Task-Based Support in Search Engines

by

Darío Garigliotti

Thesis submitted in fulfilment of
the requirements for the degree of
PHILOSOPHIAE DOCTOR
(PhD)

Faculty of Science and Technology
Department of Electric Engineering and Computer Science
2020

University of Stavanger
NO-4036 Stavanger
NORWAY
www.uis.no

ISBN: 978-82-7644-883-2
ISSN: 1890-1387
PhD: Thesis UiS No. 485

Acknowledgements

I would like to thank the members of the assessment committee, Prof. Emine Yilmaz, Prof. Matthias Hagen, and Prof. Vinay J. Setty, for reviewing my dissertation and providing very detailed and helpful comments and suggestions.

I would like to thank my supervisor, Prof. Krisztian Balog. He trusted in my skills, and helped me improve them and develop new ones in countless opportunities. His encouragement and insights always left me inspired to advance in my work after each meeting. He was available to help me whenever I needed regardless the reason. He looked for enriching my academic life as much as possible. When I met him in September 2015, he already gave me some hints about his long-term vision for his research team. Since then, I have witnessed him accomplish many of his ambitions through a permanent dedication. The admiration for the tenacity is likely reciprocal, as he has seen me fall and stand up three times from a sled pulled by wild dogs. I am certainly happy and honored to have had Krisztian in this path.

I would like to thank my co-supervisor, Prof. Kjetil Nørvåg, for the numerous occasions that he and his group kindly hosted our research visits in Trondheim.

I would like to thank the head of our department, Dr. Tom Ryen, for his support and speedy response whenever I needed. I would also like to thank everyone in the department who helped me through issues regarding, among others, paperwork, equipment, and safety.

I would like to thank my colleagues and friends. They helped me when I had just arrived in Norway, they hosted me for Christmas, they tried to teach me how to ski, they gave me advice and insights through our time in the office, they walked with me rocky hills in sunny mornings and down elegant streets in cold evenings.

I would like to thank my parents for their immeasurable efforts and support to help me be what I am. Knowing it or not, they made me a reader, a designer, a singer, a traveller, a dreamer.

Abstract

Web search has become a key technology on which people rely on a daily basis. The evolution of the search experience has also shaped the expectations of people about it. Many users seem to expect today's search engines to behave like a kind of "wise interpreter," capable of understanding the meaning behind a search query, realizing its current context, and responding to it directly and appropriately. Semantic search encompasses a large portion of information retrieval (IR) research devoted to study more meaningful representations of the user information need. Entity cards, direct displays, and verticals are examples of how major commercial search engines have indeed capitalized on query understanding. Search is usually performed with a specific goal underlying the query. In many cases, this goal consists of a nontrivial task to be completed. Current search engines support a small set of basic tasks, and most of the knowledge-intensive workload for supporting more complex tasks is left to the user. Task-based search can be viewed as an information access paradigm that aims to enhance search engines with functionalities for recognizing the underlying tasks in searches and providing support for task completion. The research presented in this thesis focuses on utilizing and extending methods and techniques from semantic search in the next stage of the evolution: to support users in achieving their tasks. Our work can be grouped in three grand themes: (1) *Entity type information for entity retrieval:* we conduct a systematic evaluation and analysis of methods for type-aware entity retrieval, in terms of three main dimensions. We revisit the problem of hierarchical target type identification, present a state-of-the-art supervised learning method, and analyze the usage of automatically identified target entity types for type-aware entity retrieval; (2) *Entity-oriented search intents:* we propose a categorization scheme for entity-oriented search intents, and study the distributions of entity intent categories per entity type. We develop a method for constructing a knowledge base of entity-oriented search intents; and (3) *Task-based search:* we design a probabilistic generative framework for task-based query suggestion, and principledly estimate each of its components. We introduce the problems of query-based task recommendation and mission-based task recommendation, and establish respective suitable baselines.

Table of contents

Chapter 1

Introduction

Web search is a human experience of a very rapid and impactful evolution. It has become a key technology on which people rely daily for getting information about almost everything. This evolution of the search experience has also shaped the expectations of people about it. Many users seem to expect today's web search engines to behave like a kind of "wise interpreter," capable of understanding the intent and *meaning* behind a search query, realizing her current context, and responding to it directly and appropriately [6]. Search by meaning, or *semantic search*, rather than just literal matches, became possible by a large portion of information retrieval (IR) research devoted to study semantically more meaningful representations of the information need expressed by the user query. Consider the examples in Fig. 1.1, which shows the top search result from Google for each of three individual queries. The left-side query, "oslo weather," leads to a widget that reports recent weather conditions, as well as presents forecasts for the next twenty-four hours and the next seven days. The display in the middle contains the result of the latest Liverpool-Barcelona match, with some statistics, and provides a link to a video with the match highlights on YouTube. The last search result presents an interactive chart with the stock exchange performance for Tesla Inc. intended in the query "tsla." These three examples correspond to direct displays in which the search engine has gone beyond the traditional ten blue links, by aiming to *understand* the query intent and return in the search engine result page (SERP) a direct answer when possible. Note also that the verticals appropriately provide rich content depending on each query: maps and images for the weather, videos and images for the match, and finance and news for the stock performance. Direct displays and verticals are examples of how major commercial search engines have indeed responded to user expectations, capitalizing on query semantics, i.e., *query understanding*, by

