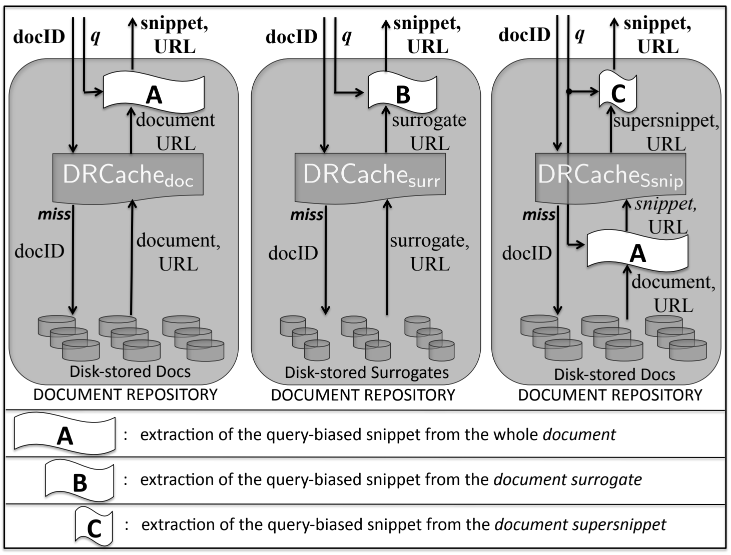

docID
q
snippet, URL
A
document URL
DRCache_doc
miss
docID
document, URL
Disk-stored Docs
DOCUMENT REPOSITORY
B
surrogate URL
DRCache_surr
surrogate, URL
Disk-stored Surrogates
C
supersnippet, URL
DRCache_Ssnip
snippet, URL
A : extraction of the query-biased snippet from the whole document
B : extraction of the query-biased snippet from the document surrogate
C : extraction of the query-biased snippet from the document supersnippet

Figure 2.6: Three possible DR organizations, hosting either $\mathsf{DRCache_{doc}}$, $\mathsf{DRCache_{surr}}$, or $\mathsf{DRCache_{Ssnip}}$. Note the activities labelled **A**, **B**, and **C**, whose size models the associated computational load, which in turn depends on the size of the input: either documents d, surrogates S_d, or supersnippets $SS_{d,Q}$.

2.7 Surrogates, snippets, and supersnippets

In this section we discuss how the best sentences to include into S_d, $S_{d,q}$, and $SS_{d,Q}$ can be devised. For the sake of simplicity hereinafter we will assume surrogates, snippets, and supersnippets composed by sets of sentences. The definitions and metrics discussed can be however trivially adapted to passages or any other piece of text of fixed or variable size.

2.7.1 Surrogates

Several proposals appeared in the literature regarding the generation of concise surrogates retaining most of the informative content of a document. In Section 2.2 we survey some of these methods. They are generally based on a function to evaluate the informative content $I(s)$ of each sentence $s \in d$, and on a constraint on the size of the surrogate, e.g. the surrogate has to contain a fraction x, $0 < x < 1$, of all the sentences of the original document. Given $I(s)$ and x, the best surrogate S_d for d is the one that maximizes:

$$S_d = \arg\max_{\sigma \subseteq d} \sum_{s \in \sigma} I(s), \qquad \text{where} \quad |\sigma| \leq \lfloor x \cdot |d| \rfloor$$

2.7.2 Query-biased Snippets

Query biased snippets $S_{d,q}$ are generated for all the top-k documents retrieved by a WSE for query q, by means of a function $R(s, q)$, which measure the relevance to q of the various sentences $s \in d$. Hence, given function $R(s, q)$ and the maximum size sz of a snippet, expressed in terms of the number of sentences, the best snippet $S_{d,q}$ for a query q and a document d is the one that maximizes:

$$S_{d,q} = \arg\max_{\sigma \subseteq d} \sum_{s \in \sigma} R(s, q), \qquad \text{where } |\sigma| \leq sz \tag{2.1}$$

In our experiments, a query-based snippet, extracted from either a document d or a surrogates/supersnippet of d, is simply obtained by applying Equation (2.1) with a relevance score computed as follows:

$$R(s, q) = \frac{|\overline{s} \cap \overline{q}|^2}{|\overline{q}|}$$

where $\overline{s}$ and $\overline{q}$ are the bags of terms occurring in s and q.
This corresponds to ranking all the sentences and selecting the top-k ranked sentences. In our experiments, we set $k = 3$.

2.7.3 Supersnippets

Given a set Q of queries submitted in the past to the WSE, e.g., the set of queries recorded in a query log, let us consider the set Q_d, $Q_d \subseteq Q$ of past queries having document d in the result set returned to the user. Set Q_d contains all the past queries for which document d was considered relevant by the WSE ranking function. By using a snippeting technique as the one sketched above, the set $U_d = \bigcup_{q \in Q_d} S_{d,q}$ can be easily built. While a document d can contain several different sentences, the above set only contains those sentences of d that are likely to be of some interest to past WSE users. U_d can be thus considered a surrogate of d, whose construction, and thus the relative criterium for sentence selection, is *usage* based. By analyzing a query log spanning a long time period, we have shown that the number of different snippets for a given document is in most cases very small. Thus, we can expect also the size U_d to be small. Moreover, some redundancies can be found in U_d because some sentences are very similar to others.

Given a similarity score $sim(s, s')$ between sentences s and s' of a document, we define the *supersnippet* $SS_{d,Q}$ as the set of *medoids* [85] obtained by a k-medoids clustering, run on the set of sentences contained in U_d that minimizes inter-cluster similarities and maximizes intra-cluster similarities. The number of clusters, i.e. k, itself is the optimal one and it is not a-priori fixed.

A careful reader may note some resemblance of the term "*supersnippet*" to "*superstring*". In the "*superstring problem*", given a input set of symbol strings, the goal is to find the shortest string that includes as sub-strings all the others. Our supersnippet is made up of several strings, but, in a similar way as superstring, we aim to reduce its size by eliminating redundant sentences.

The above definition of a supersnippet cannot be used in practice. The main reason is that we need an efficient algorithm for constructing them. Therefore, we adopt the following greedy strategy. We read queries in the same order as they arrive. For each query we retrieve the corresponding SERP. For each snippet in the result set, we consider each sentence contained within. In the corresponding supersnippet, we add a sentence only if its similarity with all the previously added ones is below a certain threshold. In our experiments, we used the Jaccard index measured on sentence terms as similarity metric, and set the threshold to 0.8, i.e. less than 80% of the terms in the candidate sentence appear in the sentences already added to the supersnippet.

Furthermore, in our experiments we bound the maximum number of sentences forming a supersnippet. Several approaches can be adopted to limit to sz sentences the maximum size of a supersnippet. A first *static* approach consists in sorting all the sentences of $SS_{d,Q}$ by frequency of occurrence within snippets $S_{d,q}, \forall q \in Q_d$. The first sz sentences can in this case be considered as the most relevant for document d on the basis of their past usage.
Another possibility is to consider a supernippet as a fixed size *bin*. This bin is filled with new sentences occurring in snippets retrieved for the queries in Q_d. When the bin is completely filled, and another sentence has to be inserted, a replacement policy can be used to evict from the supersnippet a sentence to make place for the new one.

In this work we followed this second possibility and adopted a simple LRU policy to always evict the *least recently used* sentence first. We preferred to adopt a dynamic supersnippet management policy to avoid the aging of statically-generated supersnippets as time progresses and interests of users change. We leave as a future work the analysis of the effect of aging on static supersnippets, as well as the study of the performance of alternative sentence replacement policies. The pseudocode reported in Algorithm 1 sketches the simple management of $\mathsf{DRCache_{Ssnip}}$. Function $Lookup(q, DocID)$ is called to lookup for a supersnippet of d in the cache, and implements the LRU management of $\mathsf{DRCache_{Ssnip}}$ entries. $Lookup(q, DocID)$ calls function $UpdateSS(SS_{d,Q}, DocID)$ to update as above described the sentences stored within $SS_{d,Q}$.

Unlike other approaches, where the document surrogates are statically determined, our supersnippets are thus dynamically modified on the basis of their usage. Their contents change over time, but depend on the query-biased snippets extracted in the past from the original documents to answer to user queries. As a consequence of this approach, in our supersnippet cache hosted in the DR, a miss can occur also when the docID key is matched, but the query-biased snippet returned for it results to be of *low quality.* Even in this case a miss penalty must be paid, and a query-biased snippet must be extracted from the integral document stored on disk. This new snippet is then used to update the cached supersnippet on a LRU basis.

2.7.4 Query-biased Snippets: Documents vs. Surrogates

In Figure 2.6 we show that when DR hosts either $\mathsf{DRCache}_{\mathsf{surr}}$ or $\mathsf{DRCache}_{\mathsf{Ssnip}}$, a query-biased snippet is extracted from a cache entry that contains a surrogate of d. More specifically, according to Equation (2.1), we extract the query-biased snipped $S_{d',q}$ from d', where $d' = S_d$ or $d' = SS_{d,Q}$. While S_d is a proper surrogate of d, the supersnippet $SS_{d,Q}$ is a special surrogate whose construction is query-log based.

2.8 Cache management policies

In this section we discuss the three possible organizations of DR illustrated in Figure 2.6, and the relative cache management issues. The first two caches $\mathsf{DRCache}_{\mathsf{doc}}$ and $\mathsf{DRCache}_{\mathsf{surr}}$ are pretty standard. The docID is used as look-up key. If no cache entry corresponding to the key exists, a miss occurs. Then the lower memory level (disk-based) is accessed to retrieve the associated URL, along with either the document or the surrogate. These caches can be *static*, i.e. the cache content does not change during the cache utilization (although it can be updated by preparing off-line a new image of the cache), or *dynamic.* In the last case, many replacement policies can be used in order to maximize the hit ratio, trying to take the largest advantage possible from information about recency and frequency of references.

Note that for both caches, once retrieved the matching cache entry, the query-biased snippet must be extracted from the content of this entry. The two activities, labeled as **A** and **B**, correspond to snippet extractions. The cost of extracting the snippet from the document surrogate (**B**) is smaller than from the whole document (**A**). In general, both the *hit cost* and the *miss penalty* of $\mathsf{DRCache}_{\mathsf{surr}}$ are smaller than the ones of $\mathsf{DRCache}_{\mathsf{doc}}$.

$\mathsf{DRCache}_{\mathsf{Ssnip}}$ is more complex to manage. As for the other two caches, docIDs are used as look-up keys. If no cache entry corresponding to the key exists, a miss occurs. In this case the miss penalty is more expensive, since the query biased snippet must be first extracted from the document (see activity labeled as **A**), and the copied to a cache entry.

$\mathsf{DRCache}_{\mathsf{Ssnip}}$ needs to exploit a special cache policy, i.e. a special rule, to determine whether a request can be satisfied using the cached supersnippet. More specifically, even when the look-up of a docID succeeds, the query-biased extraction (see activity labeled as **C**) can fail, because the snippet results of *low quality* with respect to the query. In this case we have a *miss of quality*. This leads to accessing to the disk-stored repository to retrieve the whole document. The query biased snippet is thus extracted from the document (see activity labeled as **A**). This query-biased snippet is finally used to update the corresponding cache entry, which contains the document supersnippet.

2.9 Cache Effectiveness

The component **C**, described in the previous section and illustrate in Figure 2.6, plays an important role for $\mathsf{DRCache}_{\mathsf{Ssnip}}$. For each incoming query, it must be possible to decide whether or not a cached supersnippet contains the sentences needed to produce an high quality snippet. If this is not the case, a miss of quality occurs, and the disk-based document repository is accessed.

To this end, a quality function must be defined, which, given the query q and the supersnippet $SS_{d,q}$ is able to measure the goodness of $SS_{d,q}$. In the following we propose and analyze a simple quality metrics, and we compare the snippets produced by $\mathsf{DRCache}_{\mathsf{Ssnip}}$ with those returned by a commercial WSE. Indeed, we compare the quality of the snippets generated by our caching algorithm, with the quality of the results returned by the Yahoo! WSE. Note that the Yahoo! WSE is also used to simulate the underlying BE and DR, thanks to the property of $\mathsf{DRCache}_{\mathsf{Ssnip}}$ of being independent from the snippet generation engine.

The proposed metrics takes into account the number of terms in common between the query q and the generated snippet $S_{d,q}$, and it is illustrated in the following section. This metrics can be used directly by $\mathsf{DRCache}_{\mathsf{Ssnip}}$ to detect quality misses.

2.9.1 Snippet quality measure

An objective metrics measuring the quality of the snippet $S_{d,q}$ for the query q can take into account the number of terms q occurring in $S_{d,q}$: the more query terms included in the sentences of a snippet, the greater the snippet goodness. More specifically, we adopt the scoring function proposed in [143]:

Algorithm 1 DRCache$_{\mathsf{Ssnip}}$ caching algorithm.

```
 1: function LOOKUP($q$, $DocID$)
 2:     if $DocID \in \mathcal{C}$ then                          ▷ cache hit
 3:         $SS_{d,Q} \leftarrow \mathcal{C}(DocID)$              ▷ retrieve the supersnippet
 4:         $S_{d,q} \leftarrow$ SNIPPET($q, SS_{d,Q}$)           ▷ generate a snippet
 5:         if $\neg$HIGHQUALITY($S_{d,q}, q$) then                ▷ quality miss
 6:             $S_{d,q} \leftarrow$UPDATESS($SS_{d,Q}$, $DocID$, $q$)
 7:         end if
 8:         $\mathcal{C}$.MOVETOFRONT($SS_{d,Q}$)                 ▷ LRU update
 9:     else                                                      ▷ cache miss
10:         $SS_{d,Q} \leftarrow \emptyset$
11:         $S_{d,q} \leftarrow$UPDATESS($SS_{d,Q}$, $DocID$, $q$)
12:         $\mathcal{C}$.POPBACK()
13:         $\mathcal{C}$.PUSHFRONT($SS_{d,Q}$)                   ▷ add a new supersnippet
14:     end if
15:     return $S_{d,q}$                                          ▷ Return the snippet
16: end function

                          ▷ Update the supersnippet with a new snippet for $q$
17: function UPDATESS($SS_{d,Q}$, $DocID$,$q$)
18:     $d \leftarrow$ GETDOC($DocID$)                            ▷ access repository
19:     $S_{d,q} \leftarrow$ SNIPPET($q, d$)
20:     for $s \in S_{d,q}$ do                                    ▷ update the supersnippet
21:         $s' \leftarrow \arg\max_{t \in SS_{d,Q}} sim(s, t)$
22:         if $sim(s, s') \geq \tau$ then
23:             $SS_{d,Q}$.MOVETOFRONT($s'$)                      ▷ LRU update
24:         else
25:             $SS_{d,Q}$.POPBACK()
26:             $SS_{d,Q}$.PUSHFRONT($s'$)                        ▷ add a new sentence
27:         end if
28:     end for
29:     return $S_{d,q}$
30: end function
```

$$score(S_{d,q},\ q) = \frac{|\overline{S}_{d,q} \cap \overline{q}|^2}{|\overline{q}|} \tag{2.2}$$

where $\overline{S}_{d,q}$ and $\overline{q}$ correspond to the sets of terms appearing, respectively, in the sentences of $S_{d,q}$ and in the query q.

We claim that a snippet $S_{d,q}$ is of *high quality* if $q \subseteq S_{d,q}$ for queries with one or two terms, or if $S_{d,q}$ contains at least two query terms for longer queries. It can easily shown that our good quality criteria is met whenever $score(S_{d,q}, q) \geq 1$.

Proposition 1. *Let $\overline{q}$ and $\overline{S}_{d,q}$ be the sets of terms of q and $S_{d,q}$, respectively. If $score(S_{d,q},\ q) \geq 1$ then either $|\overline{q}| \leq 2$ and $\overline{q} \subseteq \overline{S}_{d,q}$, or $|\overline{q}| > 2$ and $|\overline{S}_{d,q} \cap \overline{q}| \geq 2$.*

Proof. If $|\overline{q}| = 1$ then $score(S_{d,q},\ q) \geq 1$ trivially implies that $\overline{q} \subseteq \overline{S}_{d,q}$.

If $|\overline{q}| > 1$ then $score(S_{d,q}, q) \geq 1$ means $|\overline{S}_{d,q} \cap \overline{q}|^2/|\overline{q}| \geq 1$. By a simple arithmetical argument we have that $|\overline{S}_{d,q} \cap \overline{q}|^2 \geq |\overline{q}| \geq 2 \Rightarrow |\overline{S}_{d,q} \cap \overline{q}| \geq \sqrt{2} \approx 1.4$. Since $|\overline{S}_{d,q} \cap \overline{q}|$ must be an integer value, then we can conclude that $|\overline{S}_{d,q} \cap \overline{q}| \geq 2$. This also is equivalent to $\overline{q} \subseteq \overline{S}_{d,q}$ for the case $|\overline{q}| = 2$. □

This rough but fast-to-compute score measure is exploited in the implementation of DRCache$_{\mathsf{Ssnip}}$ to evaluate on-the-fly the goodness of the snippets extracted from the document supersnippets stored in the cache. If the score is lower than 1, the cache policy generates a *quality miss*.

We apply this quality score to the snippets generated by the Yahoo! WSE for the top 10 document returned to the *D1* query log. It is worth noting that the fraction of these snippets resulting of good quality – i.e., with score greater or equal to 1 – is high, but remarkably lower than 100%: the high quality Yahoo! snippets resulting generated from the *D1* query log are about 81% only. In fact, the snippets returned by Yahoo! do not always contain the terms of the query. This is not always due to badly answered queries. For example, this may happen when a query matches some metadata (e.g., the URL name) and not the snippet's sentence, or when only a few terms of a long query are actually included in the document retrieved. Even misspells or variations in navigational queries aiming at finding well-known portal websites are typical example of this phenomenon. A deeper study of this issue is however out of the scope of this work.

To compare with DRCache$_{\mathsf{Ssnip}}$, we set up the following experiment. First we created the set of supersnippets for the documents being returned by the system. Each query in the *D1/training* query log was submitted to the Yahoo! WSE, and we collected the returned documents and their snippets. For each document, the corresponding supersnippet was generated by picking the most representative sentences on the basis of the returned snippets as describe in the previous section. We run different experiments by allowing a maximum number of 5 and 10 sentences in each supersnippets. Note that the resulting supersnippets have varying size, which depends on the number of distinct queries related with a document, and with the number of its sentences included in the snippets related with those queries. Therefore, many supersnippets have a number of sentences smaller than the allowed maximum.

Second, the *D1/test* was used to assess the quality of the generated supersnippets. Similarly as above, for each query in *D1/test* we collected the corresponding results, by ignoring those documents for which a supersnippet was not generated. Note that in case of previously unseen documents, the DRCache$_{\mathsf{Ssnip}}$ caching algorithm would retrieve the correct snippet from the document repository, therefore it makes sense to discard those documents for the evaluation of the supersnippets goodness. For each collected document we computed the score of the snippets generated from the corresponding supersnippet, and, in particular, we measured the fraction of them being of high quality.

In Figure 2.7 we report the fraction of high quality snippets generated by the Yahoo! WSE and by the exploiting supersnippets of maximum size 5 and 10 sentences. Values are averaged over buckets of 100,000 consecutive (according to the query log order) snippets.

The percentage of good snippets generated by Yahoo! is only slightly higher[2]. Of course, the quality measured for the supersnippeting technique increases with the number of sentences included. The score of $score(S_{d,q}, q)$ averaged over all the snippets in *D1/test* is 1.42, 1.39, and 1.38 respectively for the Yahoo! ground truth, and our technique exploiting supersnippets limited to 10 and 5 sentences. In conclusion, the proposed supersnippeting technique produces snippets of high quality, very close to those generated by the Yahoo! search engine. Indeed, with only 5 sentences at most for each supersnippet, a fraction of 81.5% of the snippets from the whole *D1/test* are of high quality, with a very small loss (0.8%) compared with Yahoo!, where this fraction rises up to 82.3%.

2.9.2 Hit Ratio Analysis

In the previous section we have analyzed the quality of snippets generated using our novelapproach. In this section we evaluate the effectiveness of the various DRCache strategies (i.e. DRCache$_{\mathsf{doc}}$, DRCache$_{\mathsf{surr}}$, and DRCache$_{\mathsf{Ssnip}}$) when used in combination with a FECache. Four different DRCache sizes have

[2]The peak measured for the snippets in the fifth bucket plotted in Figure 2.7 is due to the presence in the dataset of some bursty queries caused by specific events regarding public persons.

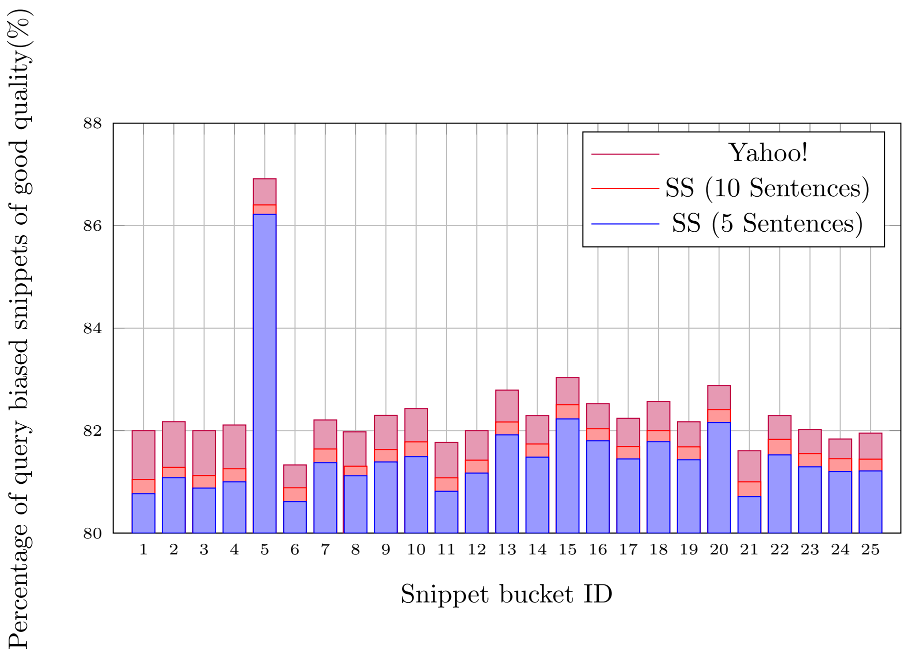


Figure 2.7: Average fraction of high quality snippets returned by Yahoo! and $\mathsf{DRCache}_{\mathsf{Ssnip}}$ measured for consecutive groups of queries in *D1/test*.

been experimented: 256, 512, 1024, and 2048MByte. Our experiments are run on the D2 dataset. We warm up the various caches by executing the stream of queries from D2/training and we evaluate the various hit ratios using D2/test. It is worth recalling that, as discussed in the previous section, a hit in DRCache is when the snippet returned by the cache has $score(S_{d,q},\ q) \geq 1$. On the other hand, if the score is less than one, we say DRCache incurs in a *quality miss*.

The first set of experiments consists in evaluating the hit ratio of DRCache when used in combination of a $\mathsf{RCache}_{\mathsf{DocID}}$. Since $\mathsf{RCache}_{\mathsf{DocID}}$ only stores the document identifiers of each results list, all the queries generate requests to DR, and the size (better to say, the hit ratio) of the FECache does not affect the performance of DRCache. Table 2.3 shows hit ratio results for five different DRCache organizations. $\mathsf{DRCache}_{\mathsf{doc}}$, $\mathsf{DRCache}_{\mathsf{surr}}$ (*doc*, and *surr* in the table), and $\mathsf{DRCache}_{\mathsf{Ssnip}}$. The latter is tested by using three different *maximum* supersnippet sizes (i.e. the *maximum* number of sentences forming the supersnippet), namely 5, 10, and 15 (i.e. ss_5, ss_{10}, and ss_{15}) sentences. The tests regarding $\mathsf{DRCache}_{\mathsf{doc}}$ were done by considering all the documents as having a size equal to the average size of the documents retrieved by Yahoo! for all the queries in dataset D1. The size of the surrogates used for testing $\mathsf{DRCache}_{\mathsf{surr}}$ were instead fixed to be half of the corresponding document [144].

Both $\mathsf{DRCache}_{\mathsf{doc}}$ and $\mathsf{DRCache}_{\mathsf{surr}}$ are managed using a LRU policy.

The first observation is that in all the cases, hit ratios increase linearly when cache size increases. Also in this case, the linear trend in the hit ratio growth is confirmed. Indeed, the growth cannot be indefinite as the maximum hit ratio attainable is bounded from above by the number of distinct snippets (considering their quality, too) that can be generated by SERPs for queries in the stream.

In terms of hit ratio, $\mathsf{DRCache}_{\mathsf{Ssnip}}$ outperforms $\mathsf{DRCache}_{\mathsf{doc}}$, and $\mathsf{DRCache}_{\mathsf{surr}}$ independently of the size of the cache. On the other hand, we can note that the maximal size of supersnippets does not affect sensitively $\mathsf{DRCache}_{\mathsf{Ssnip}}$ hit ratio.

DRCache	Size (in MB)	Hit Ratio DRCache
doc	256M	0.38
	512M	0.41
	1024M	0.44
	2048M	0.49
surr	256M	0.41
	512M	0.453
	1024M	0.49
	2048M	0.53
ss_5	256M	0.42
	512M	0.463
	1024M	0.51
	2048M	0.554
ss_{10}	256M	0.416
	512M	0.462
	1024M	0.51
	2048M	0.55
ss_{15}	256M	0.413
	512M	0.461
	1024M	0.51
	2048M	0.55

Table 2.3: Hit ratios of DRCache (with $\mathsf{DRCache}_{\mathsf{doc}}$, $\mathsf{DRCache}_{\mathsf{surr}}$, and DR-$\mathsf{Cache}_{\mathsf{Ssnip}}$ strategies) and $\mathsf{RCache}_{\mathsf{DocID}}$.

An important observation, is that we have a miss in $\mathsf{DRCache}_{\mathsf{Ssnip}}$ when either the requested document is not present in cache or when the quality score is too small (i.e. less than one). Not necessarily this happens when a “good” snippet cannot be retrieved from $\mathsf{DRCache}_{\mathsf{Ssnip}}$. In real cases, for instance, this might be due to misspelled queries that if automatically corrected before being submitted to the underlying back end, would result in an acceptable score. As an example, consider the query "*gmes*", clearly this query can be easily corrected into "*games*". In fact, the snippet generated for the first result returned by Yahoo! for the query "*gmes*" is:

> " Games.com has a new free online game everyday. Play puzzle games, hidden object games, arcade games and more! Games.com has exclusive poker, casino and card games."

Since the original query term "*gmes*" is not included into the snippet, its quality score is 0, and even if we generate exactly the same snippet as the one generated by Yahoo!, we would count it as a quality miss. To overcome this issue, when we have quality score less than one, we consider the snippet returned by Yahoo! for the same query-document pair. If even the score of the Yahoo! snippet is low as our, a hit could be counted. Table 2.4 shows the hit ratio results for the same tests of the previous table when quality misses are counted as just described. In all the considered cases hit ratio values raise by about $8\% - 9\%$. Furthermore, this can be probably considered as a better estimation of real hit ratios that can be attained by our $\mathsf{DRCache}_{\mathsf{Ssnip}}$ in a real-world system.

Therefore, we can conclude that in the case of a FE running a $\mathsf{RCache}_{\mathsf{DocID}}$, the presence of a DRCache decreases considerably the load on the document server, and that $\mathsf{DRCache}_{\mathsf{Ssnip}}$ is the best cache organization among the ones tested.

DRCache	Size (in MB)	Hit Ratio DRCache
ss_5	256M	0.5
	512M	0.55
	1024M	0.59
	2048M	0.63
ss_{10}	256M	0.497
	512M	0.548
	1024M	0.59
	2048M	0.625
ss_{15}	256M	0.493
	512M	0.546
	1024M	0.59
	2048M	0.62

Table 2.4: Hit ratios of $\mathsf{DRCache}_{\mathsf{Ssnip}}$ ($\mathsf{RCache}_{\mathsf{DocID}}$ case) when quality-misses are counted by comparing quality of snippets with Yahoo! ones.

The next experiment consists in evaluating the effect of a $\mathsf{RCache}_{\mathsf{SERP}}$ as a filter for requests arriving to DRCache. A $\mathsf{RCache}_{\mathsf{SERP}}$ cache, in fact, stores the whole SERP including the generated snippets. In case of a $\mathsf{RCache}_{\mathsf{SERP}}$ cache hit, then, we do not have to request those snippets to the DRCache, thus reducing considerably the load on the DRCache. On the other hand, since frequently requested (i.e. popular) snippets are already built and stored into the $\mathsf{RCache}_{\mathsf{SERP}}$ cache, we expect DRCache's hit ratio to be lower than the one obtained when used in combination with $\mathsf{RCache}_{\mathsf{DocID}}$.

Tables 2.5 and 2.6 show hit ratios for various configurations of DRCache. As in the above experiment, Table 2.6 reports hit ratios when quality-misses are counted by using the comparison between snippets generated from the supersnippets and Yahoo! ones.

As expected, in all the cases hit ratios are lower than in the previous cases. The differences in the hit ratios obtained by $\mathsf{DRCache}_{\mathsf{Ssnip}}$ with respect to $\mathsf{DRCache}_{\mathsf{doc}}$, and $\mathsf{DRCache}_{\mathsf{surr}}$ instead increase remarkably. This prove that our supersnippets are very flexible and can provide effective snippets also for less popular queries. Moreover, if we sum up the hit ratios occurring on both FECache and $\mathsf{DRCache}_{\mathsf{Ssnip}}$, we obtain a impressive *cumulative* hit ratio of about 62%. Note that this is an upper bound to the real cumulative hit ratio. Indeed, FECache stores duplicate snippets (due to possible shared snippets among SERPs), therefore the actual cumulative hit ratio may be slightly lower. We can observe that also in this set of experiments, the maximum number of sentences stored in the supersnippet does not influence heavily the $\mathsf{DRCache}_{\mathsf{Ssnip}}$ hit ratio.

2.10 Summary

We have presented a novel technique for scaling up search engine performance by means of a novel caching strategy specifically designed for document snippets. Design choices of our novel DR cache are motivated by the analysis of a real-world query log that allowed us to better understand the characteristics and the popularity distribution of URLs, documents and snippets returned by a WSE. Our $\mathsf{DRCache}_{\mathsf{Ssnip}}$ stores in its entries the supersnippets that are generated by exploiting the knowledge collected from the query-biased snippets returned in the past by a WSE. $\mathsf{DRCache}_{\mathsf{Ssnip}}$ enables the construction of effective snippets (having an average quality very close to that measured on our ground truth) for already processed query/docID pairs and, more importantly, the "*in-cache*" generation of snippets also for query/docID pairs not previously seen. A deep experimentation was conducted using a large real-world query log by varying the size and the organization of the caches present in an abstract WSE architecture model. The hit ratios measured for $\mathsf{DRCache}_{\mathsf{Ssnip}}$ result to be remarkably higher than those obtainable with other DR cache organizations.

In particular a hit-ratio of 62% was measured using a $\mathsf{DRCache}_{\mathsf{Ssnip}}$ cache of 2,048MB and a docID result cache in th FE. In the experimental evaluation, we considered also the presence of a SERP cache on the FE that filters out most frequent queries, without asking neither the WSE BE nor the DR. Even in this case the hit rates measured for our supersnippet-based cache results to be very high (up to 42%), with a very large difference over the other cache organizations tested.

FECache #Entries (Hit Ratio)	DRCache	Size (in MB)	Hit Ratio DRCache
128K (0.36)	*doc*	256M	0.087
		512M	0.11
		1024M	0.14
		2048M	0.19
	surr	256M	0.11
		512M	0.148
		1024M	0.19
		2048M	0.25
	ss_5	256M	0.15
		512M	0.217
		1024M	0.29
		2048M	0.36
	ss_{10}	256M	0.141
		512M	0.21
		1024M	0.287
		2048M	0.35
	ss_{15}	256M	0.14
		512M	0.207
		1024M	0.287
		2048M	0.35
256K (0.38)	*doc*	256M	0.083
		512M	0.1
		1024M	0.134
		2048M	0.17
	surr	256M	0.104
		512M	0.135
		1024M	0.177
		2048M	0.23
	ss_5	256M	0.139
		512M	0.194
		1024M	0.266
		2048M	0.34
	ss_{10}	256M	0.13
		512M	0.189
		1024M	0.266
		2048M	0.334
	ss_{15}	256M	0.12
		512M	0.18
		1024M	0.266
		2048M	0.33

Table 2.5: Hit ratios of DRCache (with $\text{DRCache}_{\text{doc}}$, $\text{DRCache}_{\text{surr}}$, and $\text{DRCache}_{\text{Ssnip}}$ strategies) and $\text{RCache}_{\text{SERP}}$.

FECache #Entries (Hit Ratio)	DRCache	Size (in MB)	Hit Ratio DRCache
128K (0.36)	ss_5	256M	0.21
		512M	0.29
		1024M	0.36
		2048M	0.42
	ss_{10}	256M	0.2
		512M	0.28
		1024M	0.356
		2048M	0.419
	ss_{15}	256M	0.19
		512M	0.27
		1024M	0.355
		2048M	0.419
256K (0.38)	ss_5	256M	0.19
		512M	0.258
		1024M	0.336
		2048M	0.4
	ss_{10}	256M	0.18
		512M	0.251
		1024M	0.334
		2048M	0.398
	ss_{15}	256M	0.18
		512M	0.249
		1024M	0.332
		2048M	0.395

Table 2.6: Hit ratios of $\mathsf{DRCache}_{\mathsf{Ssnip}}$ ($\mathsf{RCache}_{\mathsf{SERP}}$ case) when quality-misses are counted by comparing quality of snippets with Yahoo! ones.

Chapter 3

Semantic Query Recommendations

3.1 Introduction

The typical interaction of a user with a Web search engine consists in *translating her information need in a textual query made of few terms.* Such queries are usually stored by the search system in its *query log.* Mining query logs to study the users' past interactions is an effective approach for improving the efficacy of a search system, and in particular for producing relevant query suggestions. This is based on the assumption that successful search activities by previous users can be of interest for others.

In this Chapter we focus on how to improve query recommendation through entities. We believe that the "*Web of Data*" can be profitably exploited in the query log mining process to alleviate possible vocabulary mismatch problems, and that it can be leveraged to provide more user-friendly query suggestions. Search query recommendation techniques [35, 29, 13] are commonly used in web search engines to help users refine their queries. These technologies analyze the user behavior by mining the system logs in order to find the correlation between the user's information need (i.e. what? - visited pages), what the user is searching for (i.e. how? - query terms) and the content and structure of the information pool (i.e. search index). Even so, the usefulness of the query recommendations is limited by their inherent weaknesses, such as:

Data Skewness usually query popularity presents a power-law-like curve distribution. Due to that, query recommender systems whose models rely on the previous queries are really good in producing recommendations for head queries, poor in producing recommendations for queries in the long tail, and often they are not able to produce suggestions for queries that do not appear in the previous history [32, 139].

Misspelling and Multilinguality users often spell wrongly the names of the searched items, or perform queries in different languages. For example, one could simply check the various ways of addressing in DBPedia the information about Da Vinci's famous painting Mona Lisa[1] (see the label and the wikiPageRedirects sections);

Incompleteness and Ambiguity when searching for information about the well known painter, most of the users are simply using the terms `da vinci`, which could mean several different things[2].

In this Chapter we present *Semantic Search Shortcuts*(S^3) a new approach for producing enriched query recommendations. In Section 3.2 we introduce Europeana, one of the largest digital libraries in the world, collecting more than 20 million records from European museums, libraries, archives and multi-media collections. We collaborated with the Europeana Fondation, which shared with us their users' historical data. In Section 3.3 we perform a detailed analysis of these logs, both from a *query* and *session* perspective. In Section 3.4 we resume state of the art in query recommendation and in Section 3.5 we introduce *Search Shortcuts* [35] (SS), a query recommender system able to provide relevant suggestions for rare or never seen queries. In Section 3.6 we describe our semantic version S^3; it is worth observing that our extension does not only suggestion raw text queries but *entities* (represented by URIs). This involves several benefits: e.g., being able to personalise the suggestions with the user's language, improve diversification (since there are not suggestions about the same entity e.g., `mona lisa` and `gioconda`). Moreover, in Section 3.7, we propose a new approach for reducing human effort in evaluating the quality of the suggestions, based on some measures for evaluating relevance and diversity on a list of suggestions. These measures exploit the *entity relatedness* concept, a graph based measure that estimates the semantic distance between two entities (which we investigate in deep in Chapter 4). We show that our proposed recommender overperforms SS both for quality and diversification of the suggestions, also if we consider rare queries. Finally in Section 3.8 we draw our conclusions.

3.2 Europeana

The strong inclination for culture and beauty in Europe created invaluable artifacts starting from antiquity up to nowadays. That cultural strength is

[1]`http://dbpedia.org/page/Mona_Lisa`

[2]see for example `http://dbpedia.org/page/Da_Vinci_(disambiguation)`

recognized by all people in the world and makes Europe the destination for a half of the international tourists [3]. More than 220 million people visit the European countries yearly for spending their holidays.

The European Commission is aware about the value of this cultural heritage and decided to make it more accessible to the public by supporting digitization of the cultural heritage and by financing the Europeana group projects. The first prototype of the Europeana Portal[4] was launched in autumn 2008 and contains more than 20 million items.

Due to increasing amount of information published within the portal, the access to the description of a specific masterpiece becomes each day a more time consuming task, when the user is not able to create a very restrictive query. For example, searching for general terms like `renaissance` or `art nouveau` produces more than $10,000$ results. If a user searches for the term `Gioconda` the system retrieves a couple of hundred documents, while searching for *Mona Lisa, Da Vinci* only returns twenty images of the well known painting. These examples show how important is to formulate good queries in order to satisfy an information need. This is a challenging task, given the fact that the document base is cross-domain, multi-lingual and multi-cultural.

3.3 The Europeana Query Log

A query log keeps track of historical information regarding past interactions between users and the retrieval system. It usually contains tuples $\langle q_i, u_i, t_i, V_i, C_i \rangle$ where for each submitted query q_i the following information is available: i) the anonymized identifier of the user u_i, ii) the submission timestamp t_i, iii) the set V_i of documents returned by the search engine, and iv) the set C_i of documents clicked by u_i. Therefore, a query log records both the activities conducted by users, e.g. the submitted queries, and an implicit feedback on the quality of the retrieval system, e.g. the clicks.

In this work, we consider a query log coming from Europeana portal[5], relative to the time interval ranging from August 27, 2010 to February, 24, 2011. This is a six months worth of users' interactions, resulting in $1,382,069$ distinct queries issued by users from 180 countries (3,024,162 is the total number of queries). We preprocessed the entire query log in order to remove noise (e.g., stream of queries submitted by software robots instead of humans).

It is worth noticing that $1,059,470$ queries (i.e., 35% out of the total) also contain a *filter* (e.g., YEAR:1840). Users can filter results by *type*, *year* or

[3] http://www.unwto.org/facts/eng/pdf/highlights/UNWTO_Highlights10_en_HR.pdf

[4] http://www.europeana.eu/portal

[5] http://www.europeana.com/portal/

provider simply by clicking on a button, so it is reasonable that they try to refine retrieved results by applying a filter, whenever they are not satisfied. Furthermore, we find that users prefer filtering results by type, i.e., images, texts, videos or sounds. Indeed, we measure that 20% of the submitted queries contains a filter by type. This is an additional proof of the skillfulness of Europeana users and their willingness to exploit non trivial search tools to find their desired contents. This also means that advanced search aids, such as query recommendation, would be surely exploited.

Similarly to Web query log analysis [128], we discuss two aspects of the analysis task: i) an analysis on the *query set* (e.g., average query length, query distribution, etc.) and ii) a higher level analysis of *search sessions*, i.e., sequences of queries issued by users for satisfying specific information needs.

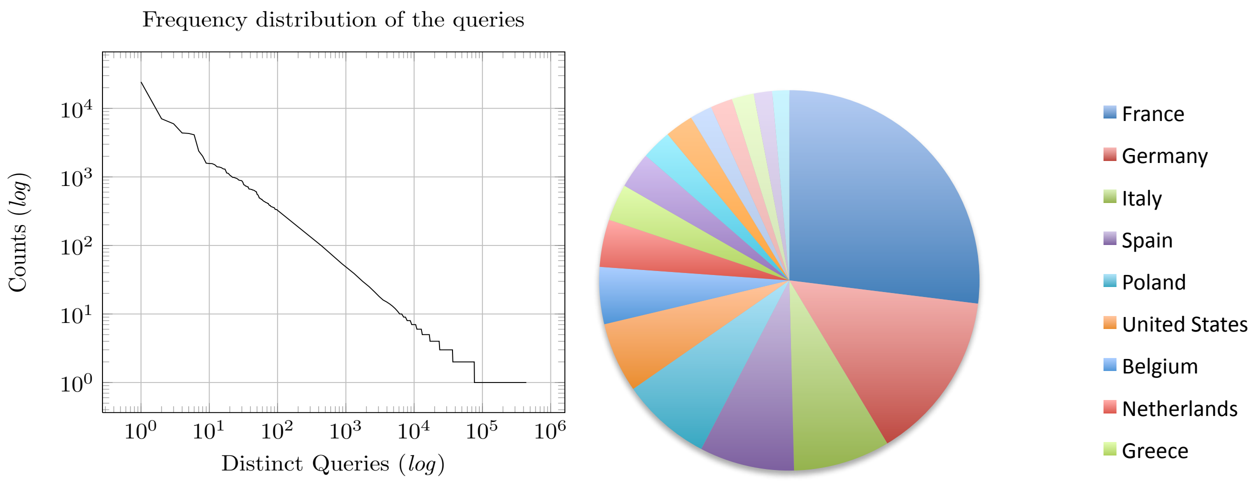
3.3.1 Query Analysis

First we analyze the load distribution on the Europeana portal. An interesting analysis can be done on the queries themselves. Figure 3.1(a) shows the frequency distribution of queries. As expected, the popularity of the queries follows a power-law distribution ($p(x) \propto kx^{-\alpha}$), where x is the popularity rank. The best fitting α parameter is $\alpha = 0.86$, which gives a hint about the skewness of the frequency distribution. The larger α the larger is the portion of the log covered by the top frequent queries. Both [99] and [11] report a much larger α value of 2.4 and 1.84 respectively from a Excite and a Yahoo! query log. Such small value of α means that the most popular queries submitted to Europeana do not account for a significantly large portion of the query log. The might be explained by looking at and comparing the main characteristics both of Europeana and Web search engines users. Indeed, since Europeana is strongly focussed on the specific context of cultural heritage, its users are likely to be more skilled and therefore they tend to use a more diverse vocabulary.

In addition, we found that the average length of queries is 1.86 terms, which is again a smaller value than the typical value observed in Web search engine logs. We can argue that the Europeana user has a more rich vocabulary, with discriminative queries made of specific terms.

Figure 3.1(b) shows the distribution of the queries grouped by country. France, Germany, and Italy are the three major countries accounting for about the 50% of the total traffic of queries submitted to the Europeana portal.

Figure 3.2(a) reports the number of queries submitted per day. We observe a periodic behavior over a week basis, with a number of peaks probably related to some Europeana dissemination or advertisement activities. For example, we observe several peaks between the 18th and the 22th November, probably due to the fact that, in those days, Europeana announced to have reached a



(a) (b)

Figure 3.1: Frequency distribution of queries (a) and distribution of the queries over the countries (b).

threshold of 14 million of indexed documents[6].

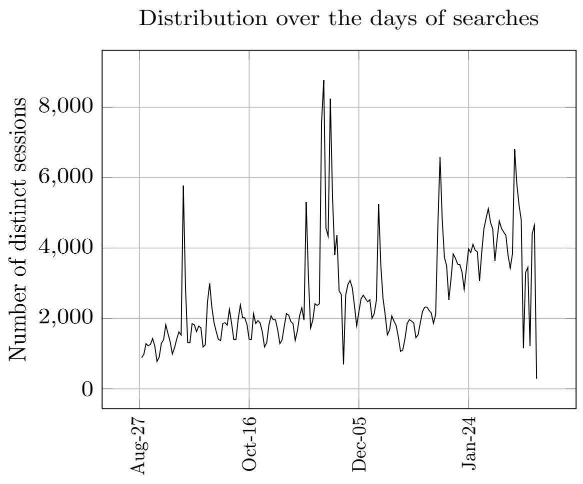
Figure 3.2(b) shows the load on the Europeana portal on a per hour basis. We observe a particular trend. The peak of load on the Europeana portal is in the afternoon, between 15 and 17. It is different from commercial Web search engines where the peak is reached in the evening, between the 19 and the 23 [19]. A possible explanation of this phenomenon could be that the Europeana portal is mainly used by people working in the field and thus, mainly accessed during working hours. From the other side, a commercial Web search engine is used by a wider range of users looking for the most disparate information needs and using it through all the day.

3.3.2 Session Analysis

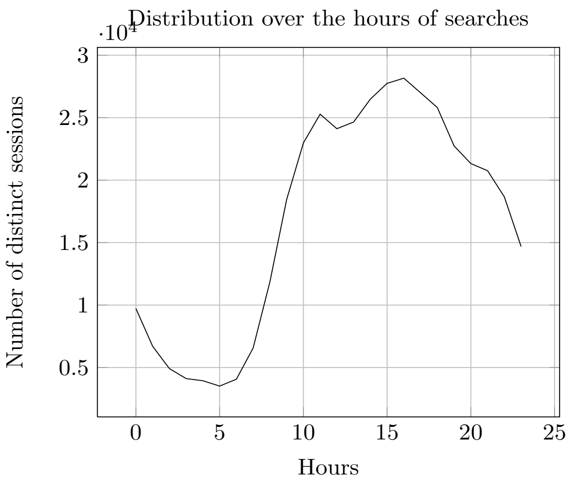
To fully understand user behavior, it is important to analyze also the sequence of queries she submits. Indeed, every query can be considered as an improvement of the previous done by the user to better specify her information need.

Several techniques have been developed to split the queries submitted by a single user into a set of sessions [29, 82, 95]. We adopted a very simple approach which has proved to be fairly effective [128]. We exploit a 5 minutes inactivity time threshold in order to split the stream of queries coming from each user. We assume that if two consecutive queries coming from the same

[6]`http://www.sofiaecho.com/2010/11/18/995971_europes-cultural-heritage-online`



(a)



(b)

Figure 3.2: Distribution of the searches over the days (a) and over the hours (b).

user are submitted within five minutes they belong to the same logical session, whereas if the time distance between the queries is greater, the two queries belong to two different interactions with the retrieval system.

By exploiting the above time threshold, we are able to devise 404,237 sessions in the Europeana query log. On average a session lasts about 276 sec, i.e., less than 5 minutes, meaning that, under our assumption, Europeana's users complete a search activity for satisfying an information need within 5 minutes. The average session length, i.e., the average number of queries within a session, is 7.48 queries. This number of queries is an interesting evidence that the user is engaged by the Europeana portal, and she is willing to submit many queries to find the desired result.

Moreover, we distinguish between *successful* and *unsuccessful* sessions. According to [35], a session is supposed to be successful if its *last* query has got a click associated. To this end, we find 182,280 occurrences of successful sessions in the Europeana query log, that is about 45% of the total. We notice that in [35] it was observed a much larger fraction of successful sessions, about 65%.

Figure 3.3 shows the distributions of session lengths, both for successful and unsuccessful sessions. On the x-axis the number of queries within a session is plotted, while on the y-axis the frequencies, i.e., how many sessions to contain a specific number of queries are reported. We expect successful sessions contain on average less queries than unsuccessful ones, due to the ability of the retrieval system to return early high quality results in successful session. The fact that the session length distributions are very similar, suggests that

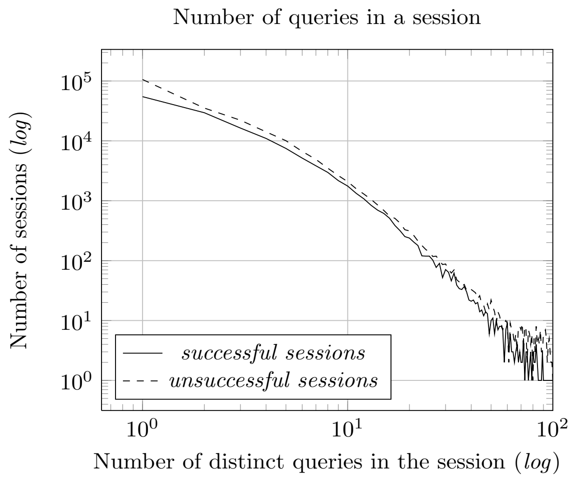


Figure 3.3: Distribution of successful and unsuccessful sessions lengths (in queries).

	Europeana	Web Search Engines
avg. query terms	1.86	2.35 [99], 2.55 [128]
query distribution (i.e., power-law's α)	0.86	2.40 [99], 1.84 [11]
avg. queries per session	7.48	2.02 [128]
% of *successful* sessions	45	65 [35]

Table 3.1: Europeana vs. Web Search Engines: a comparison on query log statistics.

high quality results are not in the top pages, and that the Europeana ranking can be improved in order to present interesting results to the user earlier, thus reducing the successful session length with a general improvement of the user experience.

Table 3.3.2 shows some statistics extracted both from the analysis of the Europeana query log as well as from general purpose Web Search Engines historical search data.

3.4 Query Recommendations

The problem of query suggestion is related to two related research fields that have been traditionally addressed from different points of view: query suggestion algorithms and recommender systems. Recommender systems are used in several domains, being especially successful in electronic commerce.

They can be divided in two broad classes: those based on content filtering, and those on collaborative filtering. As the name suggests, content filtering approaches base their recommendations on the content of the items to be suggested. On the other side, collaborative filtering solutions are based on the preferences expressed by the users.