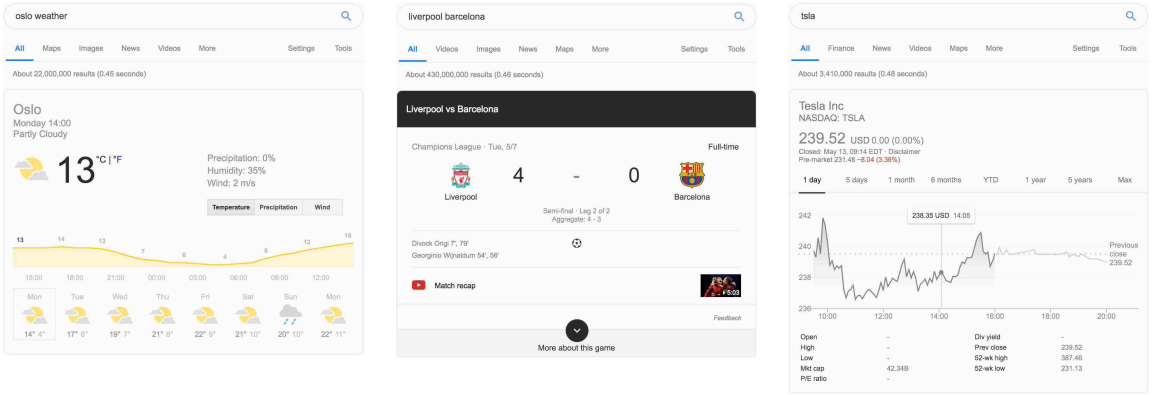


Fig. 1.1 Examples of direct displays in Google SERPs.

introducing features which not only provide information directly but also engage the user to stay interacting with the SERP.

Knowledge panels are perhaps one of the most prominent features in the web search experience reflecting this recent evolution trend from search engines into *answer engines* [134]. These panels frequently correspond to *entities*, and thus are also known as entity cards. Entities, such as persons, organizations, and locations, are natural units for organizing information [7]. The pivotal component that sparked this evolution trend is the increased availability of structured data published in knowledge repositories and *knowledge bases* (KBs), such as Wikipedia, DBpedia, or the Google Knowledge Graph, now primary sources of information for *entity-oriented search*. Figure 1.2 shows the entity card that results from searching "oslo" in Google. The entity Oslo is described by some of its properties, retrieved from some KB of reference, such as a brief abstract, a handful of literal values like elevation and population, and related entities like the University of Oslo. Our notion of semantics is inherently based on meaningful structures as the ones mentioned above. Given the large portion of web search queries looking for entities, entities and their properties—attributes, types, and relationships—are first-class citizens in our space of structured knowledge.

Semantic search is a search paradigm that intends to identify and leverage structured knowledge to answer an information need, by trying to understand the query. That is, giving meaning to information pieces within the query and within the searched items, by mapping them to elements in some knowledge structure of reference. Semantic search can then be seen as a rich toolbox, encompassing multiple techniques to recognize essential knowledge units in the information need, identify them uniquely in the underlying knowledge repositories, and exploit them to address a particular aspect of

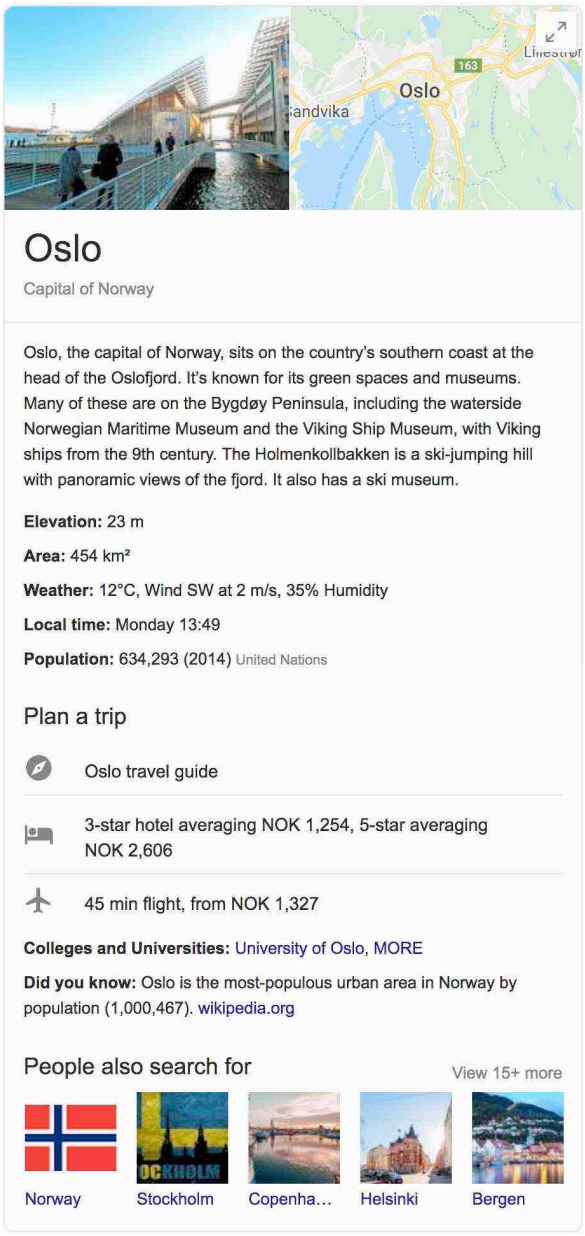


Fig. 1.2 Google entity card for the query "oslo."

query understanding [7]. To some extent, the whole research presented in this thesis centers around query understanding: identifying target entity types, recognizing and building a knowledge base of entity-oriented search intents, and providing support for task-based search by the means of query suggestions or task recommendations.

Search is usually performed with a specific goal underlying the query. In many cases, this goal consists of a nontrivial *task* to be completed. Indeed, task-based search corresponds to a considerable portion of the query volume [50, 121]. Addressing task-based

search can have a large impact on search behavior [31], yet the interaction processes behind performing a complex task are very far from being fully understood [167]. The middle part of the entity card shown in Fig. 1.2 can be also seen as recommending a suitable task that the user might be interested in, "Plan a trip," and provides links anchored by subtasks like exploring a travel guide, booking a hotel room, and booking a flight. This task recommendation, inferred from the query seeking a city, is not generalized but rather seems available only for certain kind of queries, which suggests that it may be based on some convenient structured interpretation such as query templates [26]. Current search engines support a small set of basic tasks, and most of the knowledge-intensive workload for supporting more complex tasks is left to the user. The ultimate challenge then is to build useful systems "to achieve work task completion" [180]. The research presented in this thesis focuses on utilizing and extending methods and techniques from semantic search in the next stage of the evolution of search engines, namely, to support users in achieving their tasks, referred to as *task completion* [6].