Due to their characteristic features, query suggestion calls for specifically tailored algorithm being able to exploit all the additional information available in this scenario, such users session, click, query results, etc. Techniques proposed during last years are very different, yet they have in common the exploitation of usage information recorded in query logs. Many approaches extract the information used from the plain set of queries recorded in the log, although there are several works that take into account the chains of subsequent queries that belong to the same search session.

The authors of [12] exploit click-through data as a way to provide recommendations. The method is based on the concept of Cover Graph. A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs. The sharing of clicked URLs results to be very effective for devising related queries.

Boldi *et al.*[29] introduce the concept of Query Flow Graph (QFG), an aggregated representation of the information contained in a query log. A QFG is a directed graph in which nodes are queries, and the edge connecting node q_1 to q_2 is weighted by the probability that users issue query q_2 after issuing q_1. Authors highlight the utility of the model in two concrete applications, namely, devising logical sessions and generating query recommendation. The authors refine the previous studies in [30, 31] where a query suggestion scheme based on a random walk with restart model on the QFG is proposed.

Baraglia *et al.* [15] propose a new model for query recommendation, the Search Shortcut (SS), which we describe in detail in the next section.

Query suggestion has been an effective approach to help users narrow down to the information they need. However, most of the existing studies focus only on popular queries. Since rare queries possess much less information (e.g., clicks) than popular queries in the query logs, it is much more difficult to efficiently suggest relevant queries to a rare query.

Yang *et al.* [131] propose an optimal rare query suggestion framework by leveraging implicit feedbacks from users in the query logs.

Broder *et al.* leverage the results from search engines as an external knowledge base for building the word features for rare queries [38]. The authors train a classifier on a commercial taxonomy consisting of 6,000 nodes for categorization. Results show a significant boost in term of precision with respect to the baseline query expansion methods. Lately, Broder *et al.* propose an efficient and effective approach for matching ads against rare queries [37]. The approach builds an expanded query representation by leveraging offline processing done for related popular queries. Experimental results show that the proposed technique significantly improves the effectiveness of advertising on rare queries with only a negligible increase in computational cost.

Bonchi *et al.* [32] define the Term-Query graph, composed by QFG nodes and additional nodes modeling terms within queries, and edges connecting any term to the queries that contain it. They produce the suggestions computing the center-piece subgraph induced by terms in the query submitted by the user. Authors show that their method produce high quality recommendations and works for long-tail queries, where other methods fail even to produce any suggestion.

Several works exploiting the concept of entity were proposed. Meij *et al.* [102] propose to return a list entities: they match the query and all its possible n-grams against the entity labels in DBpedia and then experiment several machine learning approaches to select the entities to suggest.

Szpektor *et al.* [139] exploit the concept of *template*, in order to learn rules such as `<CITY> flight` $\rightarrow$ `<CITY> hotel`. Given these rules, if a user search `honolulu flight`, the system is able to suggest the query `honolulu hotels` also if the query never appeared in the logs. Their model is an extension of the QFG with additional template nodes. The graph has edges that connect queries in the QFG to templates and edges that connect templates if they cooccur in the same session. A similar idea is proposed by Bordino *et al.* [33] which propose the the EQGraph (Entity-Query Graph). The EQGraph contains entities instead of templates, and there are edges connecting query with entities, and entities with the other entities. The input of their recommender is not a query but the current document visited by the user: they perform Personalized PageRank computation starting from the entities contained in the visited page, and recommend relevant query suggestions.

3.5 Search Shortcuts

The analysis conducted in Section 3.3 shows that the search experience of the user interacting with Europeana could be improved. To this extent, we now introduce an application exploiting the knowledge extracted from the Europeana query log aiming at enhancing the interaction of users by suggesting a list of possible interesting queries.

A search session is an interactive process where users continuously refines their search query in order to better specify their information need. Sometimes, the successful query is not known in advance, but users might adopt concepts and terminologies also on the basis of the results pages visited. Query recommendation is a very popular technique aiming at proposing successful queries as early as possible. The approach described below, exploits successful queries from successful session to *recommend queries that allowed "similar" users, i.e., users which in the past followed a similar search process, to successfully find the information they were looking for*, and it is able to catch non trivial semantic relationships among queries.

We adopt the *Search Shortcuts* (SS) model proposed in [15] and its terminology. The SS has a clear and sound formulation as the problem of recommending queries that can reduce the search session length, i.e., leading users to relevant results as early as possible.

Let $\mathcal{U}$ be the set of users of a WSE whose activities are recorded in a query log QL, and $\mathcal{Q}$ be the set of queries in QL. We suppose QL is preprocessed by using some session splitting method (e.g. [82, 95]) in order to extract query *sessions*, i.e., sequences of queries which are related to the same user search task. Formally, we denote by $\mathcal{S}$ the set of all sessions in QL, and σ^u a session issued by user u. Moreover, let us denote with σ_i^u the i-th query of σ^u. For a session σ^u of length n its *final query* is the query σ_n^u, i.e. the last query issued by u in the session. To simplify the notation, in the following we will drop the superscript u whenever the user u is clear from the context.

As previously introduced, we say that a session σ is *successful* if and only if the user has clicked on at least one link shown in the result page returned by the WSE for the final query σ_n, *unsuccessful* otherwise.

We define a novel algorithm that aims to generate suggestions containing only those queries appearing as final in successful sessions. The goal is to suggest queries having a high potentiality of being useful for people to reach their initial goal. In our view, suggesting queries appearing as final in successful sessions is a good strategy to accomplish this task.

The SS algorithm works by efficiently computing similarities between partial user sessions (the one currently performed) and historical successful sessions recorded in a query log. Final queries of most similar successful sessions are suggested to users as search shortcuts.

Let σ' be the current session performed by the user, and let us consider the sequence τ of the concatenation of all terms with possible repetitions appearing in $\sigma'_{t|}$, i.e. the head of length t of session σ'. Then, we compute the value of a scoring function $\delta\left(\tau, \sigma^s\right)$, which for each successful session measures the similarity between its queries and the set of terms τ. Intuitively, this similarity measures how much a previously seen session overlaps with the user need expressed so far (the concatenation of terms τ serves as a bag-of-words model of user need). Sessions are ranked according to δ scores and from the subset of the top ranked sessions we suggest their final queries. It is obvious that depending on how the function δ is chosen we may have different recommendation methods. In our particular case, we opt for δ to be the similarity computed as in the BM25 metrics [123]. The choice of an IR-like metric allows us to take much care of words that are discriminant in the context of the session to which we are comparing. BM25, and other IR-related metrics, have been designed specifically to account for that property in the context of query/documents similarity. We borrow from BM25 the same attitude to adapt to this condition. The shortcuts generation problem has been, thus, reduced to the information retrieval task of finding highly similar sessions in response to a given sequence of queries. In most cases, it is enough to use only the last submitted query to propose optimal recommendations.

The idea described above is thus translated into the following process. For each unique *final query* q_f contained in successful sessions we define what we have called a *virtual document* identified by its *title* and its *content*. The title, i.e., the identifier of the document, is exactly query string q_f. The content of the virtual document is instead composed of all the terms that have appeared in queries of all the successful sessions ending with q_f. At the end of this procedure we have a set of virtual documents, one for each distinct final query occurring in some successful sessions. Just to make things more clear, let us consider a toy example. Consider the two following successful sessions: (*dante alighieri* $\rightarrow$ *divina commedia* $\rightarrow$ *paolo e francesca*), and (*divina commedia* $\rightarrow$ *inferno canto V* $\rightarrow$ *paolo e francesca*). We create the virtual document identified by title *paolo e francesca* and whose content is the text (*dante alighieri divina commedia divina commedia inferno canto V*). As you can see the virtual document actually contains also repetitions of the same terms that are considered in the context of the BM25 metrics. All virtual documents are indexed with the preferred Information Retrieval system, and generating shortcuts for a given user session σ' is simply a matter of processing the query $\sigma'_{t|}$ over the inverted file indexing such virtual documents. We know that processing queries over inverted indexes is very fast and scalable, and these important characteristics are inherited by our query suggestion technique as well.

The other important feature of our query suggestion technique is its robustness with respect to rare and singleton queries. Singleton queries account for almost 50% of the submitted queries [129], and their presence causes the issue of the sparsity of models [3]. Since we match τ with the text obtained by concatenating all the queries in each session, we are not bound to look for previously submitted queries as in the case of other suggestion algorithms. Therefore, we can generate suggestions for rare queries of the query distribution whose terms have some context in the query log used to build the model.

3.6 Semantic Query Recommendation

3.6.1 Hidden Knowledge in Search Sessions

A search session is an interactive process where users continuously refine their queries in order to better specify their information need. Sometimes, the successful query is not known in advance, but users might adopt concepts and terminologies also on the basis of the results pages visited. The approach described here, exploits successful queries from successful sessions to recommend queries that allowed "*similar*" users, i.e., users which in the past followed a similar search process, to successfully find the information they were looking for, and it is able to catch non trivial semantic relationships among queries.

Even if query logs hide precious knowledge that can be profitably exploited for many applications, they contain a lot of noise. In the Europeana query log, several kinds of noise could be identified: i) multiple representation of the same entity, due to different languages (e.g., en/Divine Comedy vs. it/Divina Comedia), or different way to denote the same entity (e.g., *leonardo*, or *da vinci*, are both referring to the painter *Leonardo Da Vinci*), ii) homonyms (e.g., the term *war* refers to several different historical events), iii) multi-goal search sessions, when users search for multiple loosely connected information pieces in the same search session (e.g. search for *Inferno* and for *Gioconda*). In order to be able to provide good search terms recommendations it is mandatory to remove most of the noise from the recommendation index.

These problems are well known within *annotation systems* literature [57, 65, 89, 110] which we described in Section 1.3.4.

3.6.2 Semantic Search Shortcuts

In order to perform the mapping between a query and its associated entities, we preliminary annotate query sessions and virtual documents. Most query annotation approaches consider single queries and try to map them to an entity. If a query is ambiguous, the risk is to always map it to the most popular associated entity. On the other hand, by performing the mapping by using all the queries in a user session, we can exploit other queries as a source of contextual information and improve precision remarkably.

The Collective Entity Linking approach proposed by Han *et al.* [65] was implemented by using a recent dump of English Wikipedia, we adopted this method because was easy to implement and authors reported it is more effective than other available approaches. We direct interested readers to the original paper for a detailed explanation of the annotation method. The dump contains 4,677,051 entities, denoted by 11,318,080 distinct spots, and interconnected by

104,220,848 semantic relations. It is worth to observe that we used only the English language part of Wikipedia. Even though queries in the query log are written in different languages we decided to use the English version of Wikipedia as it contains a large number of spots in other languages, in form of anchors or redirects (e.g., Italian *Divina Commedia* is redirected to *Divine Comedy*). We leave the investigation regarding the use of different languages as future work.

By means of the Collective Entity Linking annotator we tagged the text of each virtual document $\mathcal{VD}$. More precisely, we keep track of each session belonging to the content of the virtual documents, and we used the annotator to identify mappings between queries and entities. Each identified entity is then added to a new field *entities* of the virtual document. The annotator provides a confidence value of every annotation proposed. We store also such confidence value in the index. At the end of the process, our inverted index contains a set of virtual documents, each one consisting of three fields: *title*, *content*, and *entities*. We observe that by following this approach, if the title of a given virtual document is affected by homonymy, the entities field is likely to contain disambiguations representing some possible meanings. We define *Semantic Search Shortcuts* (S^3) the query recommender system exploiting this new added knowledge that allows us to suggest queries by exploiting the semantic relations among the current query and those submitted in the past. Note that, differently from traditional recommenders that generate for each query a flat list of recommendations, S^3 provides a list of related entities. Suggesting entities instead of textual queries can be convenient for a number of reasons. First entities do not suffer from language issues. Moreover, the enriched semantic of S^3 recommendations enables a more user friendly representation within the portal. For example, some kind of suggested entities (e.g., paintings, people) can be more clearly represented by images.

Given an input query q, in order to compute the entities to be suggested we first retrieve the top-k most relevant virtual documents by processing the query over the SS inverted index built as described in Section 3.5. The result set R_q contains the top-k relevant virtual documents along with its associated entities. Given an entity e in the result set, we define:

$$score(e, \mathcal{VD}) = \begin{cases} conf(e) \times score(\mathcal{VD}), & \text{if } e \in \mathcal{VD}.entities \\ 0 & \text{otherwise} \end{cases}$$

where $conf(e)$ is the confidence of the annotator in mapping the entity e in the virtual document $\mathcal{VD}$, while $score(\mathcal{VD})$ represents the similarity score returned by the information retrieval system. We then rank the entities appearing in R_q using their score w.r.t. the query, as below:

$$score(e,q) = \sum_{\mathcal{VD} \in R_q} score(e, \mathcal{VD})$$

3.7 Experimental Evaluation

We use a large query log coming from the Europeana portal, containing a sample of users' interactions covering two years (from August 27, 2010 to January, 17, 2012). We preprocess the entire query log to remove noise (e.g., queries submitted by software robots, mispells, different encodings, etc). We then apply a time-based session splitting technique. We use a threshold of 30 seconds between each consecutive query pair and we filter out sessions composed by only one query. By doing so, we obtain $139,562$ *successful* sessions. Finally, we use these successful session to build the inverted index as described in Section 3.6.2.

3.7.1 Relatedness and Diversity

We are interested in evaluating two aspects of the set of suggestions provided. These are our main research questions:

Relatedness : How much information related to the original query a set of suggestions is able to to provide?

Diversity : How many different aspects of the original query a set of suggestions is able to cover?

In order to evaluate these aspects we borrow from the annotators the concept of *semantic relatedness* between two entities proposed by Milne and Witten [110]:

$$rel(a,b) = \rho^{\mathsf{MW}}(a,b) = 1 - \frac{\log(\max(|in(a)|,|in(b)|)) - \log(|in(a) \cap in(b)|)}{\log(|W|) - \log(\min(|in(a)|,|in(b)|))} \quad (3.1)$$

where W is the set of all Wikipedia entities, while $in(a)$ and $in(b)$ are the sets of Wikipedia articles linking to a and b, respectively. When $|in(a) \cap in(b)| = 0$, we have $\rho^{\mathsf{MW}}(a,b) = 0$. In addition, ρ^{MW} is maximum (equal to 1) when $in(a) \cap in(b) = in(a) = in(b)$, and thus all the articles that cite a also cite b, and vice versa.

We extend this measure to compute the similarity between two set of entities (the function $in()$ gets a set of entities and returns all the entities that link *at least* on entity in the given set). At the same time, given two sets of entities A, B, we define the diversity as $div(A, B) = 1 - rel(A, B)$. Given a query q, let E_q be the set of entities that have been manually associated with the query. We define the relatedness and the diversity of a list of suggestions S_q as:

Definition 1. *The average relatedness of a list of suggestions is computed as:*

$$rel(S_q) = \frac{\sum_{s \in S_q} rel(E_s \setminus E_q, E_q)}{|S_q|}$$

where E_s represents the set of entities mapped to a suggestion s (could contain more than one entity in the manual annotated dataset). Please note that we remove the entities of the original query from each set of suggestions as we are not interested in suggesting something that do not add useful content w.r.t. the starting query ($E_s \setminus E_q$).

Definition 2. *The average diversity of a list of suggestions is defined as:*

$$div(S_q) = \frac{\sum_{s \in S_q} div(E_s, E_{S_q \setminus s})}{|S_q|}$$

For each suggestion, we want to evaluate how much information it adds w.r.t. the other suggestions. $E_{S_q \setminus s}$ denotes the union of the entities belonging to all the suggestions except the current suggestion s.

Note that diversity and relatedness are very often two contrasting objectives. In order to maximize relatedness, and algorithm should provide the results being most related with the user query. On the other hand, such results are likely to be very similar to each other. The correct trade-off between relatedness and diversity would be able to provide high quality results covering multiple topics being relevant for the user query.

3.7.2 Evaluation Methodology

The evaluation of query recommendation techniques is an open issue. In most cases, it is addressed by performing ad-hoc user studies. User studies are time-consuming activities, very difficult to be repeated for comparative evaluations of different methods. In this work, we intend to overcome the problem by proposing a new way to evaluate query recommendation. In particular, we want to assess how query recommender systems performs in terms of entities suggested thus enabling us to use the metrics (i.e., relatedness and diversity) defined above. We built a dataset consisting in 130 queries split in three disjoint sets. The 50 queries in the first set are short (composed by only one term). The second set contains 50 queries having an average length of 4.2 terms, while the 30 queries in the third set have an average length of 9.93 terms. For each query in the three sets, we compute the top-10 recommendations produced by the SS query recommender system and we map them to entities by using a simple interface providing an user-friendly way to associate entities to queries. Three annotators participated to the manual annotation. They manually annotated queries and their related recommendations with one or more Wikipedia entities. An entity is represented by its *title* and its numerical id, available in a recent English Wikipedia dump. The whole set of queries and recommendations has been randomly divided in three distinct sets. Each of those sets has been annotated by one annotator and cross-checked by the other two.

The rationale of using different sets of queries varying for their length is that we want to assess how our technique performs (in terms of entities suggested) w.r.t. the length of the query. Short queries are, in fact, easier to manage as they usually represent a single entity. Longer queries are more difficult as they may be associated with more than one entity thus complicating the scenario. The dataset has been made available for download[7]. Please note that each set of queries also corresponds to a different class of popularity in the query log. In particular, the first set is made up of queries that are typically in the head of the power-law, while the second and the third set are made up of queries in the torso/tail of the power-law. Furthermore, some of the queries in the third set (containing the longest queries) are singleton queries (i.e., queries that occur only once in the query log).

3.7.3 Experimental Results

We evaluate S^3 by comparing its performance with those obtained with the original version of the SS algorithm. In particular, for each set of queries in the dataset described above (*short*, *medium* and *long*), we computed average relatedness and average diversity.

Figure 3.4 shows the average relatedness computed for each query q belonging to a particular set of queries. Results confirm the validity of our intuition as, for all the three sets, the results obtained by S^3 are always greater than the results obtained by considering the SS suggestions. It is worth to observe that the entities suggested by S^3 are potentially completely different by the entities annotated in the suggestions of SS. In fact, while in SS we are exploiting only the entities in the titles, in S^3 we are leveraging all the entities in the *whole virtual document*, using the virtual document relevance to boost

[7]Interested readers can download it from: `http://www.di.unipi.it/~ceccarel/sac13`

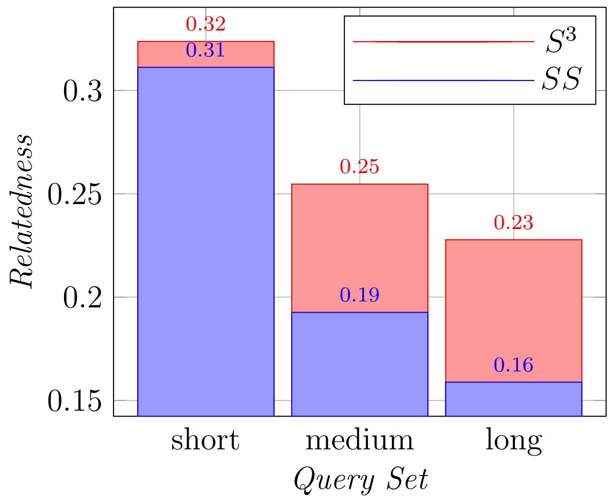


Figure 3.4: Per-set average relatedness computed between the list of suggestions and the given query.

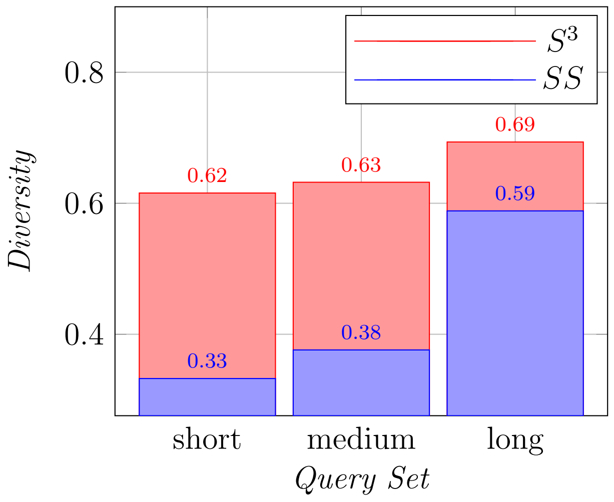


Figure 3.5: Per-set average diversity computed between the list of suggestions and the given query.

the most important entities. Furthermore, the longer the queries the more difficult the suggestion of related queries. This happens because long queries occur less frequently in the log and then we have less information to generate the suggestions. If we consider single sets, the highest gain of S^3 in terms of average relatedness is obtained for medium and long queries: this means that relying on entities allows to mitigate the sparsity of user data.

Figure 3.5 reports the average diversity of the suggestions over the queries of each set. Here, we observe an opposite trend, due to the fact that the longer the queries, the more terms/entities they contain, and the more different the suggestions are. Furthermore, we observe that, for the most frequent queries, SS has a very low performance *w.r.t.* S^3. This happens because in the case of frequent queries SS tends to retrieve popular reformulations of the original query, thus not diversifying the returned suggestions. S^3 does not suffer for this problem since it works with entities thus diversifying naturally the list of suggestions. We leave as future work the study of a strategy for suggesting entities aiming at maximizing the diversity on a list of suggestions.

Let us clarify with the example in Table 3.2 how the two techniques behave differently. Given the query "*dante*", SS returns the following suggestions: *dante banquet*, *dante boska komedia*, *dante paradiso*, *dante kupferstich*, *dante's divina*, *dante divine comedy*, *dante ali*, *dante alle*. Please note the previously highlighted behavior of SS. The suggestions it produces are often reformulations of the same query, while S^3 is able to expand the set of suggestions to the entities: *Divine_Comedy*, *Dante_Falconeri*, *Italian_battleship_Dante_Alighieri*, *Inferno_(Dante)*, *Ludovico_Ariosto*, *Sándor_Petõfi*, *Petrarch*, *Convivio* with an average relatedness of 0.48 (SS, 0.43) and a diversity of 0.40 (SS, 0.10).

SS Suggestions	S^3 Suggestions
dante banquet	*Divine Comedy*
dante boska komedia	*Dante Falconeri*
dante paradiso	*Italian battleship Dante Alighieri*
dante kupferstich	*Inferno (Dante)*
dante's divina	*Ludovico Ariosto*
dante divine comedy	*Sándor Petõfi*
dante ali	*Petrarch*
dante alle	*Convivio*

Table 3.2: Suggestions provided for the query "*dante*"

3.8 Summary

In this Chapter we propose an analysis of a large query log coming from a digital library. We reused the concepts of session identification, time series analysis, query chains and task based search when analyzing the Europeana logs. To the best of our knowledge, this is first analysis of the user interaction with a cultural heritage retrieval system.

Our analysis highlights some significative differences between the Europeana query log and the historical data collected by general purpose Web Search Engine logs. In particular, we find out that both query and search session distributions show different behaviors. Such phenomenon could be explained by looking at the characteristics of Europeana users, which are typically more skilled than generic Web users and, thus, they are capable of taking advantage of the Europeana portal features to conduct more complex search sessions.

For this reason, we believe that interesting knowledge can be extracted from Europeana query log in order to build advanced assistance functionalities, such as query recommendation. In fact, we investigated the integration of a state-of-the-art algorithm into the Europeana portal.

We then explored the use of entities extracted from a query log to enhance query recommendations. We presented our technique and we assessed its performance by using a manually annotated dataset that has been made available for download to favor the repeatability of experiments. The quality of suggestions generated has been measured by means of two novel evaluation metrics that measure semantic relatedness and diversity.

Chapter 4

Learning Relatedness Measures for Entity Linking

4.1 Introduction

Entity Linking is the task of detecting, in text documents, relevant mentions to entities of a given knowledge base. To this end, entity-linking algorithms use several signals and features extracted from the input text or from the knowledge base. The most important of such features is *entity relatedness*. Indeed, we argue that these algorithms benefit from maximizing the relatedness among the relevant entities selected for annotation, since this minimizes errors in disambiguating entity-linking.

The definition of an effective relatedness function is thus a crucial point in any entity-linking algorithm. In this Chapter we address the problem of learning high-quality entity relatedness functions. First, we formalize the problem of learning entity relatedness as a learning-to-rank problem. We propose a methodology to create reference datasets on the basis of manually annotated data. Finally, we show that our machine-learned entity relatedness function performs better than other relatedness functions previously proposed, and, more importantly, improves the overall performance of different state-of-the-art entity-linking algorithms.

A typical entity linking system performs this task in two steps: *spotting* and *disambiguation*. The *spotting* process identifies a set of candidate spots in the input document, and produces a list of candidate entities for each spot. Then, the *disambiguation* process selects the most relevant spots and the most likely entities among the candidates. The *spotting* step exploits a given catalog of named entities, or some knowledge base, to devise the possible mentions of entities occurring in the input.