1.1 Research Outline

We view task-based search as an information access paradigm that aims to enhance search engines with functionalities for recognizing the underlying tasks in searches and providing support for task completion.

Our work can be grouped in three grand themes: entity type information for entity retrieval, entity-oriented search intents, and task-based search.

1.1.1 Entity Type Information for Entity Retrieval

The first challenge we identify is *utilizing entity type information*. Types—semantic classes grouping multiple entities—are a distinctive property of entities. This grouping characteristic makes types capable to generalize information of many entities, since that types are much more stable in comparison with new entities daily emerging. Types are typically organized in a hierarchy, referred to as *type taxonomy* or *ontology*. Type information is known to contribute to the task of *entity retrieval*, this is, returning a ranked list of entities upon an entity-oriented query.

As an illustration, consider the scenario of a user planning a trip across Europe, and so, wishing to know in which cities she can use the Uber car sharing service. After issuing the query "cities in europe where uber is available," the search engine we

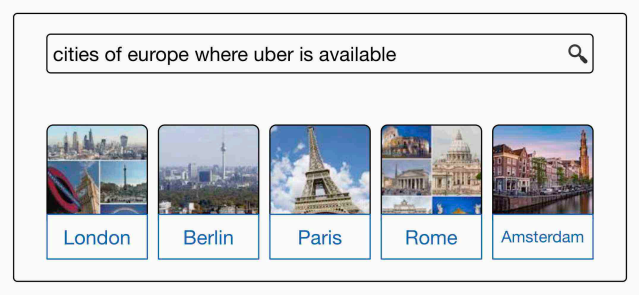


Fig. 1.3 Excerpt from an envisaged search engine with type-aware entity retrieval.

envisage would return a list of entities, according to some ranking criteria, as shown in Fig. 1.3. To achieve this, the system could try to identify the type of entities (here, European cities) that the query is seeking, which we refer to as *target entity types*. This type-based information could then be combined with the types assigned to each entity in a reference knowledge base, in order to improve the ranking of results.

As we will see, the nature of type information brings about a set of challenges for its usage. The primary objective of this research is then formulated as the following research question:

RQ1: How can entity type information be utilized in ad-hoc entity retrieval?

We identify three main dimensions within the problem of *type-aware entity retrieval*: (i) type taxonomy, (ii) retrieval model, and (iii) representation of hierarchical type information. Using a standard test collection, we compare all the combinations of these dimensions with respect to a strong text-based baseline. In these experiments, we assume "perfect" type-based information provided by a target entity types oracle. We provide general recommendations on using type-based information for entity retrieval, based on an extensive analysis.

As described in the previous example, we also aim to account for the challenge of *target entity type identification*. Indeed, the single-search-box paradigm has become widespread, and ordinary users have little incentive (or knowledge) to formulate structured queries where they would explicitly indicate the target entity types for the query. This motivates our next research question:

RQ2: How can we automatically identify target entity types?

Given that types are commonly organized in a hierarchy, taking into account this hierarchical information seems convenient when identifying target entity types. Indeed, we revisit the *hierarchical target type identification* task, that is, the problem of finding the single most specific type within a reference type taxonomy that is general enough to cover all relevant entities for the query. We then build a test collection for evaluating this task, as well as we propose an approach for automatically identifying target entity types and verify that it is able to significantly outperform previously established methods.

1.1.2 Entity-oriented Search Intents

The second challenge identified by this research deals with *understanding query intents*, that is, understanding what entity-oriented queries ask for, by observing patterns in the way that users express their search queries. Within the large category of *entity-oriented queries*, i.e., queries revolving around entities, we focus on studying those that typically consist of (one of the names of) an entity and possibly additional words that refine the intent behind the query. An envisaged search engine would identify the intent behind such an entity-oriented query and present a result object according to the nature of the intent. For example, a user might ask for the population of the city of Stavanger, say, by querying "stavanger population." The engine would recognize the population-related intent as a property that can be retrieved from the corresponding entry for Stavanger in a knowledge base of reference. The result might then boil down to a simple entity card for the city, that in particular displays an entry with the number of its population, as taken from the KB. This case is similar to the population field in the entity card for Oslo presented in Fig. 1.2. In another scenario, the user instead issues "taxi to stavanger airport." Here, we would instead aim for recognizing that the intent for the Stavanger Airport behind the refiner "taxi to" rather deals with performing a transaction and booking a taxi. A possible result format, illustrated in Fig. 1.4, allows the user to provide details of her travel, such as her position and desired time, and displays a button to order a taxi.

Our first objective then is to understand entity-related search intents by studying those query refiners. We address the following research question:

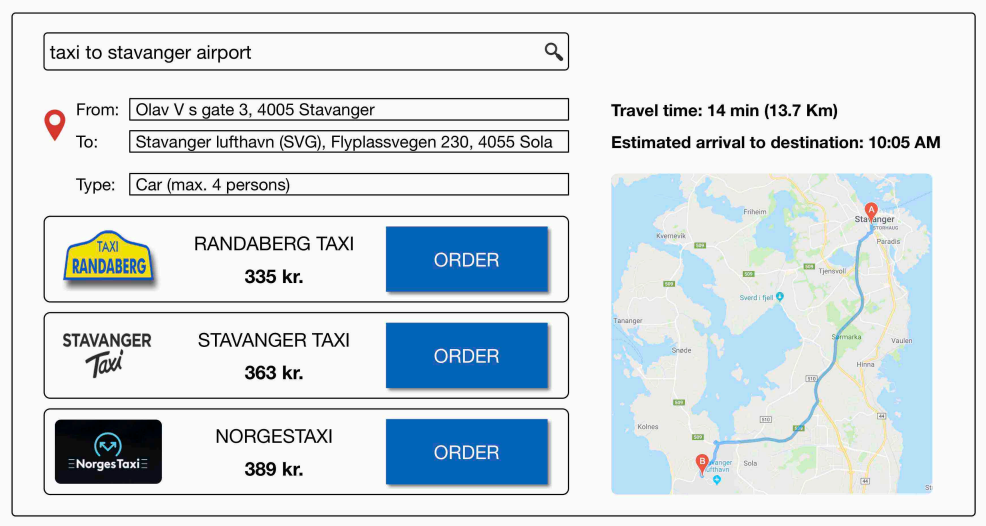


Fig. 1.4 Excerpt from an envisaged search engine with intent-driven answer displays.

RQ3: What do entity-oriented queries ask for, and how can they be fulfilled?

We obtain query patterns on the level of entity types, capturing various entity-oriented search intents. For example, by representing with *[airport]* any entity of the type airport, we are interested in the intent behind "taxi to" in the type-level query pattern *"taxi to [airport]"*. By analyzing an annotated collection of entity-oriented queries, aggregated by entity types, we obtain a clearer understanding of what entity-oriented queries ask for. In particular, we make observations regarding which searches can be fulfilled by looking up direct answers from a knowledge base (e.g., when seeking the population of a city), and which would require to interact with external services (such as booking a taxi).

Our follow-up step is building a structured repository or knowledge base of entity-oriented search intents. A strategy that represents search intents as structured knowledge could be leveraged in various applications, e.g., for generating entity cards or query recommendations. In particular, it could enable further mechanisms to provide support for service-oriented intents as the aforementioned scenario of ordering a taxi. This representation involves modeling search intents in a structured fashion, as well as devising methods for populating this KB. Hence, the following research question is of our interest:

RQ4: How can we build a knowledge base of entity-oriented search intents?

We devise an approach for automatically assigning intent categories to type-level query refiners, and a clustering stage of refiners which express the same underlying intent. Additionally, we represent each intent uniquely in a knowledge repository, and map to it the facts we acquire about its categorization and refiners. We evaluate performance both component-wise and end-to-end, and demonstrate that our approach is able to generate high-quality data.

1.1.3 Task-based Search

A third challenge of interest is *supporting task-based search*, as a mechanism to assist the user with articulating her information need. In particular, we are interested in generating query suggestions to support task-based search. Many search scenarios are driven by a specific target or goal, which is often complex and knowledge-intensive, making the user to issue a sequence of queries to eventually complete her underlying task. A way to formally frame this objective is by aiming for the generation, upon an initial query, of a ranked list of query suggestions that cover all the possible subtasks related to the task the user is trying to achieve. In an envisaged user interface like the one depicted in Fig. 1.5, these queries can be suggested once the user has issued her initial query. We ask the following research question:

RQ5: How can we generate query suggestions for supporting task-based search?

We propose an architecture for extracting keyphrases from a variety of sources and generating query suggestions from them. This architecture encompasses components to select documents from particular information sources, extract keyphrases from the documents, and combine the keyphrases with the initial query into query suggestions. Using a test collection from a dedicated benchmark campaign, we propose a principled approach for estimating each of these components.

We envisage a semantic search engine where a specific widget presents task-based query suggestions once the user has issued the initial query. These query suggestions may come in two flavors: query completions (QC) and query refinements (QR), differing only in that the former are prefixed by the initial query. Hence, we address

further related questions, such as answering whether a unified method can produce suggestions in both flavors, and whether a keyphrase-based method can avoid relying on suggestions from a major web search engine. We adopt a two-step approach consisting of suggestion generation and suggestion ranking steps, and focus exclusively on the first component, aiming to generate sufficiently many high-quality query suggestion candidates. We compare three different methods to generate candidate suggestions, using three information sources, and evaluate them in a purpose-built test collection of query suggestions for task-based search.

Web search is an experience that naturally lends itself to recommendations, including, as we mentioned, query suggestions and related entities. A possible extension of this strategy would be to recommend specific tasks to users, based on their search queries, such as planning a holidays trip or organizing a birthday party. This problem of *task recommendation* appears to be a suitable mechanism to reduce the user-side workload and to help complete the task at hand.

Consider, for example, the scenario illustrated in the left side of Fig. 1.5. A user wishes to organize a birthday party. With a task recommender in place, her single query "cake recipe" would lead to the recommendation of tasks including the task "Make a chocolate cake." Each task is taken from a task repository, similarly to how entities are stored in a knowledge base. For each task result, the engine could display buttons directing to specific elements of that task. In the example, the relevant information for the first and second tasks concerns methods for accomplishing the task (in the first task, the methods are particular chocolate cake recipes), and since in the third one there is a single known method, it rather provides direct access to the ingredients and the steps.

Our main inquiry here can be expressed with this research question:

RQ6: How can we recommend tasks based on search queries and missions?