Let us introduce a simple example to describe how the entity linking process works:

> *On July 20, 1969, the Apollo 11 astronauts - Neil Armstrong, Michael Collins, and Edwin "Buzz" Aldrin Jr. - realized President Kennedy's dream.*

The text "President Kennedy" can be easily spotted and linked to *John F. Kennedy*, since in Wikipedia there are 98 anchors exactly matching such fragment of text and linking to the U.S. president page. In addition, the text "Apollo 11" may refer to two distinct candidates: the famous spaceflight mission, or a 1996 film directed by Norberto Barba. Similarly, the text "Michael Collins" may refer to either the well known astronaut, or to the Irish leader and president of the Irish provisional government in 1922. Indeed, mentions to the latter (408) are much more frequent than those to the former (141)[1].

The above spots and the relative candidate entities are further processed during the *disambiguation* step. The goal of disambiguation is twofold. First, only relevant spots have to be filtered. For instance, the word "the" may refer to the entity associated with the definite article, but this linking might be relevant only for documents discussing the English grammar. Second, the best candidate entity for each spot has to be selected. This is usually done by considering the context of close mentions and by maximizing some measure of *relatedness* among the linked entities [57, 50, 69, 110, 127]. In our example, the astronaut "Michael Collins" and the "Apollo 11" spaceflight mission entities are preferred since they are clearly strongly related to each other and to the other entities found in the document, i.e., Buzz Aldrin and John F. Kennedy.

The effectiveness of the *entity relatedness* function adopted is thus a key-point for the accuracy of any entity-linking algorithm. In this work we investigate to which extent a machine learning approach can be exploited to devise a high-quality entity relatedness function. The main contributions presented in this Chapter are:

- a formalization of the problem of devising high-quality entity relatedness functions as a learning-to-rank problem;
- a novel technique to build benchmark datasets for learning and testing entity relatedness functions;
- an extensive experimentation showing that our automatically learned function outperforms state-of-the-art relatedness functions. More importantly, our approach can improve the performance of a whole class of entity-linking algorithms;

[1]Throughout this chapter, we use the 04/03/2013 dump, available at `http://dumps.wikimedia.org/enwiki/20130403/enwiki-20130403-pages-articles.xml.bz2`

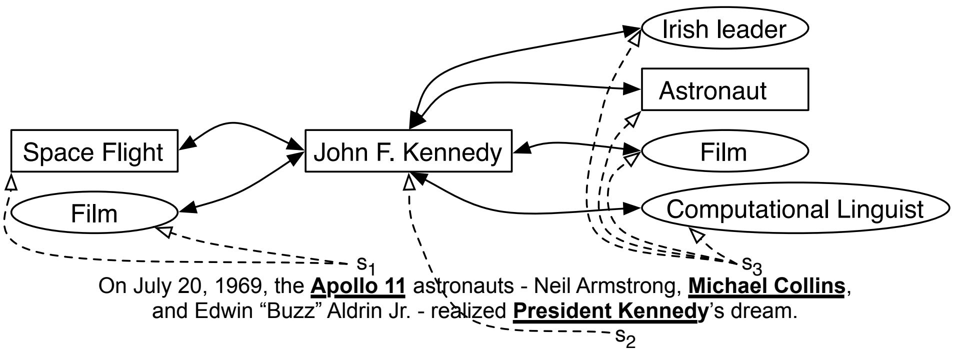

Irish leader
Astronaut
Space Flight
John F. Kennedy
Film
Film
Computational Linguist
s1
s3
s2
On July 20, 1969, the Apollo 11 astronauts - Neil Armstrong, Michael Collins,
and Edwin "Buzz" Aldrin Jr. - realized President Kennedy's dream.

Figure 4.1: Entity relationships for spotting and disambiguation. Three spots extracted from the above example are underlined: s_1 = "*Apollo* 11", s_2 = "*President Kennedy*", and s_3 = "*Michael Collins*". The graph shows relatedness edges connecting candidates entities. For simplicity of representation the candidates for spot s_3 are omitted. Rectangles are used to indicate correctly disambiguated entities, while ellipses refer to other candidate entities.

- an open source, publicly available framework for addressing the entity linking problem and evaluating new algorithms in a fair test environment.

The Chapter is organized as follows. In Section 4.2 we formalize the problem of learning automatically a entity relatedness function. In Section 4.3 we discuss related works, and how entity relatedness functions are used in the proposed approaches. In Section 4.4 we evaluate some machine learned entity relatedness functions, and in Section 4.5 we evaluate their impact on entity linking algorithms. Finally, in Section 4.6 we provide a summary of the Chapter.

4.2 Entity Relatedness Discovery

Given a set of known entities $\mathcal{E}$ from a knowledge base $\mathcal{KB}$, and an unstructured text document D, entity linking aims at identifying all the relevant mentions in D to the entities of $\mathcal{E}$. The entity linking process involves two steps that we are going to detail in the following.

Spotting and Candidate Selection. Spotting aims at identifying spots, i.e., contiguous sequences of n terms (*n-grams*) occurring in D that might mention some entity $e \in \mathcal{E}$. A common method to identify the spots $S_D = \{s_1, s_2, \ldots\}$ is to exploit a *controlled vocabulary* of spots $\mathcal{L}$, and to search the input document for the n-grams that exactly match an entry of this vocabulary.

When Wikipedia is used as $\mathcal{KB}$, each Wikipedia article identifies an entity, and the vocabulary $\mathcal{L}$ can be easily built by considering the article *titles* along with the *anchor texts* of all internal Wikipedia hyperlinks.

Each spot $s_i \in S_D$ is then associated with a set of candidate entities $C(s_i) \subseteq \mathcal{E}$. This is done by considering all the entities of $\mathcal{E}$ that are referred to in $\mathcal{KB}$ by using spot s_i as an anchor text. Unfortunately the same spot s_i can occur in different places of $\mathcal{KB}$ (and even of D!) and refer to distinct entities. Finally, we denote by $\epsilon(s_i) \in C(s_i)$ the entity that is actually mentioned by s_i in D.

Figure 4.1 illustrates the three spots $\{s_1, s_2, s_3\}$ detected in our text example. For each s_i, the outgoing dashed directed edges identify the set of candidate entities $C(s_i)$, where $\epsilon(s_i) \in C(s_i)$, i.e., the entity that is actually mentioned by s_i, is represented as a rectangle.

To limit the set of spots and candidate entities to the most meaningful ones, *link probability* and *commonness* properties can be usefully exploited [105]. The link probability for a spot s_i is defined as the number of times s_i occurs as a mention in $\mathcal{KB}$, divided by its total number of occurrences. This permits to discard spots that are rarely used as a mention to a relevant entity. For example the spot "July 20", introduced in the example of Section 4.1, occurs hundreds of times in Wikipedia, and, even if it is the title of an article, only in a few cases it used as anchor text.

The *commonness* of a candidate $c \in C(s_i)$ for spot s_i is instead defined as the fraction between the number of occurrences of s_i in $\mathcal{KB}$ actually referring to c, and the total number of occurrences of s_i in $\mathcal{KB}$ as a mention to an entity. For example, the spot "Michael Collins" may refer to more than 20 different entities, but the Irish revolutionary leader (421 mentions, commonness 0.5), the film about his life (126 mentions, commonness 0.15) and the astronaut (132 mentions, commonness 0.15) are largely the most common.

Setting a threshold on minimum linking probability and minimum commonness has been proven to be a simple and effective strategy to limit the number of spots and associated candidates, without harming the recall of the entity linking process [110].

Disambiguation and Linking. Since in many cases we have several candidates for a single spot s_i (i.e., $|C(s_i)| > 1$), the spot has to be disambiguated by choosing the right entity $\epsilon(s_i)$ among the candidates $C(s_i)$. For each spot, a disambiguation algorithm outputs the selected entity and a confidence score. This confidence score can be used to select the most likely matching entities, and to trade precision with recall.

In order to choose the best entity for a spot, disambiguation may exploit different signals and features. These include commonness and linking probability, and many others features considering the text surrounding the spot, and the other spots of the document. The most important of such features is *entity relatedness*, usually defined as a real function $\rho : \mathcal{E} \times \mathcal{E} \rightarrow [0,1]$, where 0 and 1 are the minimum and maximum relatedness measure, respectively. To guarantee the accuracy of entity linking, the entities selected by the disambiguation process to be linked to the detected spots have in fact to be strongly related to each other.

Figure 4.1 shows the relatedness graph referred to our example. The selection of the best entities is often implemented on top of this relatedness graph, where edges are weighted by some entity relatedness function. Therefore, the role of such entity relatedness function is crucial for the accuracy of the disambiguation process.

Even if the definition of a entity relatedness function is not a trivial task, several works agree on the effectiveness of the Wikipedia-based relatedness function proposed by Milne and Witten [110, 109]. The relatedness between two entities a and b is in this case computed by exploiting the graph structure of Wikipedia:

$$\rho^{\mathsf{MW}}(a,b) = 1 - \frac{\log(\max(|in(a)|,|in(b)|)) - \log(|in(a) \cap in(b)|)}{\log(|W|) - \log(\min(|in(a)|,|in(b)|))}$$

where W is the set of all Wikipedia entities, while $in(a)$ and $in(b)$ are the sets of Wikipedia articles linking to a and b, respectively. When $|in(a) \cap in(b)| = 0$, we have $\rho^{\mathsf{MW}}(a,b) = 0$. In addition, ρ^{MW} is maximum (equal to 1) when $in(a) \cap in(b) = in(a) = in(b)$, and thus all the articles that cite a also cite b, and vice versa.

The ρ^{MW} function, promoting entities that are co-cited by the same Wikipedia articles, is considered the *state-of-the-art* relatedness measure, adopted also in [57, 89, 68]. On the other hand, there is no guarantee that ρ^{MW} would produce a proper scoring of the candidate entities.

Example 4.2.1. *Given the entities a= "Andronicus of Rhodes", b= "Chondrichthyes", and c= "Aristotle" occurring in a document, we have that $\rho^{\mathsf{MW}}(a,b) = 0.54$ and $\rho^{\mathsf{MW}}(a,c) = 0.56$[2]. The connection between entities a and c is very strong since Andronicus of Rhodes is credited with the production of the first reliable edition of Aristotle's works. The (unexpected) high relatedness score between entities a and b is instead due to a single co-citing Wikipedia article (which is c) that reports about Aristotle's studies of a group of fishes he named selachians, a.k.a. chondrichthyes. Therefore, in this case a single co-citation is enough to produce an unexpected high value $\rho^{\mathsf{MW}}(a,b)$, which is similar to the expected large value of $\rho^{\mathsf{MW}}(a,c)$.*

[2]The values to compute the two ρ^{MW} measures are: $|in(a)| = 24$, $|in(b)| = 261$, $|in(c)| = 3502$, $|in(a) \cap in(b)| = 1$, $|in(a) \cap in(c)| = 17$, and $|W| = 4,255,306$.

Another interesting observation is that ρ^{MW} is symmetric: Andronicus of Rhodes is relevant to Aristotle, to the same degree Aristotle is relevant to Andronicus of Rhodes.

Our claim is that a good entity relatedness function ρ can improve the performance of a large class of entity linking algorithms. We propose a set of properties that an optimal entity relatedness measure should satisfy, and we formalize the problem of discovering a good entity relatedness function into a learning-to-rank problem.

Relatedness as a Ranking Function. Suppose that an entity linking algorithm identifies only two spots s_h and s_i for a document D, and for these spots it generates the two sets of candidate entities $C(s_h)$ and $C(s_i)$ respectively. Most disambiguation algorithms assume that if one of the candidate entities in $C(s_h)$ is highly related to another entity in $C(s_i)$, then it is very likely that they are the entities $\epsilon(s_h)$ and $\epsilon(s_i)$ actually mentioned by the two spots.

We claim that a good entity relatedness function ρ should promote the relatedness of correct entities: given entity $\epsilon(s_h)$, its relatedness with $\epsilon(s_i)$ should be larger then that with any other candidate in $C(s_i)$. This should hold for every spot $s_i \neq s_h$.

Proposition 4.2.1. *Given D, $S_D = \{s_1, s_2, \ldots\}$, and, for each spot s_i, $C(s_i)$ and $\epsilon(s_i)$, a relatedness function ρ improves entity-linking accuracy if the following constraint holds:*

$$\begin{aligned} &\forall s_h \in S_D,\ \ \forall s_i \in S_D \setminus \{s_h\},\ \ \forall c \in\ C(s_i) \setminus \{\epsilon(s_i)\} : \\ &\qquad\qquad \rho(\epsilon(s_h), \epsilon(s_i))\ >\ \rho(\epsilon(s_h), c). \end{aligned} \tag{4.1}$$

Indeed, the constraint in Eq. 4.1 nicely fits into a learning-to-rank based formulation [80]. The relatedness function ρ can be in fact modeled as a ranking function, with entity $\epsilon(s_h)$ used as a query. According to the above Proposition, function ρ should score all the entities actually mentioned in the document, i.e. $\epsilon(s_i)$ for all $s_i \neq s_h$, higher than any other false-positive candidate, i.e. $c \in C(s_i) \setminus \{\epsilon(s_i)\})$ for all $s_i \neq s_h$.

Given document D and spots $S_D = \{s_1, s_2, \ldots, s_h, \ldots\}$, we denote by $\mathcal{R}_D^h = \cup_{i \neq h} C(s_i)$ the set of candidates to be ranked for query $\epsilon(s_h)$, and by $\mathcal{E}_D^h = \cup_{i \neq h} \epsilon(s_i)$ the set of relevant entities for the query, where $\mathcal{E}_D^h \subseteq \mathcal{R}_D^h$. From an information retrieval perspective, items in $\mathcal{R}_D^h$ are relevant for query $\epsilon(s_h)$ if and only if they belongs to $\mathcal{E}_D^h$. Let us denote with π_ρ^h the score descending ordering of $\mathcal{R}_D^h$ induced by our ranking relatedness function ρ for query $\epsilon(s_h)$. According to Proposition 4.2.1, a scored list π_ρ^h is effective when entities in $\mathcal{E}_D^h$ are in the top positions of the list. We can thus measure the effectiveness of our ranking relatedness function by using common information retrieval quality metrics such as NDCG [79]. In our context we define $\mathsf{DCG}(\pi^h_\rho)$ as:

$$\mathsf{DCG}(\pi^h_\rho) = \sum_{j=1}^{|\pi^h_\rho|} \frac{[\![\pi^h_\rho[j] \in \mathcal{E}^h_D]\!]}{\log(j+1)}$$

where $\pi^h_\rho[j]$ denotes the j-th item of the scored list, and $[\![x]\!]$ equals 1 if x is true and 0 otherwise. NDCG is defined as the usual normalized version of DCG.

We can now introduce the *Entity Relatedness Discovery* problem we are going to address in this paper by means of a learning-to-rank approach.

Problem 4.2.1 (Entity Relatedness Discovery)**.**
Let $\mathcal{D}$ be a collection of entity-linked documents, where for each document $D \in \mathcal{D}$ and every relevant spot s_i of S_D we know $\epsilon(s_i)$. Given the entity $\epsilon(s_h)$ and the set $\mathcal{R}^h_D$, a ranking relatedness function ρ induces an ordering π^h_ρ of $\mathcal{R}^h_D$.
The Entity Relatedness Discovery Problem *requires to find the function ρ that maximizes the ranking quality:*

$$\frac{1}{|\mathcal{D}|}\sum_{D\in\mathcal{D}}\frac{1}{|\mathcal{S}_\mathcal{D}|}\sum_{s_h\in S_D}\mathsf{NDCG}(\pi^h_\rho)$$

In our experiments, we chose to optimize NDCG to find a good entity relatedness function, but we used several other ranking quality functions to assess the goodness of results.

Unlike previous approaches, we do not suggest any new entity relatedness function. Rather, we define a learning-to-rank framework to discover the optimal entity relatedness function.

4.3 Related works

In the following we discuss how the notion of *entity relatedness* is exploited by state-of-the-art entity linking algorithms. Emphasis is given to the solutions proposed in [110] and [89] and [57] which are the most relevant proposals in the field, and they are all adopting ρ^{MW} as entity relatedness function. We show that the entity relatedness function defined in Proposition 4.2.1 can replace ρ^{MW} since it fits better the framework and the objectives of the above algorithms.

WikiMiner [110]. Given a document D, let us consider its spots S_D and for each spot s_i the associated set of candidates $C(s_i)$. Let us suppose that a subset of the spots are associated with only a single entity. We denote with $U \subseteq \mathcal{E}$ the *context*: the set of unambiguous entities linked to spots in S_D, i.e., $U = \bigcup_{|C(s_i)|=1} \epsilon(s_i)$. The WikiMiner algorithm exploits the entities in U as safe reference points to help the disambiguation of the other ambiguous spots for which $|C(s_i)| > 1$ holds. The idea is to select for every ambiguous spot of S_D the entity which is, on average, the most related with the "safe" entities in U. The relatedness function adopted is ρ^{MW}. It is worth noting that not all the entities in U have the same impact: an entity $u \in U$ is in fact considered of high quality if it is strongly related to the other entities in U, and if the *link probability* of the corresponding spot is high. These two criteria allow a weight w_u, $0 \leq w_u \leq 1$ to be assigned to each entity u in U. Note that the main aim of this weight is to reduce the impact of low-quality entities occurring in U. When applied to our simple example, the low resulting weight would demote the importance of the safe entity "July 20".

Every candidate c in $C(s_i)$ is scored according to the following function:

$$score(c \mid U) = \frac{1}{\sum_u w_u} \sum_{u \in U} w_u \cdot \rho^{\mathsf{MW}}(u, c). \tag{4.2}$$

It is easy to show that the accuracy of the disambiguation would improve if we adopted, instead of ρ^{MW}, a relatedness function ρ that satisfies our Proposition 4.2.1.

Given an entity $u \in U$, we can rewrite Eq. 4.1 and derive as follows:

$$\begin{aligned} \rho(u, \epsilon(s_i)) &> \rho(u, c) \quad \Rightarrow \\ \frac{1}{\sum_u w_u} \sum_{u \in U} w_u \cdot \rho(u, \epsilon(s_i)) &> \frac{1}{\sum_u w_u} \sum_{u \in U} w_u \cdot \rho(u, c) \quad \Rightarrow \\ score(\epsilon(s_i) \mid U) &> score(c \mid U) \end{aligned}$$

Therefore, a relatedness function satisfying Proposition 4.2.1 would always correctly rank entity $\epsilon(s_i)$ higher than any other candidate for the corresponding spot s_i even when integrated in the WikiMiner framework.

Interestingly, the authors of [110] use machine learning to combine the above relatedness score with other two features: *commonness* and *context quality* (measured as $\sum w_u$). They experiment with a training set built from 500 Wikipedia articles. However, machine learning is not exploited to improve the relatedness function as in our proposal.

Referent Graph [89]. *Referent Graph*, a graph-based method still exploiting the relatedness function ρ^{MW}, is proposed in [89]. Let $RG(V, E)$ be a weighted directed graph where the nodes includes all the spots s_i of S_D and candidates $C(s_i)$. RG has a directed edge from s_i to every $c \in C(s_i)$, and reciprocal edges connecting every pair of candidate entities a and b, $a \in C(s_i), b \in C(s_h), i \neq h$. Spot-candidate edges (s_i, c) are weighted according to the cosine similarity between the Wikipedia article corresponding to entity c and a local context window of 50 words around the spot s_i. The candidate-candidate edges (a, b) are weighted by using ρ^{MW}. Finally, weights are normalized so that weights on outgoing edges from a given node always sum up to 1. The graph shown in Figure 4.1 is a toy referent graph, where relatedness edges connecting candidates entities for the spot s_1 with candidates entities for the spot s_3 are omitted for clarity.

The score of a candidate entity for a given spot is given by the steady state distribution of a random walk with restarts [114] in RG, where candidate nodes have restart probability 0, and spot nodes have a restart probability proportional to their inverse document frequency score in the Wikipedia corpus. Also in this case, assigning a different restart probability to spot nodes, and weighting as above explained spot-candidate edges is aimed to limit the impact of non relevant or incorrectly matched mentions.

The rationale of the random walk approach is to evaluate the relationships among the whole set of candidates simultaneously, in contrast to previous methods where the scores of candidate entities are assigned independently of each other. Also in this algorithm the choice of the entity relatedness function ρ has a strong impact on the performance since it drives the random walk process. A set of entities being very related to each other is likely to produce a reinforcement loop, and eventually include the most probable states of the random walk.

Even if we do not provide a formal proof as for WikiMiner, it is clear that a good relatedness function should promote the reciprocal relatedness among the right entities in the graph, thus helping the random walk to converge to the correct ranking of candidates.

A similar approach is used in [159], where a slightly differently weighted referent graph is pruned progressively by removing iteratively the node with the lowest weighted degree (sum of the weights of incoming edges). Even in this case, the weights of candidate-candidate edges are computed with the ρ^{MW} relatedness function. The paper do not compare performances with those of [110], [89], or other algorithms, and it is thus difficult to estimate the impact of this proposal.

TAGME [57]. TAGME is an annotation framework focussing on efficiency that exploits two main features: *commonness* and the ρ^{MW} relatedness. First, candidate entities for a spot s_i are ranked according to their average relatedness with other candidate entities for spots $s_j \neq s_i$, weighted by their commonness.

Then, from the top 30% candidates of the resulting ranked list, the entity with the largest commonness is finally selected.

Also this algorithm would benefit by a relatedness function satisfying Proposition 4.2.1, since it would help to boost the score of the actual entities mentioned in the document. However, the benefit is limited, since the relatedness function impacts more on the pruning irrelevant candidates, while the final choice of the best entity is mainly driven by the commonness feature.

Other approaches. Relatedness function ρ^{MW} is partially inspired by the so-called Normalized Google Distance (NGD) [48], which borrows from Kolmogorov complexity and information distance concepts. While NDG is tailored to measure similarity between words or phrases, ρ^{MW} measure is specifically tailored to entities represented in a graph structure such as the one of Wikipedia. In [59] another Wikipedia-based relatedness measure, named ESA, is proposed. A word is represented in a high dimensional space by considering for each Wikipedia article the relevances of the word in the article, and by summing such score vectors for longer text fragments. Also [69] investigates new text-based relatedness measures that try to go beyond link-based similarities. The study conducted in [109] shows however that ESA has a performance similar to that of ρ^{MW}, with the latter being much cheaper to be computed since it does not require to index the whole Wikipedia textual content. In [127], the authors improve only slightly the solution proposed in [50], but they do not provide any comparison with [110, 89].

The authors of [149] propose a machine learning approach to rank entity-based facets related to a given Web search query. Since the paper focuses on a special set of entities, such as monument and celebrities, the presented technique exploits information coming from image search queries and Flickr image tags. The goal of [149] is not to discover the degree of relatedness between entities, but rather to suggest entities that are most likely to generate a large click through.

Aida [159] exploits a model similar to the one proposed in [89], where spots and entities are both nodes in a graph and there are edges between spots and entities and between couples of entities. The weights over the edges are computed: i) from a spot s to an entity e: with a linear combination between the prior probability $p(e|s)$ that s links to e, and a similarity measure between the spot and the textual context of the entity (anchors, title, relevant words..) ii) from entity e to entity c using the ρ^{MW} relatedness measure. Instead of a random walk, they propose an iterative process where in each step they compute for each node e the sum of the scores of the incoming nodes $in(e)$. At the end of each step the entity with lowest score is removed, until only one entity for spot remains.

4.4 Entity Relatedness Evaluation

In the following we describe the methodology adopted to build a reference dataset for the learning process, the feature used to describe entities, and finally the performance of two automatically learned relatedness functions.

4.4.1 Building a benchmark dataset

In order to evaluate the impact of different relatedness functions, we built a benchmark dataset for Problem 4.2.1. This dataset, used to train and test our relatedness function, contains a set of tuples in the form $\langle \epsilon, \mathcal{R}_\epsilon, \mathcal{E}_\epsilon \rangle$, where ϵ is an entity occurring in a document D, $\mathcal{R}_\epsilon$ is a set of *candidate entities* possibly occurring in D, and $\mathcal{E}_\epsilon \subseteq \mathcal{R}_\epsilon$ are the *relevant entities* occurring in D besides ϵ.

In order to build these tuples, we need both positive and negative examples, i.e., positive ones from $\mathcal{E}_\epsilon$ and negative ones from $\mathcal{R}_\epsilon \setminus \mathcal{E}_\epsilon$. In most entity-annotated datasets, each document is annotated by one or more human assessors, who manually performed some kind of spotting and entity disambiguation tasks. Therefore, for each document D we only have positive examples, i.e. the set $\mathcal{A}_D$ of entities actually occurring in D. In addition, we do not know the spot in D that actually mentions each entity in $\mathcal{A}_D$.

Hence, to generate our dataset for training our relatedness function, we have to devise a sort of reverse annotation process, aimed at discovering the spots associated with the known entities, and the potential candidates of such spots. In this way, we identify also the negative examples to build the tuples $\langle \epsilon, \mathcal{R}_\epsilon, \mathcal{E}_\epsilon \rangle$. In more detail, we generate our benchmark dataset as described below:

1. we set up a knowledge base $\mathcal{KB}$ of entities based on Wikipedia. This contains entities, their mentions, i.e, anchor text of incoming links and page title, and the hyper-link structure; we created a vocabulary of entity mentions $\mathcal{L}$ containing only spots with link probability larger than 2%. Finally, for each spot we disregarded entities with commonness smaller than 3%;

2. we generate all *n-grams* of every given document D, with $n \leq 6$, and we match them against $\mathcal{L}$ to devise the spots S_D;

3. for each spot s_i in S_D, we retrieve the candidate entities $C(s_i)$ as the set of entities linked in Wikipedia by the same n-gram;

4. we finally consider the set of relevant entities $\mathcal{A}_D$ of D, as annotated by the human assessors. Since we do not know the real association between

each spot s_i and the human annotated entities in $\mathcal{A}_D$, for each spot s_i we look for the actual entity $\epsilon(s_i)$ in the set $C(s_i) \cap \mathcal{A}_D$. If $C(s_i) \cap \mathcal{A}_D = \emptyset$, we assume that $\epsilon(s_i)$ is not known, and thus discard s_i. If $|C(s_i) \cap \mathcal{A}_D| > 1$[3], we also throw away s_i, since we are not able to disambiguate. Finally, if $|C(s_i) \cap \mathcal{A}_D| = 1$, then $\epsilon(s_i) \in C(s_i) \cap \mathcal{A}_D$ is the actual entity to link to s_i.

At the end of the process, for each document D we have: a set of spots S_D and, for each spot s_i, a set of candidate entities $C(s_i)$ and also the mentioned entity $\epsilon(s_i)$.

Thus, for every spot s_h of every document D, we can generate a tuple $\langle \epsilon, \mathcal{R}_\epsilon, \mathcal{E}_\epsilon \rangle$ for the benchmark dataset that contains: (i) the actually mentioned entity $\epsilon(s_i)$, (ii) the set of candidate entities for every other spot in the document, and (iii) the set of correctly linked, and thus related, entities in the document. By assuming that close spots are more likely to be related, we did not consider in this tuple generation step those spots occurring at a distance larger than $\omega = 150$ characters from the current spot associated with entity ϵ.

In our experiments we used a subset of the CoNLL 2003 entity recognition [68] task dataset, which includes annotated news stories of the Reuters Corpus V1. The dataset contains 1494 documents with an average length of 187 terms. Each document contains on average 11.7 entities.

We processed the corpus as explained above, and we thus built a dataset for evaluating the relatedness containing over 1.6 million tuples. We split the tuples in training, validation, and test set, respectively containing $977,514$, $369,798$ and $302,529$ records. Please observe that we take care of producing each dataset from a disjoint subset of documents in the collection, so that the tuples in the training and test sets were actually generated from a different subset of documents.

4.4.2 Features

A pair of entities a and b, for which the relatedness $\rho(a,b)$ has to be estimated, is represented by a set of 27 features shown in Table 4.1. The choice of such features is driven by the following considerations. First, we want to maximize their applicability by using publicly available data, and by using measures that can be easily applied to other entity knowledge bases, e.g., FreeBase.[4] For this reason we do not use click-through, access log, or query log based data, which are very difficult to obtain. We use instead several features related to

[3]In our datasets, this happens in only 2% of the cases.

[4]`http://www.freebase.com/`

the link structure of our knowledge base, such as the number of in-links $in(e)$ and out-links $out(e)$ of an entity e.

Second, there are applications of the entity relatedness function where the concept of spot is not applicable. Consider, for instance, the case of related entity recommendation where the query is a entity that is not associated with any spot. Therefore, we do not include features such as *link probability* and *commonness*.

Finally, we do not include text-based similarity measures, such as cosine similarity between Wikipedia articles pages, because this kind of approaches have been proven to perform similarly to the ρ^{MW} measure, but are much more computationally expensive [109].

Note that, by using the proposed machine learning approach, the feature set we adopt can be easily enriched with any additional feature, or by analyzing any other different knowledge base.

We categorize the features listed in Table 4.1 in three categories: *singleton*, *asymmetric* and *symmetric*.

Singleton features regard a single entity. They include only frequency and entropy, computed on the basis of the frequency of Wikipedia links to the entity article page. These features are computed for both entities of a given pair (a, b), resulting in four scores.

We claim that a relatedness function should not be symmetric. Consider for example the entities *Neil Armstrong* and *United States of America*: it seems reasonable that the relatedness of *United States of America* given *Neil Armstrong* is greater than the relatedness of *Neil Armstrong* given the *United States of America*. For this reason we included five asymmetric features, which are computed in both directions of the pair, resulting in ten scores.

Last, we considered 13 symmetric features, such as ρ^{MW}. Some of these features derive from asymmetric ones, and others are variations computed by considering outgoing links of an entity instead of incoming ones.

All the above features are computed on the basis of the same Wikipedia dump mentioned in the Section 4.1. Therefore, features are not extracted on the training or test dataset.

4.4.3 Quality of entity relatedness

To solve the Entity Relatedness Discovery problem, we used an existing tool for learning ranking functions, named RankLib.[5] This includes the implementation of several effective algorithms. We report the results of the two most effective: Gradient-Boosted Regression Trees [58] and LambdaMart [156]. We denote the models built with those algorithm ρ^{GBRT} and $\rho^{\lambda\mathsf{MART}}$.

[5]`http://people.cs.umass.edu/~vdang/ranklib.html`

Singleton Features	
P(a)	probability of a mention to entity a: $P(a) = \|in(a)\|/\|W\|$.
H(a)	entropy of a: $H(a) = -P(a)\log(P(a)) - (1-P(a))\log(1-P(a))$.
Asymmetric Features	
P(a—b)	conditional probability of the entity a given b: $P(a\|b) = \|in(a) \cap in(b)\| \ / \ \|in(b)\|$.
$\mathsf{Link}(a \to b)$	equals 1 if a links to b, and 0 otherwise.
$P(a \to b)$	probability that a links to b: equals $1/\|out(a)\|$ if a links to b, and 0 otherwise.
$\mathsf{Friend}(a, b)$	equals 1 if a links to b, and $\|out(a) \cap in(b)\|/\|out(a)\|$ otherwise.
$KL(a\\|b)$	Kullback-Leibler divergence: $KL(a\\|b) = \log\frac{P(a)}{P(b)}P(a) + \log\frac{1-P(a)}{1-P(b)}(1-P(a))$.
Symmetric Features	
$\rho^{\mathsf{MW}}\ (a, b)$	co-citatation based similarity [110].
$J(a, b)$	Jaccard similarity: $J(a, b) = \frac{in(a) \cap in(b)}{in(a) \cup in(b)}$.
$P(a, b)$	joint probability of entities a and b: $P(a, b) = P(a\|b) \cdot P(b) = P(b\|a) \cdot P(a)$.
$\mathsf{Link}(a \leftrightarrow b)$	equals 1 if a links to b and vice versa, 0 otherwise.
$\mathsf{AvgFr}(a, b)$	average friendship: $(\mathsf{Friend}(a, b) + \mathsf{Friend}(b, a))/2$.
$\rho^{\mathsf{MW}}_{\mathsf{out}}(a, b)$	ρ^{MW} considering outgoing links.
$\rho^{\mathsf{MW}}_{\mathsf{in\text{-}out}}(a, b)$	ρ^{MW} considering the union of the incoming and outgoing links.
$J_{\mathsf{out}}(a, b)$	Jaccard similarity considering the outgoing links.
$J_{\mathsf{in\text{-}out}}(a, b)$	Jaccard similarity considering the union of the incoming and outgoing links.
$\chi^2(a, b)$	χ^2 statistic: $\chi^2(a, b) = (\|in(b) \cap in(a)\| \cdot (\|W\| - \|in(b) \cup in(a)\|) + -\|in(b) \setminus in(a)\| \cdot \|in(a) \setminus in(b)\|)^2 \cdot \cdot \frac{\|W\|}{\|in(a)\| \cdot \|in(b)\|(\|W\| - \|in(a)\|)(\|W\| - \|in(b)\|)}$
$\chi^2_{\mathsf{out}}(a, b)$	χ^2 statistic considering the outgoing links.
$\chi^2_{\mathsf{in\text{-}out}}(a, b)$	χ^2 statistic considering the union of the incoming and outgoing links.
$\mathsf{PMI}(a, b)$	point-wise mutual information: $\log\frac{P(b\|a)}{P(b)} = \log\frac{P(a\|b)}{P(a)} = \log\frac{\|in(b) \cap in(a)\|\|W\|}{\|in(b)\|\|in(a)\|}$

Table 4.1: Features for entity relatedness learning.

Features	Rank	NDCG@5	NDCG@10	P@5	P@10	MRR
$\mathrm{P}(c\|e)$	**1**	**0.68**	**0.72**	**0.47**	**0.33**	**0.80**
J(e, c)	**2**	0.62	0.66	0.44	0.31	0.75
Friend(e,c)	24	0.59	0.64	0.42	0.31	0.71
$\rho^{\mathsf{MW}}(e,c)$	19	0.59	0.63	0.42	0.31	0.72
$J_{\mathsf{in-out}}(e,c)$	26	0.60	0.63	0.42	0.30	0.74
$\mathsf{AvgFr}(e,c)$	**3**	0.57	0.62	0.40	0.30	0.69
P(e,c)	27	0.56	0.60	0.39	0.28	0.70
$\rho^{\mathsf{MW}}_{\mathsf{in\text{-}out}}(a,b)$	9	0.56	0.60	0.40	0.29	0.71
$J_{\mathsf{in-out}}(e,c)$	**4**	0.54	0.58	0.39	0.28	0.67
$\rho^{\mathsf{MW}}_{\mathsf{out}}(a,b)$	17	0.52	0.55	0.37	0.27	0.65
$\chi^2(e,c)$	25	0.51	0.55	0.37	0.27	0.64
$\mathrm{P}(e\|c)$	22	0.48	0.54	0.36	0.28	0.60
H(c)	**5**	0.48	0.51	0.30	0.20	0.68
$\chi^2_{\mathsf{out}}(e,c)$	16	0.47	0.50	0.34	0.24	0.61
$\mathsf{AvgFr}(c,e)$	21	0.44	0.49	0.33	0.25	0.56
P(c)	13	0.47	0.49	0.29	0.19	0.66
$\mathsf{PMI}(e,c)$	23	0.42	0.48	0.32	0.25	0.53
$\chi^2_{\mathsf{in-out}}(e,c)$	11	0.44	0.46	0.33	0.23	0.58
$P(e \rightarrow c)$	18	0.37	0.38	0.24	0.15	0.55
$\mathsf{Link}(e \rightarrow c)$	20	0.37	0.38	0.24	0.15	0.55
$P(c \rightarrow e)$	12	0.35	0.36	0.22	0.14	0.52
$\mathsf{Link}(c \rightarrow e)$	15	0.31	0.33	0.21	0.14	0.46
$KL(c\|\|e)$	10	0.32	0.32	0.19	0.12	0.51
$\mathsf{Link}(c \leftrightarrow e)$	14	0.28	0.29	0.17	0.11	0.45
$KL(e\|\|c)$	8	0.26	0.28	0.17	0.11	0.44
P(e)	6	0.08	0.11	0.06	0.06	0.17
H(e)	7	0.08	0.11	0.06	0.06	0.17

Table 4.2: Entity ranking performance with a single feature. Features are sorted by *NDCG*@10.

Features	NDCG@5	NDCG@10	P@1	P@5	P@10	MRR
ρ^{MW}	0.59	0.63	0.62	0.42	0.31	0.72
$\rho^{\lambda\mathsf{MART}}$	**0.75**	**0.79**	**0.80**	**0.51**	**0.36**	**0.87**
ρ^{GBRT}	**0.75**	0.78	**0.80**	**0.51**	0.35	0.86

Table 4.3: Entity ranking performance of learned relatedness functions.

Note that the two models differ significantly in the objective function being optimized. The $\rho^{\lambda\mathsf{MART}}$ model was built by a list-wise algorithm and minimizing *NDCG*@10. This is indeed in perfect agreement with our definition of entity relatedness problem, and with the benchmark created. On the other hand, the ρ^{GBRT} model optimizes the error in predicting the class label (i.e., relevant vs. not relevant) of a given instance. Therefore, the prediction can be used to produce a ranking, but the model does not optimize the ranking directly.

In Table 4.3 we report the performance of the two relatedness functions ρ^{GBRT} and $\rho^{\lambda\mathsf{MART}}$, and compare it against ρ^{MW}. The improvement of using a machine learned function that exploits 27 features is apparent with every ranking quality measure adopted. If we consider *NDCG*@10, ρ^{MW} improves over $\rho^{\lambda\mathsf{MART}}$ by a factor of 25% (it is worth to observe that, for how the dataset was built, the maximum NDCG obtainable is 1). The two learned functions have very similar performance, with no significant difference. Please note that we also experienced other objective functions (MRR, MAP) obtaining similar performance.

In order to gain some insight on the learned functions, and on the role of the different features, we run a study based on a naïve feature selection algorithm [61]. This algorithm ranks features by leveraging their similarity and the score of single-features models. It promotes effective features and demotes features similar to any other already selected one. Our objective here is not to find the best performing subset of features, but rather to investigate the importance of ρ^{MW} compared with other features not considered by state-of-the-art algorithm.

We measured the performance of the models built by means of LambdaMart algorithm when exploiting a single feature. In Table 4.2 we reported for each feature the score it can achieve. Recall that the relatedness function is required to learn a score of a candidate entity w.r.t. to a correct entity, which in the table are denoted with c and e respectively. Therefore, $P(c|e)$ is the conditional probability of finding the candidate entity c given our actually mentioned entity e, while $P(c|e)$ is the converse.

Results are very similar for every quality measure. Let's consider *NDCG* @10. The function ρ^{MW} is the fourth most effective feature with a score slightly

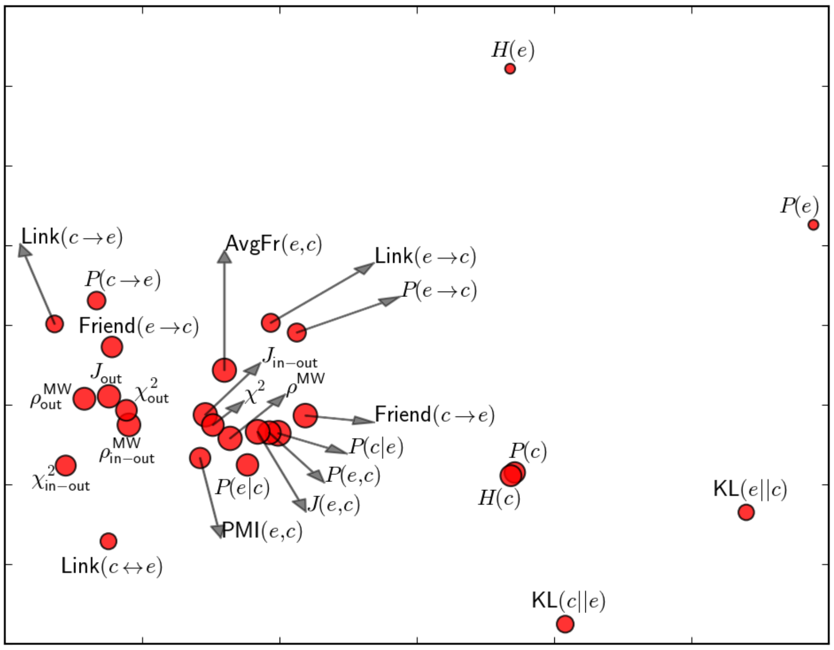


Figure 4.2: Multidimensional mapping of feature similarity computed using Kendall's τ coefficient. The size of each circle is proportional to the single-feature model score.

below that of Jaccard and Friend functions. The most effective feature is $P(c|e)$, that is the conditional probability of the finding a mention to entity c given a Wikipedia page that mentions the entity e. Note that this quite intuitive feature behaves largely better than ρ^{MW} with a score of .72, but it is however far from the score achievable with the full set of features. Also, note that statistic $P(c|e)$ comes from a collection being completely different from the test set, since it was computed on the Wikipedia corpus and not on the train collection. A third interesting property is the asymmetry of this feature.

The second column of Table 4.2 reports the rank assigned by feature selection algorithm. While $P(c|e)$ is ranked first being the most effective features, ρ^{MW} is ranked only 19-th. This is due to the heuristic strategy of the algorithm, which demotes features if they are similar to previously selected ones.

Figure 4.2 shows the result of a multidimensional scaling mapping of the 27 features into a 2-dimensional space, thus approximately preserving feature similarity. We measured the similarity between a feature pair according to the Kendall's τ coefficient. We can identify two interesting clusters. The first contains ρ^{MW} together with $J_{\text{in-out}}$ and χ^2, and, indeed, the first two have identical performance. The second cluster includes the two best performing features $P(c|e)$, $P(e, c)$ and also Jaccard similarity. Even if the features in

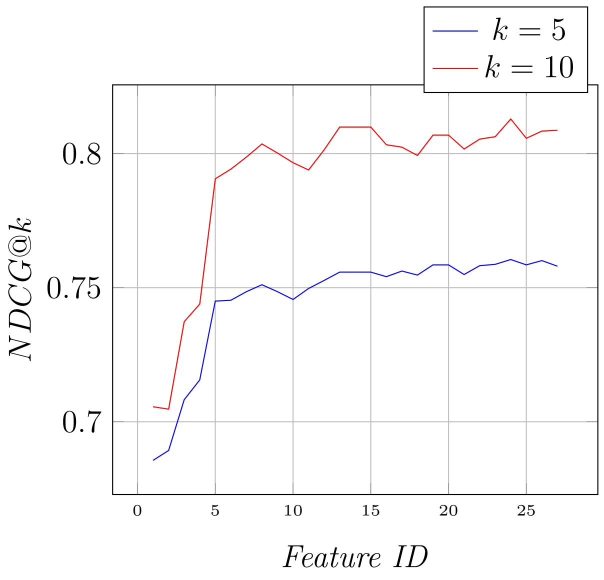


Figure 4.3: Incremental performance of $\rho^{\lambda\mathsf{MART}}$.

those clusters are similar w.r.t. the Kendall's τ coefficient, the score of the corresponding single feature model is very different, in particular for the best scoring $P(c|e)$. This suggest that the Kendall's τ coefficient may not be the best indicator in this context, and the feature selection may not be trivial.

Finally, in Figure 4.3 we measured the relative improvement provided by each feature. Features are sorted according to the ranking given feature selection algorithm mentioned above, and we measured the performance of the model by adding features incrementally. The model achieves almost optimal performance with the first 5 features. Optimal performance are achieved after 9 features are introduced in to the model. This shows that not all the features are necessary, and that a wisely chosen subset of features can provide optimal performance, or help in trading accuracy with efficiency. Several existing feature selection techniques can be used to this end. However, this is outside the scope of this work.

4.5 Impact on Entity Linking

We run a set of experiments to show how the automatically learned relatedness function can be profitably exploited by a class of entity disambiguation algorithms. We plugged the learned function into several annotation methods, which can be considered the state-of-the-art ones:

WikiMiner [110]. The method proposed by Milne and Witten that exploits the relatedness function to identify a subset of not ambiguous entities called *context*. Given an ambiguous spot, the relatedness function is employed again to select the entity that is more coherent with the context;

Referent Graph [66]. This method takes into account all the possible entities associated with the set of detected spots. The disambiguation is performed by modeling the entities as nodes of a complete graph, where the weight of each edge is the relatedness between the connected nodes;

TAGME [57]. This annotator computes the weighted average relatedness between an entity and all the other possible entities associated with the spots. It disambiguates the entity by selecting the most common entities in the subset of the possible meanings with the highest average relatedness with the others.

With the exception of WikiMiner, the source code of the frameworks proposed is not publicly available. Furthermore the code released is not easy to extend for implementing other annotators. Annotation depends on several subtasks, i.e., *(i)* process Wikipedia (parse the dump, generate the possible spots, filter stop-words, etc.); *(ii)* perform the spotting (relying on a dictionary or using a name entity recognition framework, like the Stanford Named Entity Recognizer[6]); *(iii)* disambiguate the ambiguous spots, and *(iv)* rank entity candidates.

It is worth to observe that a good performance obtained in the first tasks may heavily impact on the performance of the whole system, as well as using a different dump of Wikipedia (i.e., old dumps contain less entities, but also have less ambiguity for each spot), or a different commonness or link probability thresholds. For these reasons, we strongly believe that for this kind of research it is important to share a unique framework where these tasks are well separated and easy to isolate in order to study their performance. This would also allow us to experiment hybrid solutions combining subtask solutions of different methods (e.g., the TAGME spotter with the WikiMiner disambiguation algorithm).

We developed **Dexter** [46], an entity annotator framework, containing several utilities to manage the Wikipedia dump, a spotter based on the anchors and titles extracted from the dump, and data structures for retrieving all the features used by the annotators. Unlike WikiMiner, our framework does not rely on an external database to store the labels. In addition, during the execution it can maintain the model either on the disk or in main memory to

[6]`http://www-nlp.stanford.edu/software/CRF-NER.shtml`

	Referent Graph			TAGME			WikiMiner		
	ρ^{MW}	$\rho^{\lambda\mathsf{MART}}$	ρ^{GBRT}	ρ^{MW}	$\rho^{\lambda\mathsf{MART}}$	ρ^{GBRT}	ρ^{MW}	$\rho^{\lambda\mathsf{MART}}$	ρ^{GBRT}
$P@1$	0.59	0.68$_{+15\%}$	0.74$_{+25\%}$	0.78	0.81 $_{+4\%}$	0.80 $_{+3\%}$	0.78	**0.86**$_{+10\%}$	0.83 $_{+6\%}$
$P@5$	0.51	0.62$_{+22\%}$	0.61$_{+20\%}$	0.65	0.66 $_{+2\%}$	0.66 $_{+2\%}$	0.64	0.68 $_{+6\%}$	**0.69** $_{+8\%}$
$P@10$	0.44	0.50$_{+14\%}$	0.51$_{+16\%}$	0.50	0.50 $_{+0\%}$	0.51 $_{+2\%}$	0.50	0.51 $_{+2\%}$	**0.53** $_{+6\%}$
$iP_{r=0.10}$	0.76	0.84$_{+11\%}$	0.87$_{+14\%}$	0.87	0.89 $_{+2\%}$	0.89 $_{+2\%}$	0.88	**0.92** $_{+5\%}$	0.91 $_{+3\%}$
$iP_{r=0.50}$	0.55	0.69$_{+25\%}$	0.70$_{+27\%}$	0.67	0.68 $_{+1\%}$	0.69 $_{+3\%}$	0.66	0.73$_{+11\%}$	**0.77**$_{+17\%}$
$NDCG$	0.64	0.70 $_{+9\%}$	0.72$_{+13\%}$	0.68	0.69 $_{+1\%}$	0.69 $_{+1\%}$	0.66	0.72 $_{+9\%}$	**0.75**$_{+14\%}$
MRR	0.73	0.81$_{+11\%}$	0.84$_{+15\%}$	0.87	0.89 $_{+2\%}$	0.89 $_{+2\%}$	0.87	**0.92** $_{+6\%}$	0.90 $_{+3\%}$
$NDCG@5$	0.55	0.67$_{+22\%}$	0.68$_{+24\%}$	0.72	0.74 $_{+3\%}$	0.73 $_{+1\%}$	0.71	0.76 $_{+7\%}$	**0.77** $_{+8\%}$
$NDCG@10$	0.57	0.68$_{+19\%}$	0.70$_{+23\%}$	0.70	0.70 $_{+0\%}$	0.71 $_{+1\%}$	0.69	0.73 $_{+6\%}$	**0.75** $_{+9\%}$
$Recall$	0.76	**0.77** $_{+1\%}$	**0.77** $_{+1\%}$	0.68	0.69 $_{+1\%}$	0.69 $_{+1\%}$	0.64	0.70 $_{+9\%}$	0.75$_{+17\%}$
$Rprec$	0.46	0.58$_{+26\%}$	0.60$_{+30\%}$	0.56	0.58 $_{+4\%}$	0.58 $_{+4\%}$	0.56	0.60 $_{+7\%}$	**0.64**$_{+14\%}$

Table 4.4: Entity Linking performance

improve performance. The framework runs also on normal hardware, since we exploit efficient data structures in order to maintain compressed data in main memory.

We incorporated WikiMiner, TAGME, and Referent Graph in our framework, in order to verify if our relatedness function is able to improve the annotator performance. During the implementation, we slightly modified WikiMiner and TAGME: in WikiMiner we decided to rank the entities using a linear combination of commonness, link probability, and average relatedness with the context (the authors employed a classifier trained with several features that were heavy to retrieve); in TAGME we relied on our spotter that returns all the possible spots detected in the text, while in the original version the authors employ a specific policy for deleting spots in case of overlaps (we remove overlapping annotations at the end of the process, relying on the final ranking of the entities). We set the commonness threshold to 0.03 and we discard spots with link probability lower than 0.02.