We introduce the problem of *query-based task recommendation* and develop methods that combine well-established text-based matching techniques with continuous semantic representations. Using a purpose-built test collection, we find that our method is able to substantially outperform a strong text-based matching baseline. Further, we extend our approach to using a set of queries that all share the same underlying task, referred to as *search missions*, as input. In the example of Fig. 1.5, assuming that the user issues a second query "cake decorating ideas," the task "Decorate a cake" would also

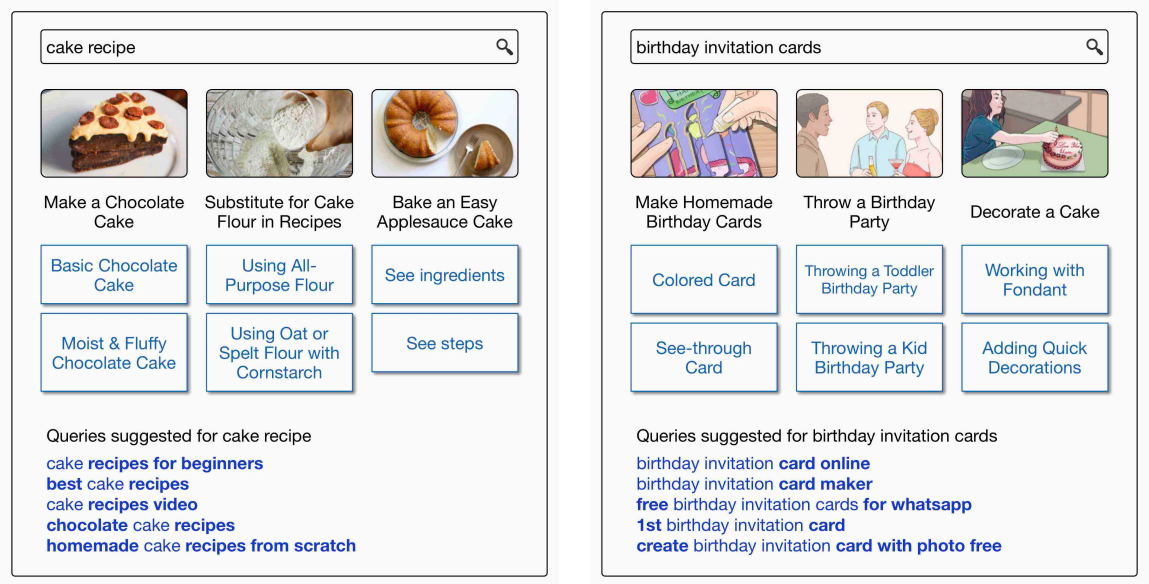


Fig. 1.5 Excerpt from an envisaged search engine with task-based support.

be suitable for recommendation. Further, if the user issues a third query "birthday invitation cards," then the recommended tasks would also include "Throw a birthday party." A possible set of results after considering these three queries is depicted in the right side of Fig. 1.5.

1.2 Contributions

In this section, we summarize the main contributions of this thesis. These contributions are grouped in two categories: (i) theoretical and empirical contributions, and (ii) resource contributions.

Theoretical and empirical contributions

C1 We conduct a systematic evaluation of methods for type-aware entity retrieval, in terms of three main dimensions that we identify: (i) type taxonomy, (ii) retrieval model, and (iii) type representation. We provide general recommendations on using type-based information for entity retrieval, after an extensive analysis.

C2 We revisit the problem of hierarchical target type identification, and present a state-of-the-art supervised learning method.

C3 We introduce a categorization scheme for entity-oriented search intents, that comprises four main categories, property, website, service, and other, and analyze the distributions of entity intent categories per entity type.

C4 We develop a method for constructing a knowledge base of entity-oriented search intents, which consists of four main components: refiner acquisition, refiner categorization, intent discovery, and knowledge base population.

C5 We introduce a probabilistic generative framework for task-based query suggestion, and conduct a principled estimation of each of the components in the architecture.

C6 We introduce the problem of query-based task recommendation, and establish a supervised learning method as suitable baseline. We also introduce the problem of mission-based task recommendation, and extend our approach to using a set of queries that all share the same underlying task.

Resource contributions

C7 **Test collection for Hierarchical Target Type Identification.**

This test collection contains a total of 649 query-type pairs, corresponding to 479 entity-bearing queries and relevance assessments for hierarchical target type identification task.

C8 **Test collection for entity-oriented search intent categorization.**

This dataset consists of 4,490 instances (type-level query patterns) labelled with one of the four intent categories in our categorization scheme.

C9 **IntentsKB: A knowledge base of entity-oriented search intents.**

Our knowledge base IntentsKB contains 31,724 intent profiles (i.e., unique entity-oriented search intents), and comprises a total of 155,967 quadruples of the shape *(intent ID, predicate, object, confidence)*.

C10 **Test collection of high-quality task-based query suggestions.**

A total of 12,790 query completion and 9,608 query refinement suggestions are labeled with binary relevance judgments.

C11 **Corpus of procedural search missions, and test collection for task recommendation.**

Our corpus consists of 1,378 search missions, 54 of which are procedural missions. A test collection of 59 queries and corresponding relevance assessments is built for evaluating query-based task recommendation. A similar test collection is obtained for mission-based task recommendation, in this case comprising task relevance assessments for 54 search missions.

1.3 Thesis Overview

We now present the outline of the thesis and the reading direction, which is depicted in Fig. 1.6. The Introduction and Background chapters are followed by five technical chapters (Chapters 3-7), and Conclusions are presented in Chapter 8.

Chapters 1 and 2 provide introduction and background for the rest of the thesis. Chapter 1 includes motivation, research outline, contributions, and origins of the thesis. Chapter 2 describes fundamental concepts of information retrieval pertinent to this work, and reviews foundational and state-of-the-art related literature.

Chapters 3 and 4 cover our work on type-aware entity retrieval. Chapter 3 presents and details the experimental results of evaluating the usage of type-based information for entity retrieval in terms of three problem dimensions. Chapter 4 addresses the problem of automatically identifying target entity types.

Chapter 5 describes a study of entity-oriented search intents, and presents a method for constructing a knowledge base of search intents.

Chapters 6 and 7 comprise our research on task-based search. Chapter 6 investigates generating query suggestions to support task-based search. Chapter 7 introduces and studies two problems related to task recommendation.

Chapter 8 revisits the research questions addressed in this thesis, and presents an outlook on future research.