Note that we are not interested in the absolute entity-linking performance of WikiMiner, TAGME, and Referent Graph, but rather on how the relatedness function impacts on the disambiguation process. For this reason, we implemented all the three algorithms within the same framework, and thus providing them with the output of the same spotter. For the same reason, the results of the Web services implementation of WikiMiner and TAGME are not reported. Those services use a different dump of Wikipedia, which is processed in a different way (e.g., tokenization, etc.), and they exploit a slightly different spotting algorithm, and this makes such results non significant within the scope of this work. However, it is important to report that we observed that our implementation always improves over the WikiMiner online service, and that it behaves only slightly worse then TAGME after the top 5 results, probably due to a different processing of Wikipedia.

We compared the results obtained by embedding different implementations of ρ: ρ^{MW}, $\rho^{\lambda\mathsf{MART}}$, and ρ^{GBRT}. Note that by embedding ρ^{MW} we are replicating the original algorithms that we consider as baselines to evaluate our proposed relatedness function.

The quality of the resulting algorithms is evaluated with the usual Precision@k ($k = 1, 5, 10$), Recall, and NDCG measures. We also report the interpolated precision at a certain recall cutoff r, iP_r with $r = 0.1$ and $r = 0.5$, the Mean Reciprocal Rank MRR and the Precision after R documents have been retrieved, where R is the total number of relevant entities for the document ($RPrec$).

We remind that in this evaluation we want to evaluate the number of correctly annotated entities *for a given document*; the evaluation is not spot-based, but we are rather considering the entity linking process as a whole, and its goodness on the full document.

The test dataset adopted is the same as the one of previous experiment, meaning that there is no overlap among the documents used for training the function ρ, and the documents used to evaluate its impact on the entity annotation process.

Table 4.4 reports the performance of the three annotators: for each annotator we show the performance using the original ρ^{MW} relatedness function, and then the effects of replacing the relatedness function with our learned relatedness $\rho^{\lambda\mathsf{MART}}$ and ρ^{GBRT}. The performance improvement given by the trained functions is significant:

Referent Graph. The proposed functions improve the ranking of results, in particular if we annotate only one entity per document using the ρ^{MW} relatedness only the 59% is correctly annotated, while with ρ^{GBRT} the percentage of correct documents is 74%. The relatedness function also reinforces the correct entities, improving the final ranking on the top entities as showed by the $NDCG$ measure which exhibits from a 14% up to a 25% of performance gain;

WikiMiner. ρ^{GBRT} improves both recall and $NDCG$, with gains superior to 10%. In both ρ^{GBRT} and $\rho^{\lambda\mathsf{MART}}$ the entity annotated with the largest confidence is correct in more than the 80% of the documents, with a improvement of 6% (ρ^{GBRT}) and of 10% ($\rho^{\lambda\mathsf{MART}}$) with respect to ρ^{MW};

TAGME. Recall, $NDCG$, and precision exhibit a positive improvement (from 1% up to 4%). The reader will note that ρ^{GBRT} and $\rho^{\lambda\mathsf{MART}}$ does not improve TAGME in the same measure as the other annotators: this is not surprising because the TAGME annotator is designed to manage short texts, and relies less on the relatedness and more on the commonness.

In general, the best result quality was obtained using the ρ^{GBRT} function.

4.6 Summary

In this Chapter, we have proposed a machine learning based approach aimed at discovering the entity relatedness function that can better support the entity linking task. We illustrated some of the properties that such function should preserve, and we presented a simple method to generate a training set form a collection of document human assessed entity linked documents. We casted the problem of discovering a suitable entity relatedness function into a learning to rank formulation. Our proposed approach is thus able to learn how to wisely blend the available features to generate a good entity relatedness function. We demonstrated that by exploiting our framework it is possible to better estimate the relatedness of two entities, and to compare and improve the performance of different state-of-the-art entity linking algorithms.

Chapter 5

On Generating News Explanations

5.1 Introduction

The large amount of research work done in the last decade in the area of recommender systems was mainly motivated by the great success this kind of systems have, and are still gaining, in real-life Web applications. Amazon is usually credited among the first ones to exploit the potential of these tools to enhance user engagement. Recommender systems, however, are becoming increasingly popular in diverse application domains. Originally used for products, recommender systems are now popular also for other types of data, such as music, videos, queries, friends on social networks, news items, among others.

In this Chapter, we focus on news recommendations: due to the amount of news items available, online news services deploy recommender systems that help the users find potentially interesting news items. To the best of our knowledge, these recommendations are displayed with a very shallow explanation of why a given news item has been suggested for reading (e.g., a snippet of text from the news item, or the number of views of the news item). However, even in the case that a relevant news item has been recommended for a user, the action of accessing the item will largely depend on how accurately the interestingness is assessed by the user *before* clicking on the item. Not generating an informative explanation might downgrade the performance of recommender systems, their applicability and, consequently, their value for monetization.

Explaining news recommendation is the goal *at-large* of the research presented in this Chapter. In particular, we aim at enhancing the users' experience on news platforms, such as Google News, Yahoo! News, and the alike by motivating the recommended news items shown by means of a tool that automatically generates brief, yet significant, explanations. Even though explaining recommendations is a topic covered by the literature on recommender systems [101, 25, 67, 152], researchers have mainly focused on evaluating effectiveness and the impact of explanations on increasing the number of clicks (and, possibly, eventual purchases) from users. Choosing the right explanation models to be used by a news recommender system is one of the challenges we face in this research. Notably, these explanations do not aim at increasing the click-through ratio of recommendations. Rather, they try to help a user to realize whether a news item can be of her interest or not, by providing her with additional information on the recommendations being proposed. The ultimate goal is thus to inspire the trust and increase the engagement of users with news platforms, by presenting a correct explanation on *why* the recommendation has been shown.

Given a pair of news items, the one actually read in the past by a user and the one suggested by the news recommender we generate a set of possible explanations for the recommendation itself. We study the following types of explanations: *text-based* explanations, which use similarity measures between the source and recommended news items, *entity-based* which employ information of named entities automatically detected in both news items, and *usage-based* explanations, which are related to how users reached and consumed the news items. It is important to remark that in this research we consider the news recommender system as a *black-box* to foster the flexibility of the solution and the independence from the underlining recommendation model.

Further, we developed a machine learning model out of a set of editorially annotated explanations of the different types. We build a Markov Logic Network [122] which is able to learn probabilities out of handcrafted rules expressed in first order logic. All the possible dependencies among features and explanations are taken into account seamlessly in the model generation phase. Our solution is developed to be flexible with respect to the associated recommender system and for not disclosing any information about the recommendation method adopted by the system enhanced.

We experiment our techniques on a dataset we specially build for this purpose. We exploit the browsing traces contained in toolbar data of a popular search service to extract pairs of news consulted consecutively by a user. For each pair we make the assumption that the user has been recommended the more recently read news by an oracle knowing that she read the first news item in the pair. We ask editors to evaluate how explicative each explanation is, and this editorial dataset is then used to learn the model. The results we obtain show that the explanations automatically generated are relevant and useful in order to learn the relationship between a news item and its associated recommended news item. Furthermore, the model is able to rank different types of recommendations for different news items, using the characteristics of both of them.

The Chapter is structured as follows. In Section 5.3 we present a brief survey of the state of the art. Section 5.2 introduces the news recommendation problem and the notation used. Section 5.4 presents the techniques devised to automatically generate *text*-based, *entity*-based, and *usage*-based explanations. In Section 5.5 we present the learning problem and a detailed description of the methods used to select the most suitable explanation to a given news item, as well as the features the model accounts for. In Section 5.6 we present experiments and the main findings. Finally, Section 5.7 presents some conclusions.

5.2 Ranking Explanations for News Recommendations

News systems usually exploit some recommendation algorithm able to suggest one or more interesting news items being related to the news item a user is currently reading. Explaining news recommendations consists in building and ranking a predefined list of explanation templates.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of news items, and let $E = \{e_1, e_2, \ldots, e_m\}$ the set of possible explanations. Given a *source news item* i_s, i.e., the news item a user was reading, and a *target news item* i_t, i.e., the recommended news item, we aim at ordering the elements in E according to their *relevance* or *elucidatory value*. The ordering shall reflect how good the explanation is in helping a user that read i_s in the past to take the decision of reading or not news i_t. The idea is that the first explanation in the ordering has to be the most *explicative* for the user.

The problem formulation is generic and there are many possible ways to tackle this problem. In this paper we take two alternative paths:

- *Static ordering* is a straightforward method that does not require any effort at recommendation time. In essence, the ordering of explanations is decided once and for all, and it is never changed. However, one could change the number of explanations combined to present the final, overall, statement.

- *Machine-learning based ordering* In this setting we map the recommendation explanation problem into a machine learning framework. In particular, we represent explanations and recommendations in a common input feature space (which could include user and context data) and learn a ranking function into a structured output space [49].

Given a pair $(i_s, i_t) \in \mathcal{I}$, with $\mathcal{I} = (I \times I)$, such that at least a user has been suggested a news item i_t after reading i_s, we denote with $Y = \{y_1, \dots, y_{|E|}\}$, with $y_j \in E$, a ordering of E where the explanations are sorted in decreasing order of relevance. If Y^{st} denotes the output for (i_s, i_t), then y_1^{st} is the best explanation for i_t given i_s. The problem is to learn a function

$$f : \mathcal{I} \to Y, \tag{5.1}$$

over a class of functions H, such that it minimizes a loss function $\Delta(Y, \hat{Y})$ measuring the penalty for making a prediction Y if the correct output is $\hat{Y}$. If $P(\mathcal{I}, Y)$ denotes the data generation distribution, then the goal is to minimize the risk [150]:

$$f = \arg\min_{f \in H} \int \Delta(f(i_s, i_t), Y^{st}) \; dP(\mathcal{I}, \mathcal{Y}) \tag{5.2}$$

Rather than evaluating the whole news items pairs space $\mathcal{I}$, we restrict to minimizing the empirical error on a training set T:

$$f = \arg\min_{f \in H} \sum_{(i_s, i_t) \in T} \Delta(f(i_s, i_t), Y^{st}) \tag{5.3}$$

Our goal is to find such f that, for a given pair of news items, is able to predict the optimal ranking of the explanations in the set E.

As usual in machine learning methodology, at recommendation time the learned function f is applied to the features extracted on-the-fly from the news items and explanations are ranked accordingly.

Note that the problem formulation does not make any assumptions on what algorithms are used by the news recommender system. This is a carefully informed choice as we aim at building a system that is oblivious with respect to the recommendation algorithm. This assumption has two major benefits. The first one is that we can take our explanation method and adapt it to any news site using any recommendation method and, being decoupled, any update on the recommender system would not imply an update on the explanation system. Secondly, we do not disclose any information about the recommendation algorithm that is usually a, well-kept, trade secret.

5.3 Explanations

Most of the research in the last years on explanations has focused on evaluating effectiveness and impact on users of existing explanations techniques. For instance, Herlocker *et al.* [67] propose a taxonomy categorizing different approaches for explaining recommendations over three dimensions: (i) reasoning model: white box vs. black box explanations; (ii) recommendation paradigm: content-base vs. collaborative filtering; and (iii) exploited information categories: user modeling, recommended items. The first dimension is the most interesting to us. The authors support the choice of a black box model where explanations are produced independently of the recommendation algorithm, rather than a white box approach, where the explanations are generated on the basis of the complex rules and techniques exploited by the algorithm, thus being too complex to help the user in understanding the recommendations. On the same line, Vig *et al.* [152] discuss the difference between transparency and justification: the former allows the disclosure of how the system really works, while the latter can be decoupled from the recommendation algorithm. Often justifications are preferred because i) the real algorithm is difficult to explain (e.g., matrix factorizations), ii) the algorithm is secret, iii) more flexible design of the explanations. For these reasons, in this work we considered the recommender system as a black box.

In [25] the authors discuss the trade-off between satisfaction and promotion. In the first case the goal of the explanation is to improve the user experience, while in the second the goal is to persuade a user to adopt an item. With a preference towards user satisfaction, the authors propose a novel way to evaluate the goodness of an explanation by asking the user to rate a recommended item first on the basis of the explanation only, and then after having inspected the item. We partially borrowed this idea in labeling of the dataset, by allowing the user to view the title or the full content of the recommended news.

In their works Tintarev and Masthoff [141, 142] describe most of the research done in the area. They review several recommendation systems and algorithms and they categorize them according to several criteria. The first is the way recommendations and their explanations are presented: (i) top items, (ii) similar to top item, (iii) predicted ratings for all items, and (iii) structured overview. A second important criterion is the kind of benefit expected by the explanations: (i) *transparency* explains how the system works; *scrutability* is the capability that allows users to tell whether the system it is wrong; *trust* to increase users confidence in the system; *effectiveness* to help users to take good decisions; *persuasiveness* the ability of the system to convince users to try or buy; *efficiency* the amount of time saved by the user in taking decisions; and *satisfaction* the ability of increasing the ease-of-use or enjoyment by the user. It is hard to create explanations that perform well on all criteria. In this work we asked the assessors to rate mostly on the basis of *effectiveness*, but also considering *satisfaction* and *efficiency*, but the framework can be adapted to optimize other goals if a proper labeling of the data is provided.

Interestingly, Ahn *et al.* [4] present a personalized news system that, aiming at maximum transparency, allows the users to edit their interest profile, affecting the provided recommendations. The user is modeled as a bag of words, extracted from his browsing history, that can be modified at any time by adding or removing words. However, the capability of editing the user profiles is not always beneficial, in particular they observed that the more changes are done, the more harm is done to the system.

Text snippets are used in Web search to explain why results are relevant for a given query. Thus, creating high quality snippets has been a recent topic of interest. Kanungo and Orr [84] address the problem of generating query-biased summaries that maximize readability. Traditional Web search snippets are made up of document titles, text and urls, and confined to a particular bounding box in the page layout. However, users can assess the relevance of search results if these snippets contain other visual clues, like multimedia objects, or elements that allow to interact with the contents of the results page directly [64]. Finally, the problem of retrieving explanations has also been studied for entity search [28].

5.4 Generation of explanations

We generate for each pair of read and recommended news items (i_s, i_t), a small set of explanations going beyond showing only the title and a short abstract of the news item and with the goal of letting the user understand why the news item has been recommended to him and consequently increasing the awareness on that. Therefore, one of the first problems we deal with is to select what kind of explanations we have to generate and how. We adopted a very pragmatic approach and we come up with a set of 16 different explanations, grouped in three different classes according to the feature type we use to generate them. In particular we distinguish between *text*-based, *entity*-based, and *usage*-based explanations. In the following we report the list of the 16 different kinds of explanations grouped by their class.

5.4.1 Text-based explanations

The explanations in this class have the common characteristic to be generated by exploiting the textual content in either the read or recommended news items.

SIMILARITY *The recommended news item is similar to the one you are reading.* This explanation considers the similarity between the content of the source and target news items. Similarity is computed by using cosine similarity with TF-IDF weighting, where the inverse document frequencies are defined over a large collection of news items. The SIMILARITY explanation is triggered by a similarity score greater than a predefined threshold τ, fixed to 0.4 in all our experiments. The motivation for this explanation template is the following. News items are recommended, generally, by showing the title and the first sentence. This presentation methods might not be enough for the user and if we also add the information about how high the similarity is between two news items then we could indeed inform the reader about one of the possible reasons why she might be interested in the news item.

SIMILARITYSNIPPET The explanation consists in providing a snippet containing the two most similar sentences extracted from the read and recommended news items. Also in this case the cosine similarity is used. The idea for this explanation is very similar to the previous one. Indeed, the two sentences added to the motivation play the same role as the snippets (i.e., query-biased summaries) in search engine results.

TARGETSNIPPET This explanation is constructed by simply taking the first two sentences appearing in the recommended news item. The idea, here, is to increase the amount of content provided to the user. It is equivalent to increasing the space devoted to the news recommendation to fit the first two sentences. We include this template to evaluate the effect of putting more text in addition to the title of the news item.

TAGSPLANATION [152] *Both the read and recommended news items share tags X.* The explanation exploits the common tags X associated with the read and recommended news items, if any. Tags are generated by taking the 15 terms with the highest TF-IDF score from each news item.

5.4.2 Entity-based explanations

In this class fall explanations using entities, or images associated with them, as a means to convey the reason why a news item has been recommended to a user. Named entities are recognized and extracted with the SuperSense Tagger.[1]

SHAREDENTITY *The recommended news item is interesting because it tells about X as the one you are reading.* The explanation tells to the user that a given named entity X is shared between the read and recommended news items. We conjecture a user might be interested by some particular

[1]http://sourceforge.net/projects/supersensetag

fact about an entity (e.g. a person, a movie, etc.) since he was reading about that entity in the news item accessed.

TargetEntity *The recommended news item is about X.* The explanation presents to the user the named entity X extracted from the recommended news item if it does not appear in the read news item. The idea is to give the user the information that might be more important to him, i.e. the entity X of which the news item discusses.

DistinctEntities *The news item you are reading and the recommended one are about X and Y.* The explanation presents the two main distinct entities extracted from the read and recommended news items. The idea is similar to the two previous types but we aim at increasing the informativeness and capturing news diversity by adding one entity to the explanation.

TargetImage *The recommended news item is about X.* The explanation shows an image X associated with the main named entity represented in the recommended news item. The image shown is taken from *Freebase*, using the provided API[2]. The idea is that of “one picture is worth thousand words” and we aim at increasing user awareness on the explanation by adding the image depicting the main subject of the news item.

Images *The read and recommended news items are about X and Y, respectively.* The explanation shows two images X and Y, associated with entities represented in the read and recommended news items, respectively. If $X = Y$, i.e., the news item refers to the same named entity, the image is obviously shown only once. The purpose of this kind of recommendation is the same as SharedEntity or DistinctEntities but with a focus on images rather than textual description of the entities involved.

SharedPlaces *Both the read and recommended news items refer to the places X, Y, ...* The explanation proposes the geo-names shared between the source and target news items, if any. The purpose is to let the reader understand the localization of the news item she is reading and that she has received as a recommendation.

TargetPlaces *The recommended news item refer to X, Y, ...* This explanation suggests that the recommended news item refers to some geo-names. The idea, here, is the same as SharedPlaces but we let the

[2] http://wiki.freebase.com/wiki/Freebase_API

Source News:

NFL star Junior Seau remembered in surfing ceremony

OCEANSIDE, California (Reuters) - Hundreds of surfers paid their respects to Junior Seau on Sunday with a "paddle-out" ceremony in the Pacific, just beyond the beachfront home where the football great and avid surfer committed suicide at age 43 ...

Target News:

Football great Junior Seau's brain to be examined

LOS ANGELES (Reuters) - Football great Junior Seau's brain will be examined for evidence of repetitive injuries from his playing days following the retired linebacker's suicide in his California beachfront home ...

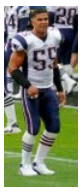
QUERIES	The recommended news item is about: **junior**, **seau**
SHAREDENTITY	The recommended news item is interesting because it tells about **Junior Seau**, **San Diego Chargers** as the one you are reading
SHAREDPLACES	Both the read and recommended news item refer to the place: **Oceanside**, **CA**, **US**, **San Diego**, **CA**, **US**
IMAGES	The read and recommended news items are about **Junior Seau** 

Figure 5.1: An example of explanations for sport-related news.

user think by herself of a possible relationship with the news item she is reading.

CATEGORIES *Both the read and recommended news items are categorizable as X, Y, ...* The explanation exploits the common categories (e.g., sport, politics, etc.) associated with the read and recommended news item, if any. Categories are provided by the news provider. The idea is to explain the user that the interest in this news item can be due to the categories it belongs.

5.4.3 Usage-based explanations

Explanations of this class are extracted by considering how news items are accessed by users. We extract knowledge from toolbar data to explain with some form of global statistics the recommendations produced.

POPULARITY *The recommended news item is interesting because N users have recently read it.* The explanation highlights the popularity of the recommended news item. This explanation is oblivious with respect to the read news items and has the purpose of telling the user that the recommended news item is important among users of the news platform.

Source News:

Conservative factions dominate Iran's run-off elections
DUBAI (Reuters) - Iranian President Mahmoud Ahmadinejad, now out of favor with Supreme Leader Ayatollah Ali Khamenei, suffered more setbacks in a run-off parliamentary election seen as a pointer for next year's presidential race...

Target News:

Heavy fighting rocks eastern Syria: residents
AMMAN (Reuters) - Heavy fighting between rebels and government troops erupted overnight in the capital of an oil producing province in eastern Syria, residents and activists said on Sunday, the latest escalation of violence in a tribal area bordering Iraq

DistinctEntities	The news items you are reading and the recommended one are about **Iranian President Mahmoud Ahmadinejad** and **government troops**
TargetSnippet	AMMAN (Reuters) - Heavy fighting between rebels and government troops erupted overnight in the capital of an oil producing province in eastern Syria
SimilaritySnippet	(Reporting by Khaled Yacoub Oweis; Editing by Andrew Osborn) ..."The fighting subsided early in the morning...
TargetPlaces	The recommended news items refer to: **Rif Dimashq**, **Syria**

Figure 5.2: An example of explanations for geo-political news.

Queries *The recommended news item is about S.* The explanation is given by the set S of terms shared among the past queries for which the recommended news item was retrieved and clicked. The motivation for this explanation is that we want to give users a view-point consisting of the terms other users were using to reach that news item.

TargetQueryBiasedSnippet The explanation consists in providing a query-biased snippet X for the recommended news item. The snippet is extracted from the news item by using the *openNLP*[3] framework and the terms shared among the queries for which the news item was clicked.

SourceQueryBiasedSnippet Similar to TargetQueryBiasedSnippet except for the fact that the terms used are those of the queries associated with the read news item and not the recommended one.

Please observe that for each pair of news items, it is not always possible to produce all the 16 explanations. For instance, the explanation Images can be produced only if we are able to retrieve the image entities for both the source and the target news items. Finally, note that snippets generated for explanations contain two sentences and do not exceed 200 characters.

[3]http://opennlp.apache.org

To exemplify some of the explanations generated, Figures 5.1, and 5.2 contain two examples of source and target news items pair (i_s, i_t) along with their explanations.

In the first example, the news items are both about Junior Seau, a linebacker in the National Football League who died in May 2012. In the figure, we present several types of explanations. All the explanations can be considered good, with the only exception of the one based on the places. We noted that the explanations based on places are rarely useful, but there are some situations in which they are very important. In the second example, where the target news item is about the fighting between rebels and government in Syria, places are more relevant. Even if Syria is far from Iran to which the source news item refers to, the news items are geo-politically related and we can rate *good* this kind of explanation. We also point out that there are explanations that are really not useful, for example the explanation produced with the SimilaritySnippet method, that does not contain any information about the relation between the news item and also does not say anything about the target news item.

5.5 Modeling and Learning

This section introduces the main features of news items and recommendations that are being engineered in the system, and a briefly surveys the learning model employed to rank explanations.

5.5.1 Features Used

Given a quadruple (i_s, i_t, e_u, y) composed by source and target news items, one of the explanations discussed in Section 5.4, and the results of the evaluations performed by our assessors, we extract the set of features described in the list below. The features are used in the learning task and exploit both the entities and the geo-names extracted from both the source and the target news item.

COS the cosine similarity $cosine(i_s, i_t)$ between the bag-of-word representations of both the read and the recommended news items;

E1 a binary value stating the presence or absence in the target news item i_t of entities. This feature is 1 when there exists an entity in the text of i_t;

E2 a binary value stating if ($\mathbf{E2} = 1$) target and source news items share some common entity;

E3 a binary value stating if (**E3** = 1) target and source news items share the main entity, i.e. the entity appearing most frequently within the news item;

SE1 the number of terms shared between the snippet in e_u (if any) and the labels of the entities recognized in the source news item i_s;

SE2 the number of terms shared between the snippet in e_u (if any) and the labels of the entities recognized in the target news item i_t;

ST1 the number of terms shared between the snippet in e_u (if any) and the title of the source news item i_s;

ST2 the number of terms shared between the snippet in e_u (if any) and the title of the target news item i_t;

SG1 the number of terms shared between the snippet in e_u (if any) and the geo-names in the source news item i_s;

SG2 the number of terms shared between the snippet in e_u (if any) and the geo-names in the target news item i_t.

Note that features SE1, SE2, ST1, ST2, SG1, SG2, are only generated for explanations based on the snippets, i.e., SimilaritySnippet, TargetSnippet, TargetQueryBiasedSnippet and SourceQueryBiasedSnippet.

5.5.2 Learning relevant explanations with Markov Logic Networks

Given the variety of features embodied in the model, we would like to build a class of functions f that are able to account for structured dependencies in input features. We make use of the Markov Logic Networks (MLNs) of [122], a joint model that combines first-order logic (FOL) and Markov networks. The model is able to capture contextual information and long-range dependencies between features, which are critical to the task addressed here.

Markov Logic Netwoks have been used elsewhere for a plethora of natural language and data mining applications, from entity recognition [130], disambiguation [51], co-reference resolution [71], extracting predicate-argument relations [160] or even mobile robot map building [154]. A key advantage of MLNs is that they allow to express semantically-rich formulas to capture a variety of long-term dependencies between features in a seamless fashion. In essence, they act as an interface layer between the learning process and the domain knowledge engineering.

Formally, MLNs are made up of two different components combined to perform joint inference.

Markov Networks. Given a set of random variables $X = \{X_1, X_2, \ldots, X_n\}$ a graphical model represents its joint distribution $\mathcal{X}$ as a product of non-negative potential functions $P(X = x) = \frac{1}{Z}\Pi_k \phi_k(x_{\{k\}})$. Each potential ϕ_k is defined over a sub-set of the variables $x_{\{k\}}$ and Z is a partition function $Z = \sum_x \Pi_k \phi_k(x_{\{k\}})$. A Markov Network or Markov Random Field is defined by an undirected graph G, which contains a node for each variable, and the model has a potential function for each clique in the graph. Provided that $P(X = x) > 0 \forall x$, the distribution can be equivalently represented as a log-linear model:

$$P(X = x) = \frac{1}{Z}\exp(\sum_i w_i f_i(x)) \;, \tag{5.4}$$

where the $f_i(x)$ are arbitrary *feature functions* of a sub-set of the variable's state. The most straightforward translation from Equation 5.4 is to assign one feature corresponding to each possible state $x_{\{k\}}$ of each clique with its weight being $\log \phi_k(x_{\{k\}})$. Even though this representation leads to an exponential number of functions in the size of the cliques, one is free to specify a much smaller number of features, like logical functions of the state of the clique which lead to more compact representations.

Given the observed values of some variables, probabilistic models aim at finding the most probable joint state of the unobserved variables (inference), and computing conditional probabilities of unobserved variables (conditional inference). Both of these problems are #P-complete and applications usually resort to approximations such as Markov Chain Monte Carlo (MCMC), most notably Gibbs sampling [62].

First-order logic. A first-order knowledge base (KB) is a set of sentences or formulas in first-order logic [60]. First-order Logic (FOL) formulas are composed of four types of symbols, namely constants, variables, functions and predicates. Constants represent objects in a domain of discourse (for instance, *words, explanations*). Variables range over the objects (*x,y*). Function symbols represent a mapping from tuples of objects to objects (*positionOf*). Predicates represent some relation between objects (*hasEntities*) or attributes of objects (*hasClick*). Variables and constants may be typed, in which case variables only range over objects of the given type (for instance, words and entities).

Furthermore, an atom is a predicate symbol applied to a list of arguments, which may be variables or constants (e.g. *hasWords(Obama, explanation1)*. If the arguments are all constants, this is called a *ground atom* (*hasClick(page1)*). Finally, a world is an assignment of truth values to all possible ground atoms. A KB is a partial specification of a world in which each atom in it is true, false or unknown. First-order logic allows for representing compactly complex relational structure. A central problem in logic is that of determining if a KB is satisfiable, i.e., if there is an assignment of truth values to ground atoms that makes the KB true [125].

Markov Logic Networks. A MLN L is a set of pairs (F_i, w_i) , where F_i is a FOL and w_i is a weight assigned to the formula. The interest of the MLN is that it can be viewed as a template for constructing Markov networks. Given a set of constants C, a Markov Network $M_{L,C}$ is defined by:

- $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.

- $M_{L,C}$ contains one feature for each possible grounding of each formula $F_i \in L$. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with $F_i \in L$.

This set-up allows for worlds that violate some constraints, which is not the case in first-order logic, in which formulas are hard constraints; violating a single formula has the same effect as violating all. When a world violates one formula of the MLN it becomes less probable, but not impossible. The MLN can be regarded as template for generating Markov Networks in the following sense. Given different sets of constants, a MLN will produce different networks, which might have different size but they will share regularities and parameters, which are given by the MLN; for instance, all the groundings of the same formula will have the same weight.

The probability distribution over the possible worlds is given by

$$P(X = x) = \frac{1}{Z}\text{exp}\sum_{i}(w_i n_i(x)) = \frac{1}{Z}\Pi_i \phi_i(x_{\{i\}})^{n_i(x)} \ , \tag{5.5}$$

where $n_i(x)$ is the number of true groundings of F_i in x, and $\phi_i(x_{\{i\}}) = e^{w_i}$. MLNs have further interest in that most common probabilistic models can be succinctly formulated as MLNs, including HMMs, CRFs, logistic regression, Bayesian networks, and so on.

Markov Logic Networks rules

We make use of *Alchemy*,[4] a software toolkit that provides a series of algorithms for statistical relational learning and probabilistic logic inference, based on the Markov Logic Networks representation.

[4]for more details about the framework and the syntax, please refer to `http://alchemy.cs.washington.edu/tutorial/tutorial.pdf`

In the following we illustrate some of the logic formulas that we used to model the problem and to train the learning model. The domain developed includes a number of ground terms, for instance:

```
Relevant(rec, expl)
Rank(rec, expl, rank)
HasExpl(rec,expl)
Similar(rec, similarity!)
Related(rec)
ShareEntities(rec)
TargetHasEntities(rec)
HasSourceEntityTerms(rec,expl)
```

We are interested in learning the probability that an explanation is relevant for a particular pair of news items (i.e., `Relevant`), and the probability that an explanation will have a certain rank for a recommendation (`Rank`). There are several features we can exploit, for example: the possibility to produce a certain explanation for the given pair of news items (`HasExpl`), the cosine similarity between the news items (`Similar`, the `!` near `similarity` means that each recommendation must be associated with exactly one similarity score), if the news items are related or not (`Related`), if they share some entities (`ShareEntities`), etc. As a rule of thumb, each signal described in Section 5.4 will have a predicate to incorporate it into the model.

The system is described by means of rules in first-order logic; the following two rules that define two straightforward facts:

```
!HasExpl(r,e) => !Relevant(r,e).
HasExpl(r,+e) => Relevant(r,+e)
```

The former means that when an explanation `e` cannot be computed for a pair `r`, then the explanation is not relevant for `r` (the dot at the end of the rule specifies an *hard constraint*), the latter denotes a set (the plus "+" symbol) of rules (it instantiates the rule for each type of explanation e, e.g. `Relevant(r,+e)` becomes `Relevant(r,`SimilaritySnippet `)`, `Relevant(r,`DistinctEntities `)` ...) . The model learns, for each instantiated rule (e.g., `HasExpl(r,IMAGES) => Relevant(r,IMAGES)`), how much a particular explanation type is relevant for an explanation.

Given the expressiveness of FOL, deriving rules is straightforward to produce. To reach a reasonably performing system, we devise several other rules involving different types of features, like:

```
ShareMainEntity(r) => Relevant(r,+e)
TargetHasEntities(r) => Relevant(r,+e)
Related(r) => Relevant(r,+e)
```

The MLN will learn a weight for each rule, out of a training file with a set of ground atoms. Then, given a set of test instances, the system performs inference and estimates the rank of a given set of explanations The weight for each rule is learned by the MLN, from a training file consisting of a set of ground atoms, one per line (the symbol `!` represents the logic negation). A fragment is shown in the following:

```
 Relevant(R1,SHARED_ENTITIES)
 Relevant(R2,TARGET_SNIPPET)
 Relevant(R10,DISTINCT_ENTITIES)
!Relevant(R20,IMAGES)
!HasExpl(R20,IMAGES)
```

At the end of the inference, MLN will estimate the probability that a given explanation is relevant. The following fragment of text shows an example of the output produced by Alchemy:

```
Relevant(R0,SHARED_ENTITIES) 0.59599
Relevant(R0,TARGET_SNIPPET) 0.636986
Relevant(R0,SHARED_PLACES) 0.0670433
Relevant(R0,TARGET_IMAGE) 0.225027
```

The number on the righthand side is the probability computed, for the sake of our purpose we consider relevant only those explanations having a probability strictly greater than 0.5. Finally, we associate to each relevant explanation the most probable rank given by the predicate `Rank`; i.e., we start considering the list of relevant explanations and we put in the first position the most probable explanation e for rank 1, that we remove e from the list and we select the most probable explanation for rank 2, and so on.

5.6 Experiments

This section firstly introduces the dataset created for experimentation. Given the novelty of the automatic explanation of news recommendation problem we compiled a dataset of news items and associated recommendations and evaluated manually the goodness of each explanation. Next, we compare on the basis of well-established metrics three different methods: a SVM-based baseline, a static, i.e., frequency-based, method, and an MLN-based model.

5.6.1 Dataset

Since there is no benchmark dataset for the problem at hand, we created an evaluation dataset as follows. The dataset contains quadruples in the form

(i_s, i_t, e, r), this is, a pair of source and target news items, an explanation e and a score r for e. This allows to evaluate the goodness of a given algorithm at predicting r, or to derive the ranked list of explanations Y^{st} and measure the ranking performance as formalized in Eq. 5.3, using standard information retrieval metrics.

The first step in creating the dataset consists in obtaining valid news items pairs. Each pair should consist of the news item currently read and a recommended news item. To avoid overfitting our dataset with recommendations generated using a particular system we resort to exploiting the information recorded by a popular search engine toolbar. The toolbar service logs the pages visited by the users over time. We filtered only the pages from the Yahoo! news portal producing for each user a chronologically ordered sequence of news items. The hypothesis is that a perfect news recommender system should be able to recommend for a given news item the immediately successive read news item. On the basis of this assumption, we consider a pair of consecutive news items in the logs as an evidence that some recommender system suggested the second news item on the basis of the first one, and that the user actually read both of them.

The dataset we are considering is made up of 121 distinct news items forming 120 news pairs. The dwell time on the news pair is about 7 minutes on average, meaning that both the news items were likely to be of interest to the user. The pairs were selected randomly from the daily activity of over $100K$ users. For each pair we generated the explanations as described in Section 5.4.

The perceived effectiveness of each explanation is assessed by human editors. For each pair (i_s, i_t) we showed in the same web page the full content of the news item i_s and the title of the recommended news item i_t. However, we give the possibility to the evaluator to expand i_t, if needed, to read its full content. We also showed all the explanations produced and we asked to rate each explanation with a mark between 0 and 5: don't know (0), bad (1), fair (2), good (3), excellent (4), perfect (5). Explanations are not labelled with their types, and they are randomly permuted for each assessment, to avoid bias. Figure 5.1 shows some of the explanations provided to the assessors for the news items; the news items are topically related, since they share the same topic, i.e., the death of the football player *Junior Seau*.

Assessors scored the explanations according to their capability at helping the user in deciding whether or not the target news item is worth of being read. This means that the explanation should make the relationship between news items i_s and i_t clear. Nonetheless, it might happen that there is not a clear relation between the pair of news items. This is very typical when the user is reading the news of the day and switching between a variety of topics. In this case, we asked the reviewers to rank higher the explanations that better

Explanations Evaluation Dataset	
Number of Evaluators	5
Number of Evaluations quadruples (i_s, i_t, e, r)	$1,632$
Distinct Pairs (i_s, i_t)	120
Avg. Explanations per Recommendation	8
Distinct i_s	29
Distinct i_t	98
Related pairs (i_s, i_t)	36
Not Related pairs (i_s, i_t)	84
Avg. Pairwise Cohen's κ (scores)	0.42
Avg. Pairwise % agreement (scores)	68%
Avg. Pairwise Cohen's κ (relatedness)	0.65
Avg. Pairwise % agreement (relatedness)	83%

Table 5.1: Dataset and Evaluation statistics

help the user in understanding that the news items are not content related. In both cases, we believe that explanations help users to understand and exploit news recommendations.

The dataset is divided in 6 bins, each one containing 20 news pairs, and we asked five human editors to evaluate them. In order to measure the agreement between the evaluators, the first bin was scored by every assessor. The remaining 5 bins were scored by one assessor each.

Table 5.1 reports the statistics on the dataset obtained. On average 8 different explanations could be produced for each news item pair, resulting in a total of $1,632$ evaluations, due to the overlapping evaluations on the first 20 news item pairs.

For evaluating the inter-annotator agreement, we considered the set of pairs with an evaluation by all the annotators and we collapsed the scores in three categories:

Positive containing the judgments that express a user's opinion about the effectiveness of an explanation, i.e., *good* ($score = 3$), *excellent* ($score = 4$), *perfect* ($score = 5$);

Negative the judgements revealing the user's disappointment, i.e., *fair* ($score = 2$), *bad* ($score = 1$);

Neutral if the user is indifferent to the explanation (*don't know* ($score = 0$)).

We want to check if different users agree on the fact that, given a recommended news, a particular explanation is useful or not. As showed in Table 5.1, the average pairwise Cohen's kappa between the evaluators is $\kappa = 0.42$, that is proved to be [90] a significant agreement. Since inter-agreement score is reasonable, we do not take into account the impact of personalization of explanations. In the presence of a lower inter-annotator agreement, instead, a careful thought on the impact of personal tastes on the explanation would have been in order. In this case, we can conclude that personalization of explanation would not make a significant difference when evaluating the effectiveness of our techniques. Furthermore, Tintarev and Masthoff [142] found that personalization does not improve effectiveness and may even get worse results.

Evaluators also marked if the current and the recommended news were related by content, and we measured a pairwise agreement $\kappa = 0.65$. This means that different users have often the same perception of the relatedness of two news items, but on the other side, there are pairs that are related for some users and not related for others (see for example the news shown in Figure 5.2). Therefore, explanations could add a significant improvement in helping users to decide if the recommended news item is related or not. For this reason, if an evaluator marks a recommendation as not related, we ask him to score the explanations based on how good they are in helping to detect the unrelatedness. Finally, for each news pair we extracted the features described in Section 5.5.1.

5.6.2 Effectiveness of Explanations

We assess in this section the effectiveness of the following explanation strategies:

Popular explanation first [POP] For each recommended item we rank explanations only on the basis of their popularity and relevance in the training set, placing the most popular explanation first.

RankSVM-Based [SVM] We generate a SVM-based model using the rankSVM library from Cornell [81]. We set the regularization parameter c to 20, employed the L1-norm for regularizing the slack variables, and the 1-slack algorithm in the dual for learning.

MLN-Based [MLN] We generate a Markov Logic Network-based model using the methodology detailed in Section 5.5.2.

Results

We report the results of the different methods using traditional performance metrics such as: Precision@k, Recall@k, F-measure@k with cut-offs $k = 1, 2, 3$, Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain

	Measure	SVM	POP	MLN
	NDCG	0.666	0.7360	**0.8132**
	MAP	0.562	0.6469	**0.744**
@1	Precision	0.666	**0.7729**	0.7432
	Recall	0.202	**0.2325**	0.2178
@2	Precision	0.55	**0.7279**	0.7131
	Recall	0.32	**0.4296**	0.4196
@3	Precision	0.447	0.7135	**0.7353**
	Recall	0.372	0.6372	**0.6632**

Table 5.2: Average performance of the different explanation ranking strategies, computed using 10-fold cross-validation. In bold we highlight the best results.

(NDCG). We make the following two assumptions: (i) we consider an explanation relevant if and only if an evaluator marked it as good, excellent or perfect, (ii) in case of more than one judgement, we consider the average rate.

In particular, the choice of testing several cut-offs instead of just one, i.e., $k = 1$, and to evaluate NDCG and MAP is driven by the following motivation. Our method is aimed at presenting a selection of explanations that would help the news site users to better understand why one might be interested in reading the news item. Therefore, one of the goals of the evaluation is to show how more than one explanation is better for the reader.

Results achieved via 10-fold cross validation are presented in Table 5.2. They show that POP, and MLN have similar performance in terms of precision, recall, and F-measure for cut offs equal to 1 and 2, being the MLN superior in terms of MAP and NDCG. In terms of NDCG, POP is about 10% better than SVM, while MLN is about 21% better than SVM. In terms of MAP, POP outperforms by about 14% SVM, while MLN is about 32% better than SVM. We performed the Wilcoxon signed-rank test over each dataset, and we found that MAP and NDCG results are statistically significant with a *p-value* always lower than 5% (while the Wilcoxon test failed to reject the null hypothesis for precision and recall at $1, 2, 3$). Anyway, significance tests we run on the results (and that we are including in the paper) show that MLN and POP are always statistically significantly different from SVM. These results, also, show that a MLN approach is, in general, more suitable for ranking meaningful explanations in top positions. SVM underperforms compared to those two methods, mostly because the dataset only contains a few relevant explanations per item and the feature sparsity.

Table 5.3 shows for each type of explanation, the average score given by the

	AVG	STD DEV	MEDIAN	MODE	MAX	MIN
SharedEntity	**3.679**	1.090	3	3	5	2
DistinctEntities	**3.362**	1.178	4	4	5	0
Images	**3.296**	1.298	3	3	5	0
TargetImage	**2.946**	1.217	3	3	5	0
TargetEntity	**2.878**	1.024	3	3	5	0
TargetSnippet	**2.862**	0.977	3	3	5	0
SourceQueryBiasedSnippet	2.299	1.072	2	2	5	0
TargetQueryBiasedSnippet	2.278	1.143	2	2	5	0
SimilaritySnippet	1.867	0.980	2	1	5	0
Similarity	2.200	1.175	2	1	5	0
Tagsplanation	1.667	0.679	2	2	3	1
TargetPlaces	1.484	0.651	1	1	3	0
SharedPlaces	1.438	0.769	1	1	4	0
Popularity	1.288	0.667	1	1	4	0
Categories	1.262	0.691	1	1	4	0
Queries	1.221	0.595	1	1	4	1

Table 5.3: Statistics of the scores assigned to different types of explanations

evaluators. Clearly, evaluators show preference for some type of explanations, in particular the explanations based on the entities, the explanations based on the images, and the snippet containing the first sentence of the target news item. We also observe that MLN performs better than the other methods, especially when the system displays three different explanations. The reason behind these results is that MLN is able to learn the relations between the different explanations, and the weight for different features in function of the type of explanation and characteristics of the news items.

5.7 Summary

In this Chapter we investigated the novel problem of devising an effective way to automatically explain news recommendations to enhance user experience on online news platforms. First we selected the sources and created a dataset of news items and related recommendations from the data recorded by the browser toolbar of a popular Web search service. Given a pair of news items, the one actually read by a user and the one suggested by a black-box recommender, our two-steps goal was thus to generate a set of suitable explanations by exploiting pieces of information from content and context of the news items, and then to rank these explanations on the basis of their expected usefulness. Regarding the generation of candidate explanations, we engineered the methods for building automatically 16 different types of suitable explanations. The techniques devised exploit text-based, entity-based, and usage-based features. Most of the features used to describe news and recommendations rely on textual similarity between sentences or entities, but the most interesting are extracted or enriched by exploiting external knowledge bases such as *Freebase*, and browser toolbar data that record how the news items have been searched for and consumed. We pursued two distinct strategies to address the problem of ranking candidate explanations: (*i*) a static (explanation popularity-based) approach, and (ii) a machine learning approach. Specifically, for the machine learning approach we employed Markov Logic Networks for learning to rank the set of explanations generated.

The model was trained with a human-assessed dataset, where we asked a set of assessors to score the explanations generated for 120 pairs of news items. Interestingly, we found that assessors usually prefer *entity-based* explanations.

Experimental results showed that the method provides high-quality explanations, and it is able to outperform state-of-the-art structured learning to rank approaches, by a percentage ranging from 10% (POP strategy) to 21% (MLN strategy) in the case of NDCG measure. We also get similar figures in the case of MAP. All the results are for cut-off equal to 3, which is a number of explanation we argue to correspond to the right amount of explaining power we want to express.

Chapter 6

Conclusions

In this Chapter we briefly resume the research work conducted during the Ph.D. project, we recall the main contributions achieved, and discuss some possible developments that can build over our results. Finally we report the references to the papers (published and under revision) in which such contributions are discussed and assessed.

6.1 Thesis Contributions and Future Work

This Ph.D. dissertation focuses on exploiting query log data together with the Web Of Data to improve effectiveness and efficiency of WSEs.

We showed that query log data can be exploited to remarkably improve the performance of the document repository used for snippet extraction. Our technique speeds up the WSE by exploiting a novel caching strategy specifically designed for document snippets. Query log data allowed us to better understand the characteristics and the popularity distribution of URLs, documents and snippets returned by a WSE. We designed and experimented several cache organizations, and we introduced the concept of *supersnippet*. A supersnippet is the set of sentences of a document that are more likely to be retrieved for building the snippet of future queries. We showed that supersnippets can be built by exploiting query logs, and that a supersnippet cache can answer up to **62**% of the requests, remarkably outperforming the other state-of-the-art caching approaches experimented. To the best of our knowledge, this is the first work discussing a cache designed to relieve the load from document repository by exploiting the knowledge about the past queries submitted. For this reason, several research directions remain open. One of the most important open questions is related to how the document cache and the result cache interact with each other. What is the best combination between the two of them. What is the best placement option we can choose: is it better to keep them separated on different machines (thus allowing the exploitation of more aggregate memory), or is it better to keep them on the same machine in order to reduce network delays? How to manage dynamic updates of the cache? Another interesting question regards more strictly our supersnippet organization: preliminary tests showed that dynamic supersnippets as the ones discussed in this work outperform static ones. An open question is the evaluation of the cost/performance ratio between these two different organizations, and the analysis of possible aging effects over statically built supersnippets. One may also investigate the effect of combining text compression and supersnippetting to allow the cache to store more surrogates. Lastly, the possibility of building supersnippets on the basis of the entities detected in the documents could be investigated.

In Chapter 3 we performed a study of Europeana query log data, showing detailed statistics on common behavioral patterns of users of the Europeana portal. Our analysis highlights some significative differences between the Europeana query log and the historical data collected by general purpose WSE's logs. In particular, we found out that the distributions of both query popularity and search sessions are different. We enriched the user sessions recorded in a long-term query log with Wikipedia entities, and we explored the use of entities to enhance query recommendations. We extended a state-of-the-art recommendation algorithm in order to take into account the semantic information associated with submitted queries. Our novel method generates highly related and diversified suggestions that we assess by means of a new evaluation technique. The manually annotated dataset used for performance comparisons has been made available to the research community to favor the repeatability of experiments. This work opens many possible new directions for future research. Among them we cite: i) the study of new measures to define the quality of semantic query suggestions; ii) the refinement of our annotator to manage multilingualism; iii) the enlargement and improvement of the dataset, by also taking into consideration entity attributes.

Chapter 4 investigates *entity relatedness*, a measure estimating how much two entities possibly mentioned in the same text are related. A precise estimation on relatedness is very important for many entity-related tasks. For example, annotation algorithms enhance entity-linking precision by disambiguating entities in a way that maximizes the relatedness among the selected candidates. The definition of an effective relatedness function is thus a crucial point in any entity-linking algorithm. We addressed the problem of learning high-quality entity relatedness functions, by formalizing entity relatedness as a *learning-to-rank* problem. We proposed a methodology to create reference datasets on the basis of manually annotated data, and we showed that our machine-learned entity relatedness function performs better than other relatedness functions previously proposed. More importantly, we demonstrated that our function improves the overall performance of different state-of-the-art entity-linking algorithms. There are several promising directions for future work. For example studying the impact of new features (e.g., categories, text, user clicks) from both quality and performance perspectives, or analyzing if the importance of features is influenced by the domain where relatedness is applied e.g., entity suggestions, annotation of tweets vs annotation of web-pages. In order to perform our experiments we designed a framework for entity linking, and we implemented several state-of-the-art algorithms. We plan to share this framework with the scientific community in order to facilitate the definition and evaluation of entity-linking algorithms and the sharing of new datasets.

Last but not least, we considered the news domain, and we proposed to enhance the effectiveness of news recommender systems by adding to each recommendation an explanatory statement that aims at helping the user to better understand if, and why, the suggested item can be of her interest. We took into account the news recommender system as a black-box, and we generated different types of explanations employing pieces of information associated with the news. In particular, we engineered text-based, entity-based, and usage-based explanations, and made use of Markov Logic Networks to rank the explanations on the basis of their effectiveness. The assessment of the model was conducted via a user study on a dataset of news read consecutively by actual users. We showed that assessors usually prefer entity-based explanations, and that we are able to produce high quality combinations of explanations of news recommended to users. We conjectured that such an explanation system can improve recommender systems since it allows users to promptly discriminate between interesting and not interesting news in the majority of the cases. The work represents a first step towards generating meaningful explanations of recommendations. Future research will try to learn the interplay between explanations and other types of factors in a fully fledged recommender system, like user engagement, dwelling time or perception on system quality. An interesting direction to investigate would be that of learning non-trivial combinations of different explanations, and measuring their direct impact on users. In the case, the Markov Logic Network machinery could be useful to learn the right way to merge them, in order to devise a readable friendly personalized explanation.