The contents of the Background chapter are essential for understanding the technical chapters, yet some sections may be familiar to readers who are knowledgeable about information retrieval and semantic search. Chapter 3 is requisite for reading Chapter 4. Chapter 6 should be read before Chapter 7. The parts corresponding to the three grand themes can be read independently, i.e., entity type information for entity retrieval

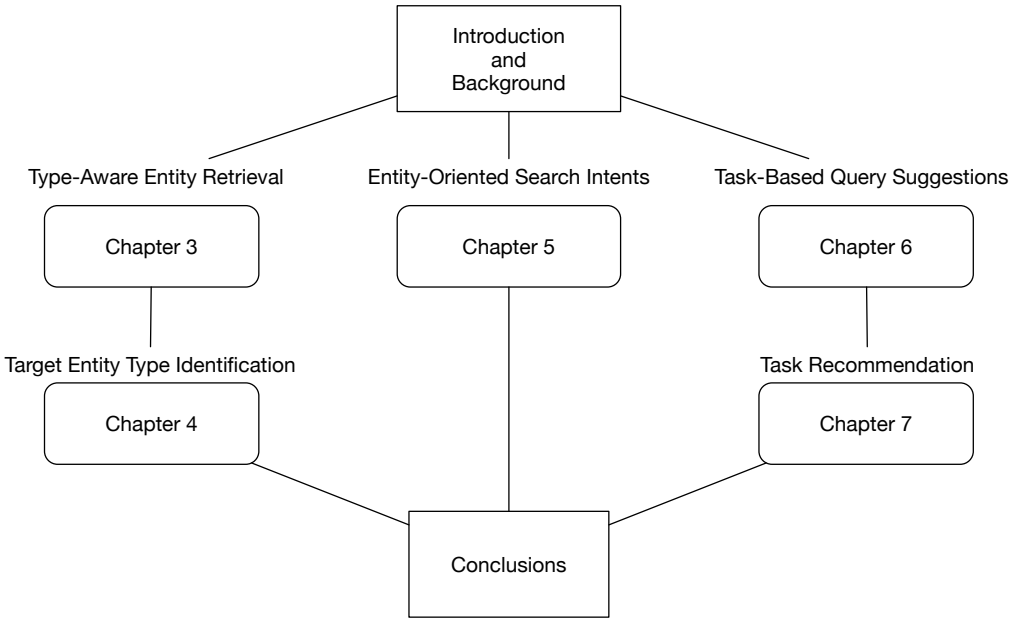


Fig. 1.6 Overview of the thesis organization.

(Chapters 3 and 4), entity-oriented search intents (Chapter 5), and task-based search (Chapters 6 and 7).

1.4 Origins

The content of this thesis is grounded in a number of publications [65, 68, 69, 67, 66, 63, 49]. Below we list these publications, and indicate to which research question and contribution they are related, alongside the chapter where they are included.

P1 Darío Garigliotti and Krisztian Balog. **On Type-Aware Entity Retrieval**. In Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR '17), pages 27—34, October 2017 ([65]).

[Related to RQ1 and contribution C1; included in Chapter 3.]

P2 Darío Garigliotti, Faegheh Hasibi, and Krisztian Balog. **Target Type Identification for Entity-Bearing Queries**. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17), pages 845–848, August 2017 ([68]).

[Related to RQ2 and contributions C2,7; included in Chapter 4.]

P3 Darío Garigliotti, Faegheh Hasibi, and Krisztian Balog. **Identifying and Exploiting Target Entity Type Information for Ad Hoc Entity Retrieval**. In Information Retrieval Journal, 22 (3), pages 285—323, August 2019 ([69]).

[Related to RQ1–2 and contributions C1–2,7; included in Chapters 3 and 4.]

P4 Darío Garigliotti and Krisztian Balog. **Towards an Understanding of Entity-Oriented Search Intents**. In Proceedings of the 40th European Conference on Information Retrieval (ECIR '18), pages 644–650, March 2018 ([67]).

[Related to RQ3 and contributions C3,8; included in Chapter 5.]

P5 Darío Garigliotti and Krisztian Balog. **IntentsKB: A Knowledge Base of Entity-Oriented Search Intents**. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18), pages 1659–1662, October 2018 ([66]).

[Related to RQ4 and contributions C4,9; included in Chapter 5.]

P6 Darío Garigliotti and Krisztian Balog. **Generating Query Suggestions to Support Task-Based Search**. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17), pages 1153–1156, August 2017 ([63]).

[Related to RQ5 and contribution C5; included in Chapter 6.]

P7 Heng Ding, Shuo Zhang, Darío Garigliotti, and Krisztian Balog. **Generating High-Quality Query Suggestion Candidates for Task-Based Search**. In Proceedings of the 40th European Conference on Information Retrieval (ECIR '18), pages 625–631, March 2018 ([49]).

[Related to RQ5 and contributions C5,10; included in Chapter 6.]

P8 Darío Garigliotti and Krisztian Balog. **Recommending Tasks based on Search Queries and Missions**. Submitted to a conference.

[Related to RQ6 and contributions C6,11; included in Chapter 7.]

Other publications contributed to advancing the thesis. These are not directly included, but rather correspond to either preliminary work that became later a publication listed above [62], or retrospective work that focused on a particular aspect of a previous publication [64], or research proposal that encompasses a complete vision of the thesis project plan [61]. Below we list these publications.

P9 Darío Garigliotti and Krisztian Balog. **The University of Stavanger at the TREC 2016 Tasks Track**. In Proceedings of the 25th Text REtrieval Conference (TREC '16), February 2017.

[Related to RQ5 and contribution C5.]

P10 Darío Garigliotti and Krisztian Balog. **Learning to Rank Target Types for Entity-Bearing Queries**. In Proceedings of LEARNER 2017, co-located with ICTIR '17, volume 2007 of CEUR Workshop Proceedings, November 2017.

[Related to RQ2 and contributions C2,7.]

P11 Darío Garigliotti. **A Semantic Search Approach to Task-Completion Engines**. In Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '18), page 1457, July 2018.

[Related to RQ1–5 and contributions C1–5,7–10.]

The following articles, although not directly related to this thesis, brought insights on the broader research area of semantic search.

P12 Faegheh Hasibi, Darío Garigliotti, Shuo Zhang, and Krisztian Balog. **Supervised Ranking of Triples for Type-Like Relations**. In WSDM Cup 2017, February 2017.

P13 Faegheh Hasibi, Krisztian Balog, Darío Garigliotti, and Shuo Zhang. **Nordlys: A Toolkit for Entity-Oriented and Semantic Search**. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17), pages 1289–1292, August 2017.

P14 Darío Garigliotti, Dyaa Albakour, Miguel Martinez-Alvarez, and Krisztian Balog. **Unsupervised Context Retrieval for Long-Tail Entities**. In Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR '19), pages 225–228, October 2019.

Chapter 2

Background

This chapter provides an overview of related literature, to serve as a basis for the rest of the technical chapters. We start by presenting fundamental concepts in information retrieval. We then review literature in the area of semantic search, in particular, entity-oriented search and entity-oriented search intents. We end this chapter describing related work in task-based search.

2.1 Information Retrieval

It is assumed that the reader is familiar with the main concepts in information retrieval (IR). Nevertheless, for the sake of making this thesis self-contained, we briefly discuss a number of concepts and techniques that are used throughout the thesis.

2.1.1 Document Retrieval

A central component of every IR system is a ranking algorithm that ranks documents with respect to a query based on a retrieval model. This section presents document retrieval models, which use term-based representations of queries and documents for ranking.

It is assumed that there is a collection from which documents are ranked by the model. For a given input query q, a relevance score is computed for each document d in the collection. The documents are then returned in decreasing order of this score. The general formula for the relevance score [41] is:

$$score(d, q) = \sum_{t \in q} w_{t,d} \cdot w_{t,q} \ , \tag{2.1}$$

where $w_{t,d}$ is the weight of the term t in the representation of d, and $w_{t,q}$ is the weight of t in q. Instead of taking into account all the terms in the vocabulary, it is enough to consider only the terms occurring in the query, which is usually shorter than any document. A simple scoring approach is to define $w_{t,q}$ as the raw frequency $f_{t,q}$, this is, the number of occurrences of term t in q. Then, $w_{t,d}$ can be defined to be 1 if and only if $f_{t,d} > 0$, otherwise to be 0. In this way, the score becomes the number of matching query terms in the document.

Vector Space Model

For term weighting, one can assign a weight in correspondence with the respective importance of the term. *Vector Space Model* [168] is a classical retrieval model, that is still used as it provides a simple approach for implementing ranked retrieval and term weighting. In the Vector Space Model, given a vocabulary of terms V, the documents and the query are represented by a $|V|$-dimensional vector of term weights:

$$d = (d_1, d_2, ..., d_{|V|}) \ ,$$
$$q = (q_1, q_2, ..., q_{|V|}) \ ,$$

where d_i and q_i are respectively the weight of the i-th term in V in d and in q. This model is said to be a bag-of-words model, since the vector representation does not consider the ordering of words in a document. As an illustration, both "The fox is slower than your turtle" and "The turtle is slower than your fox" have the same vectors. The intuition behind ranking documents with this model is that documents more similar to the query, or "nearer" in the vector space, are more likely to be relevant to the query. A standard way to compute the score is using the *cosine similarity* of the document and query vectors:

$$cosine(d, q) = \frac{\sum_t w_{t,d} \cdot w_{t,q}}{\sqrt{\sum_t w_{t,d}^2} \sqrt{\sum_t w_{t,q}^2}} \ . \tag{2.2}$$

If all the terms occurring in every document in the collection are sorted decreasingly by their raw frequency, Zipf's law indicates that each term among a few most frequent ones is very likely to occur, whereas the probability of a term occurrence rapidly becomes asymptotically zero for most of the terms. The intuition for term weighting that follows from this law is that (i) terms that occur often in a document should get high weights, and (ii) terms that occur in many documents should get low weights.

Term frequency and inverse document frequency are widely adopted to mathematically capture the phenomena (i) and (ii), respectively [41]. The *term frequency* (TF) $tf_{t,d}$ reflects the importance of the term t in the document d ($tf_{t,q}$ is analogous for a query q). Usual variants for defining $tf_{t,d}$ are: binary; raw frequency $tf_{t,d} = f_{t,d}$; and normalized term frequency:

$$tf_{t,d} = \frac{f_{t,d}}{\sum_i f_{t_i,d}} \ , \tag{2.3}$$

where the denominator is often denoted as $|d|$ and called the length of d. The *inverse document frequency* (IDF) idf_t reflects the importance of the term t in the collection of documents [41]. The more documents containing an occurrence of t, the less useful t is for retrieval since it is less discriminating between documents. IDF is defined as follows:

$$idf_t = log\frac{N}{n_t} \ , \tag{2.4}$$

where N is the total number of documents in the collection, and n_t is the number of documents where the term t occurs. The weight of a given term can then be obtained by combining TF and IDF multiplicatively:

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t \ . \tag{2.5}$$

Using this formula to compute the term weights, the cosine similarity defined in Eq. 2.2 becomes:

$$cosine(d,q) = \frac{\sum_t tfidf_{t,d} \cdot tfidf_{t,q}}{\sqrt{\sum_t tfidf_{t,d}^2 \ \sum_t tfidf_{t,q}^2}} \ . \tag{2.6}$$