6.2 List of Publications

This section lists the references to the papers – published or still under review – produced during the Ph.D. studies. The list of the papers strictly related to the Ph.D. topic is organized to reflect the contributions presented in the chapters of the thesis. Please note that results presented in Chapter 5 [27] were carried out during a research period at **Yahoo! Research** (Barcellona, Spain). At the end of this list we have two more papers, written during a internship at the **Digital Enterprise Research Institute** (Galway, Ireland). In [40] we propose the **Sindice-2011 Dataset**, a real-world data collection shared with the research community in order to make a significant step forward in web entity search. The paper reports several statistics about the dataset, useful for developing appropriate search systems. The dataset contains 11 billion statements associated with about 1.7 billion entities. In [41] we introduce a method to help users in formulating complex SPARQL queries across multiple heterogeneous data sources. Even if the structure and vocabulary of the data sources are unknown to the user, the user is able to quickly and easily formulate her queries. Our method is based on a summary of the data graph and assists the user during an interactive query formulation by recommending possible structural query elements. Initial experimentations show that our method can significantly reduce the effort required to formulate a query.

Chapter 2 - Query Biased Snippets

Caching query-biased snippets for efficient retrieval D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, F. Silvestri *Proceedings of the 14th International Conference on Extending Database Technology* Uppsala, Sweden, March 21-24, 2011;

Chapter 3 - Semantic Query Recommendations

When Entities Meet Query Recommender Systems: Semantic Search Shortcuts D. Ceccarelli, S. Gordea, C. Lucchese, F.M. Nardini, R. Perego *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13*, Coimbra, Portugal March 18-22, 2013;

On suggesting entities as web search queries D. Ceccarelli, S. Gordea, C. Lucchese, F.M. Nardini, R. Perego *Proceedings of the 4th edition of the Italian Information Retrieval Workshop* Pisa, Italy, January 16-17, 2013;

Improving europeana search experience using query logs D. Ceccarelli, S. Gordea, C. Lucchese, F. Nardini, G. Tolomei *Research and Advanced Technology for Digital Libraries - International Conference on Theory and Practice of Digital Libraries, TPDL 2011*, Berlin, Germany, September 26-28, 2011.

Discovering Europeana Users' Search Behavior D. Ceccarelli, S. Gordea, C. Lucchese, F. Nardini, R. Perego, G. Tolomei *ERCIM News 86* July 2011.

Chapter 4 - Learning relatedness measures for entity linking

Learning relatedness measures for entity linking D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, S. Trani *Proceedings of the 22th ACM International Conference on Information and Knowledge Management, CIKM'13*, October 27th - November 1st, 2013, Burlingame, CA, USA.

Dexter: an Open Source Framework for Entity Linking D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, S. Trani *Proceedings of the Sixth International Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR'13)*, October 28th, 2013, Burlingame, CA, USA.

Chapter 5 - News Explanations

Learning to Explain News Recommendations to Users R. Blanco, D. Ceccarelli, C. Lucchese, R. Perego, F. Silvestri *Submitted for review to ACM Transactions on Intelligent Systems and Technology (ACM TIST)*;

You should read this! let me explain you why: explaining news recommendations to users R. Blanco, D. Ceccarelli, C. Lucchese, R. Perego, F. Silvestri *21st ACM International Conference on Information and Knowledge Management, CIKM'12* Maui, HI, USA October 29 - November 02, 2012.

Other Publications

Twitter Anticipates Bursts of Requests for Wikipedia Articles G. Tolomei, S. Orlando, D. Ceccarelli, D., C. Lucchese *Proceedings of the CIKM 2013 Workshop on Data-driven User Behavioral Modelling and Mining from Social Media*, October 28th, 2013, Burlingame, CA, USA.

Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation S. Campinas, T.E. Perry, D. Ceccarelli, R. Delbru, G. Tummarello *23rd International Workshop on Database and Expert Systems Applications, DEXA 2012*, Vienna, Austria, September 3-7, 2012;

The sindice-2011 dataset for Entity-Oriented Search in the Web of Data S. Campinas, D. Ceccarelli, T.E. Perry, R. Delbru, K. Balog, G. Tummarello *The first international workshop on entity-oriented search, EOS'11* Beijing, China, July 25-29, 2011.

Bibliography

[1] Serge Abiteboul. *Querying semi-structured data*. Springer, 1997.

[2] S Fissaha Adafre, Maarten de Rijke, and E Tjong Kim Sang. Entity retrieval. *Proceedings of RANLP, Bulgaria, September*, 2007.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.

[4] J. Ahn, P. Brusilovsky, J. Grady, D. He, and S.Y. Syn. Open user profiles for adaptive news systems: help or harm? In *Proceedings of the 16th international conference on World Wide Web*, pages 11–20. ACM, 2007.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[6] V. Authors. About web analytics association. `http://www.webanalyticsassociation.org/?page=aboutus`, retrieved on December 2010.

[7] R. Baeza-Yates. Applications of web query mining. *Advances in Information Retrieval*, pages 7–22, 2005.

[8] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges in distributed information retrieval. In *International Conference on Data Engineering (ICDE)*, 2007.

[9] R. Baeza-Yates, A. Gionis, F.P. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. *ACM Transactions on the Web (TWEB)*, 2(4):1–28, 2008.

[10] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[11] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. The impact of caching on search engines. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 183–190, New York, NY, USA, 2007. ACM.

[12] Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 76–85. ACM, 2007.

[13] Ricardo A. Baeza-yates, Carlos A. Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proc. EDBT'04*, pages 588–596, 2004.

[14] K. Balog, P. Serdyukov, and A.P. Vries. Overview of the trec 2010 entity track. Technical report, DTIC Document, 2010.

[15] Ranieri Baraglia, Fidel Cacheda, Victor Carneiro, Diego Fernandez, Vreixo Formoso, Raffaele Perego, and Fabrizio Silvestri. Search shortcuts: a new approach to the recommendation of queries. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 77–84, New York, NY, USA, 2009. ACM.

[16] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The Google cluster architecture. *IEEE micro*, 23(2):22–28, 2003.

[17] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The google cluster architecture. *Micro, IEEE*, 23(2):22 – 28, mar. 2003.

[18] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, Ophir Frieder, and David Grossman. Temporal analysis of a very large topically categorized web query log. *J. Am. Soc. Inf. Sci. Technol.*, 58:166–178, January 2007.

[19] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, and Ophir Frieder. Hourly analysis of a very large topically categorized web query log. In *In Proc. SIGIR'04*, pages 321–328. ACM Press, 2004.

[20] Steven M. Beitzel, Eric C. Jensen, Ophir Frieder, David D. Lewis, Abdur Chowdhury, and Aleksander Kolcz. Improving automatic query classification via semi-supervised learning. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 42–49, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Steven M. Beitzel, Eric C. Jensen, David D. Lewis, Abdur Chowdhury, and Ophir Frieder. Automatic classification of web queries using very large unlabeled query logs. *ACM Trans. Inf. Syst.*, 25, April 2007.

[22] Gordon Bell, Tony Hey, and Alex Szalay. Beyond the data deluge. *Science*, 323(5919):1297–1298, 2009.

[23] Tim Berners-Lee. Linked data, 2006. *URL http://www. w3. org/DesignIssues/LinkedData. html*, 33:34, 2006.

[24] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[25] Mustafa Bilgic and Raymond J. Mooney. Explaining recommendations: Satisfaction vs. promotion. In *Proceedings of Beyond Personalization 2005: A Workshop on the Next Stage of Recommender Systems Research at the 2005 International Conference on Intelligent User Interfaces*, San Diego, CA, January 2005.

[26] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[27] R. Blanco, D. Ceccarelli, C. Lucchese, R. Perego, and F. Silvestri. You should read this! let me explain you why: explaining news recommendations to users. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1995–1999. ACM, 2012.

[28] Roi Blanco and Hugo Zaragoza. Finding support sentences for entities. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 339–346, New York, NY, USA, 2010. ACM.

[29] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: model and applications. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 609–618. ACM, 2008.

[30] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. Query suggestions using query-flow graphs. In *Proc. WSCD'09*. ACM, 2009.

[31] Paolo Boldi, Francesco Bonchi, Carlos Castillo, and Sebastiano Vigna. From 'dango' to 'japanese cakes': Query reformulation models and patterns. In *Proc. WI'09*. IEEE, September 2009.

[32] Francesco Bonchi, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini. Efficient query recommendations in the long tail via center-piece subgraphs. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 345–354. ACM, 2012.

[33] Ilaria Bordino, Gianmarco De Francisci Morales, Ingmar Weber, and Francesco Bonchi. From machu_picchu to rafting the urubamba river: anticipating information needs via the entity-query graph. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 275–284. ACM, 2013.

[34] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine* 1. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[35] Daniele Broccolo, Lorenzo Marcon, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. An efficient algorithm to generate search shortcuts. Technical Report N. /cnr.isti/2010-TR-017, CNR ISTI Pisa Italy, 2010.

[36] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36:3–10, September 2002.

[37] Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, Donald Metzler, Lance Riedel, and Jeffrey Yuan. Online expansion of rare queries for sponsored search. In *Proc. WWW'09*. ACM, 2009.

[38] Andrei Z. Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang. Robust classification of rare queries using web knowledge. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 231–238, New York, NY, USA, 2007. ACM.

[39] Razvan Bunescu and Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of EACL*, volume 6, pages 9–16, 2006.

[40] S. Campinas, D. Ceccarelli, T.E. Perry, R. Delbru, K. Balog, and G. Tummarello. The sindice-2011 dataset for entity-oriented search in the web of data. pages 26–32, 2011.

[41] S. Campinas, T.E. Perry, D. Ceccarelli, R. Delbru, and G. Tummarello. Introducing rdf graph summary with application to assisted sparql formulation. In *WebS '12: 11th International Workshop on Web Semantics and Information Processing. Vienna, Austria, September 2012*, 2012.

[42] D. Ceccarelli, S. Gordea, C. Lucchese, F. Nardini, and G. Tolomei. Improving europeana search experience using query logs. pages 384–395. Springer, 2011.

[43] D. Ceccarelli, S. Gordea, C. Lucchese, F.M. Nardini, and R. Perego. When entities meet query recommender systems: Semantic search shortcuts. In *SAC '13: 28th Symposium On Applied Computing, Coimbra, Portugal, March 2013*, 2013.

[44] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and F. Silvestri. Caching query-biased snippets for efficient retrieval. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 93–104. ACM, 2011.

[45] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and S. Trani. Learning relatedness measures for entity linking. In *Proceedings of the 22st ACM international conference on Information and knowledge management*. ACM, 2013.

[46] D. Ceccarelli, C. Lucchese, Orlando S., R. Perego, and S. Trani. Dexter: an open source framework for entity linking. In *In Sixth International Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR)*. ACM, 2013.

[47] Soumen Chakrabarti, Sasidhar Kasturi, Bharath Balakrishnan, Ganesh Ramakrishnan, and Rohit Saraf. Compressed data structures for annotated web search. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 121–130, New York, NY, USA, 2012. ACM.

[48] R.L. Cilibrasi and P.M.B. Vitanyi. The google similarity distance. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):370 – 383, march 2007.

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

[50] S. Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *Proceedings of EMNLP-CoNLL*, volume 6, pages 708–716, 2007.

[51] Hong-Jie Dai, Richard Tzong-Han Tsai, and Wen-Lian Hsu. Entity disambiguation using a markov-logic network. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 846–855, Chiang Mai, Thailand, November 2011. Asian Federation of Natural Language Processing.

[52] A. de Vries, A.M. Vercoustre, J. Thom, N. Craswell, and M. Lalmas. Overview of the inex 2007 entity ranking track. *Focused Access to XML Documents*, pages 245–251, 2008.

[53] Darcy DiNucci. Fragmented future. *Print*, 53(4):32, 1999.

[54] H P Edmundson. New Methods in Automatic Extracting. *Computing*, 16(2):264–285, 1969.

[55] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, 2006.

[56] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24:51–78, January 2006.

[57] P. Ferragina and U. Scaiella. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proc. CIKM'10*, pages 1625–1628. ACM, 2010.

[58] J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Ann. Statist*, 29(5):1189–1232, 2001.

[59] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[60] Michael Genesereth and Nils Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[61] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 407–414. ACM, 2007.

[62] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*. Interdisciplinary Statistics Series. Chapman & Hall, 1996.

[63] Luis Gravano, Vasileios Hatzivassiloglou, and Richard Lichtenstein. Categorizing web queries according to geographical locality. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 325–333, New York, NY, USA, 2003. ACM.

[64] Kevin Haas, Peter Mika, Paul Tarjan, and Roi Blanco. Enhanced results for web search. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 725–734, New York, NY, USA, 2011. ACM.

[65] X. Han, L. Sun, and J. Zhao. Collective entity linking in web text: a graph-based method. In *Proc. SIGIR'11*, pages 765–774. ACM, 2011.

[66] Xianpei Han, Le Sun, and Jun Zhao. Collective entity linking in web text: a graph-based method. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 765–774, New York, NY, USA, 2011. ACM.

[67] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 241–250, New York, NY, USA, 2000. ACM.

[68] J. Hoffart, M.A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 782–792. Association for Computational Linguistics, 2011.

[69] Johannes Hoffart, Stephan Seufert, Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. Kore: keyphrase overlap relatedness for entity disambiguation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 545–554, New York, NY, USA, 2012. ACM.

[70] Vera Hollink, Theodora Tsikrika, and Arjen P de Vries. Semantic search log analysis: a method and a study on professional image search. *Journal of the American Society for Information Science and Technology*, 62(4):691–713, 2011.

[71] Shujian Huang, Yabing Zhang, Junsheng Zhou, and Jiajun Chen. Coreference resolution using markov logic network. *Science*, 41:157–168, 2009.

[72] Bouke Huurnink, Laura Hollink, Wietske van den Heuvel, and Maarten de Rijke. Search behavior of media professionals at an audiovisual archive: A transaction log analysis. *Journal of the American Society for Information Science and Technology*, 61(6):1180–1197, 2010.

[73] Bernard J. Jansen. *Understanding User-Web Interactions via Web Analytics*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.

[74] Bernard J. Jansen and Marc Resnick. An examination of searcher's perceptions of nonsponsored and sponsored links during ecommerce web searching. *J. Am. Soc. Inf. Sci. Technol.*, 57:1949–1961, December 2006.

[75] Bernard J. Jansen and Amanda Spink. An analysis of web searching by european alltheweb.com users. *Inf. Process. Manage.*, 41:361–381, March 2005.

[76] Bernard J. Jansen and Amanda Spink. How are we searching the world wide web?: a comparison of nine search engine transaction logs. *Inf. Process. Manage.*, 42:248–263, January 2006.

[77] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32:5–17, April 1998.

[78] Bernard J. Jansen, Amanda Spink, and Sherry Koshman. Web searcher interaction with the dogpile.com metasearch engine. *J. Am. Soc. Inf. Sci. Technol.*, 58:744–755, March 2007.

[79] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[80] Thorsten Joachims. Optimizing search engines using clickthrough data. In *8th International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

[81] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[82] Rosie Jones and Kristina L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM '08*, pages 699–708. ACM, 2008.

[83] C. Kang, S. Vadrevu, R. Zhang, R. Zwol, L.G. Pueyo, N. Torzec, J. He, and Y. Chang. Ranking related entities for web search queries. In *Proceedings of the 20th international conference companion on World wide web*, pages 67–68. ACM, 2011.

[84] Tapas Kanungo and David Orr. Predicting the readability of short web summaries. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 202–211. ACM, 2009.

[85] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis.* Wiley Interscience, New York, 1990.

[86] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[87] Graham Klyne, Jeremy J Carroll, and Brian McBride. Resource description framework (rdf): Concepts and abstract syntax. *W3C recommendation*, 10, 2004.

[88] Sherry Koshman, Amanda Spink, and Bernard J. Jansen. Web searching on the vivisimo search engine. *J. Am. Soc. Inf. Sci. Technol.*, 57:1875–1887, December 2006.

[89] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of wikipedia entities in web text. In *Proc. SIGKDD'09*, pages 457–466. ACM, 2009.

[90] J.R. Landis and G.G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, pages 159–174, 1977.

[91] Tessa Lau and Eric Horvitz. Patterns of search: analyzing and modeling web query refinement. In *Proceedings of the seventh international conference on User modeling*, pages 119–128, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.

[92] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th international conference on World Wide Web*, pages 19–28. ACM, 2003.

[93] Ronny Lempel and Shlomo Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 19–28, New York, NY, USA, 2003. ACM.

[94] Jie Lu and Jamie Callan. Pruning long documents for distributed information retrieval. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 332–339. ACM, 2002.

[95] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. Identifying task-based sessions in search engine query logs. In *Proc. WSDM'11*, pages 277–286, New York, NY, USA, 2011. ACM.

[96] H.P. Luhn. The automatic creation of literature abstracts. *IBM J. of research and development*, 2(2):159–165, 1958.

[97] Christoph Mangold. A survey and classification of semantic search approaches. *International Journal of Metadata, Semantics and Ontologies*, 2(1):23–34, 2007.

[98] C.D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press New York, NY, USA, 2008.

[99] E.P. Markatos. On caching search engine query results. *Computer Communications*, 24(2):137–143, 2001.

[100] Mazlita Mat-Hassan and Mark Levene. Associating search and navigation behavior through log analysis: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 56:913–934, July 2005.

[101] David McSherry. Explanation in recommender systems. *Artif. Intell. Rev.*, 24(2):179–197, October 2005.

[102] Edgar Meij, Marc Bron, Laura Hollink, Bouke Huurnink, and Maarten De Rijke. Learning semantic query suggestions. In *The Semantic Web-ISWC 2009*, pages 424–440. Springer, 2009.

[103] Edgar Meij, Marc Bron, Laura Hollink, Bouke Huurnink, and Maarten de Rijke. Mapping queries to the linking open data cloud: A case study using dbpedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):418–433, 2011.

[104] D. Metzler and T. Kanungo. Machine learned sentence selection strategies for query-biased summarization. *Learning to Rank for Information Retrieval*, 40, 2008.

[105] R. Mihalcea and A. Csomai. Wikify!: linking documents to encyclopedic knowledge. In *Proceedings of the 16th ACM international conference on Information and knowledge management*, CIKM'07, pages 233–242, New York, NY, USA, 2007. ACM.

[106] Peter Mika, Edgar Meij, and Hugo Zaragoza. Investigating the semantic gap through query log analysis. In *The Semantic Web-ISWC 2009*, pages 441–455. Springer, 2009.

[107] Peter Mika and Tim Potter. Metadata statistics for a large web corpus. In *Proceedings of the Linked Data Workshop (LDOW) at the International World Wide Web Conference*, 2012.

[108] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[109] David Milne and Ian H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *In Proceedings of AAAI 2008*, 2008.

[110] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 509–518, New York, NY, USA, 2008. ACM.

[111] Tim Oreilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1):17, 2007.

[112] H. Cenk Ozmutlu, Amanda Spink, and Seda Ozmutlu. Analysis of large data logs: an application of poisson sampling on excite web queries. *Inf. Process. Manage.*, 38:473–490, July 2002.

[113] Seda Ozmutlu, Amanda Spink, and Huseyin C. Ozmutlu. A day in the life of web searching: an exploratory study. *Inf. Process. Manage.*, 40:319–345, March 2004.

[114] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[115] P. Pantel and A. Fuxman. Jigs and lures: Associating web queries with structured entities. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 83–92. Association for Computational Linguistics, 2011.

[116] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06, New York, NY, USA, 2006. ACM.

[117] David A Patterson and John L Hennessy. *Computer organization and design: the hardware/software interface.* Morgan Kaufmann, 2008.

[118] Benjamin Piwowarski and Hugo Zaragoza. Predictive user click models based on click-through history. In *Proc. CIKM'07.* ACM, 2007.

[119] D.R. Radev, E. Hovy, and K. McKeown. Introduction to the special issue on summarization. *Computational Linguistics*, 28(4):399–408, 2002.

[120] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proc. KDD'05.* ACM Press, 2005.

[121] Matthew Richardson. Learning about the world through long-term query logs. *ACM TWEB*, 2(4):1–27, 2008.

[122] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, February 2006.

[123] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.

[124] A. Scime. *Web Mining: Applications and Techniques.* IGI Publishing Hershey, PA, USA, 2004.

[125] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532, 1995.

[126] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Q2C@UST: our winning solution to query classification in kddcup 2005. *SIGKDD Explor. Newsl.*, 7:100–110, December 2005.

[127] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. Linden: linking named entities with knowledge base via semantic knowledge. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20*, pages 449–458. ACM, 2012.

[128] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33:6–12, September 1999.

[129] Fabrizio Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 4(12):1–174, 2010.

[130] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *In ICDM*, pages 572–582. IEEE Computer Society Press, 2006.

[131] Yang Song and Li-wei He. Optimal rare query suggestion with implicit user feedback. In *Proc. WWW'10*. ACM, 2010.

[132] Amanda Spink, Bernard J. Jansen, Dietmar Wolfram, and Tefko Saracevic. From e-sex to e-commerce: Web search changes. *Computer*, 35:107–109, March 2002.

[133] Amanda Spink, H. Cenk Ozmutlu, and Daniel P. Lorence. Web searching for sexual information: an exploratory study. *Inf. Process. Manage.*, 40:113–123, January 2004.

[134] Amanda Spink, Minsoo Park, Bernard J. Jansen, and Jan Pedersen. Multitasking during web search sessions. *IPM*, 42(1):264–275, 2006.

[135] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the web: the public and their queries. *J. Am. Soc. Inf. Sci. Technol.*, 52:226–234, February 2001.

[136] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl.*, 1:12–23, January 2000.

[137] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[138] A SZALA. Science in an exponential world. *Nature*, 440:2020, 2006.

[139] Idan Szpektor, Aristides Gionis, and Yoelle Maarek. Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th international conference on World wide web*, pages 47–56. ACM, 2011.

[140] Jaime Teevan, Eytan Adar, Rosie Jones, and Michael A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 151–158, New York, NY, USA, 2007. ACM.

[141] N. Tintarev and J. Masthoff. Designing and evaluating explanations for recommender systems. *Recommender Systems Handbook*, pages 479–510, 2011.

[142] Nava Tintarev and Judith Masthoff. Evaluating the effectiveness of explanations for recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):399–439, 2012.

[143] A. Tombros and M. Sanderson. . In *Proc. of the 21st Annual Inter. ACM SIGIR Conf. on Research and Development in Information Retrieval*. ACM, 1998.

[144] Y. Tsegay, S. Puglisi, A. Turpin, and J. Zobel. Document Compaction for Efficient Query Biased Snippet Generation. *Advances in Information Retrieval*, pages 509–520, 2009.

[145] Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice. com: Weaving the open linked data. In *The Semantic Web*, pages 552–565. Springer, 2007.

[146] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams. Fast generation of result snippets in web search. *In: ACM SIGIR*, 39(5):127–134, 2007.

[147] R. van Zwol, L. Garcia Pueyo, M. Muralidharan, and B. Sigurbjörnsson. Machine learned ranking of entity facets. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 879–880. ACM, 2010.

[148] R. van Zwol, L. Garcia Pueyo, M. Muralidharan, and B. Sigurbjornsson. Ranking entity facets based on user click feedback. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 192–199. IEEE, 2010.

[149] R. van Zwol, L. Garcia Pueyo, M. Muralidharan, and B. Sigurbjornsson. Ranking entity facets based on user click feedback. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 192–199. IEEE, 2010.

[150] V Vapnik. The nature of 6tatistical learning theory. *Data mining and knowledge discovery*, pages 1–47, 6.

[151] A.M. Vercoustre, J.A. Thom, and J. Pehcevski. Entity ranking in wikipedia. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1101–1106. ACM, 2008.

[152] J. Vig, S. Sen, and J. Riedl. Tagsplanations: explaining recommendations using tags. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 47–56. ACM, 2009.

[153] David Vogel, Steffen Bickel, Peter Haider, Rolf Schimpfky, Peter Siemen, Steve Bridges, and Tobias Scheffer. Classifying search engine queries using the web as background knowledge. *SIGKDD Explor. Newsl.*, 7:117–122, December 2005.

[154] Jue Wang and Pedro Domingos. Hybrid markov logic networks. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 1106–1111. AAAI Press, 2008.

[155] G. Weikum and M. Theobald. From information to knowledge: harvesting entities and relationships from web sources. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 65–76. ACM, 2010.

[156] Q. Wu, C.J.C. Burges, K.M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

[157] Y. Xie and D. O'Hallaron. Locality in search engine queries and its implications for caching. In *Proceedings of IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002.

[158] Jack L. Xu and Amanda Spink. Web research: The excite study. In Gordon Davies and Charles B. Owen, editors, *WebNet*, pages 581–585. AACE, 2000.

[159] M.A. Yosef, J. Hoffart, I. Bordino, M. Spaniol, and G. Weikum. Aida: An online tool for accurate disambiguation of named entities in text and tables. *Proceedings of the VLDB Endowment*, 4(12), 2011.

[160] Katsumasa Yoshikawa, Masayuki Asahara, and Yuji Matsumoto. Jointly extracting japanese predicate-argument relation with markov logic. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1125–1133, Chiang Mai, Thailand, November 2011. Asian Federation of Natural Language Processing.

[161] H. Zaragoza, H. Rode, P. Mika, J. Atserias, M. Ciaramita, and G. Attardi. Ranking very many typed entities on wikipedia. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 1015–1018. ACM, 2007.