BM25

BM25 [163] is a widely used retrieval model. It has its origins in the Probability Ranking Principle [162], a ranking criterion based on the probability of relevance of a document to a query. Term weighting in BM25 considers three principles: term frequency, inverse document frequency, and document length normalization. The BM25 ranking formula is defined as follows:

$$BM25(d,q) = \sum_{t \in q} \frac{f_{t,d} \cdot (1+k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgl})} \cdot idf_t \ , \tag{2.7}$$

where *avgl* is the average document length. The parameter $k_1 \in [0, +\infty)$ is the calibrating term frequency scaling. Setting $k_1 = 0$ leads to the binary term frequency model, and large values correspond to using raw term frequencies. This parameter is usually set between 1.2 and 2.0 [163]; a typical value is 1.2. The parameter $b \in [0, 1]$ is the document length normalization factor. Full length normalization is achieved by setting $b = 1$, whereas $b = 0$ makes no normalization at all. A good performance is obtained by taking a value for b in the range (0.5, 0.8) [163], and a typical value is 0.75. The soft document normalization $1 - b + b\frac{|d|}{avgl}$ takes into account the document length, with the intuition being that a document is more important if it is relatively long with respect to the average length. The left-side multiplier within the summation in $BM25(d, q)$, i.e., the whole fraction term, is regarded as the term saturation component. The general function $x/(k + x)$ asymptotically approaches 1 for some $k > 0$, hence the term saturation rationale, as the repetition of a term becomes less important after a saturation point.

Language Models

Language Models (LMs) are based on probabilities and processes for text generation. In this way, LMs have been effectively used for several applications in natural language processing, such as speech recognition, optical character handwriting recognition, machine translation, and completion prediction. The common intuition in these applications is that the more probable a term sequence, the more likely to be a correct output for the respective task. In information retrieval, language models constitute a sound framework for retrieval models [204], which show to perform effectively for a range of retrieval tasks [105].

A widely used instance of LMs is the *query likelihood* model [152]. The model ranks documents by $P(d|q)$, the likelihood of each document d of being relevant given a query q. Using Bayes' theorem, the probability can be rewritten as:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \propto P(q|d)P(d) \ . \tag{2.8}$$

The last transformation is due to the fact that the normalization factor $P(q)$ can be ignored in most cases. The document prior $P(d)$, i.e., the probability of the document being relevant to any query, is often assumed uniform across the document collection and so can also be ignored. The ranking model becomes then the query likelihood $P(q|d)$. Having represented each document with the unigram language model θ_d, the probability of the query q being generated by the document model θ_d is:

$$P(q|d) = \prod_{t \in q} P(t|\theta_d)^{f_{t,q}} \ , \tag{2.9}$$

where $P(t|\theta_d)$, the multinomial probability distribution over the vocabulary of terms, is raised to the number of occurrences of the term t in q. A simple setting $P(t|\theta_d) = tf_{t,d}/|d|$ would lead to possibly undesired situations in document ranking, since a probability of zero for a single term makes zero the whole likelihood $P(q|d)$. The probability of each term t in the whole document collection C, $P(t|C)$, is then incorporated to the model, using a so called smoothing technique. The Jelinek-Mercer smoothing method calculates $P(t|\theta_d)$ as follows:

$$P(t|\theta_d) = (1 - \lambda)P(t|d) + \lambda P(t|C) \ , \tag{2.10}$$

where the parameter $\lambda \in [0, 1]$ controls the smoothing component, and is typically set to be $\lambda = 0.1$. The empirical document model $P(t|d)$ can be estimated by $tf_{t,d}/|d|$, and the collection model can be estimated by maximum likelihood as:

$$P(t|C) = \frac{\sum_{d'} f_{t,d'}}{\sum_{d'} |d'|} \ . \tag{2.11}$$

An alternative smoothing technique is Dirichlet smoothing, which defines $P(t|\theta_d)$ as:

$$P(t|\theta_d) = \frac{tf_{t,d} + \mu P(t|C)}{|d| + \mu} \ , \tag{2.12}$$

where the parameter μ is set empirically for each ranking task; typical settings are $\mu = 1,500$ or $\mu = 2,000$. In Jelinek-Mercer smoothing, the same amount of smoothing is applied to all documents, whereas in Dirichlet smoothing, the smoothing is inversely proportional to the document length. Both smoothing methods are related by this setting:

$$(1 - \lambda) = \frac{|d|}{|d| + \mu} \ , \qquad \lambda = \frac{\mu}{|d| + \mu} \ .$$

Learning-to-Rank

Learning-to-Rank (LTR) approaches currently represent the state of the art in many retrieval tasks, including document retrieval. Learning-to-Rank consists of using a ranking model learnt by a supervised learning algorithm. The training data can incorporate a large number of potentially useful relevance signals from a variety of information sources. Yet, the performance of learning-to-rank methods heavily depends

on the availability of enough training data. In document retrieval, each training instance is represented as a feature vector of signals about the query and/or the document. Many different features can be used [120], but they mostly fall into one of these groups:

- Query features, which depend only on the query. Examples of query features are the length of the query in characters or in terms, or a binary feature indicating whether the first term of the query is a question word, like "what."
- Document features, which capture some aspect of the document's importance independently from the query. Typical features of this group are the "popularity" of the document (e.g., its number of views, or its number of incoming links), and the number of terms occurring in various document fields, like its title and content.
- Query-document features are often the most important features, as they capture the degree of matching between the query and the document. Examples of these features are the retrieval scores computed by unsupervised ranking methods.

A training set contains feature vectors for query-document pairs together with the relevance label indicating how relevant the document is for the query. Relevance can be binary or graded. A ranking model is then learnt, which in turn provides a ranking score for a given query-document pair. Just like for any machine learning task, the performance of learning largely depends on the choice of features. As a matter of fact, much of the research in IR focuses on the design of an effective feature set for the task at hand, whereas machine learning algorithms are applied out-of-the-box.

The learning-to-rank algorithms can be grouped into three categories:

- Pointwise approaches approximate the relevance of a single document for the given query, independently of the other candidates for the query. Standard regression or classification algorithms can be directly applied here for learning to rank. Two of the most popular pointwise approaches are Random Forests (RF) [27] and Gradient Boosted Regression Trees (GBRT) [58].
- Pairwise approaches consider a pair of documents at a time, and try to learn a binary classifier that outputs the relative order between the documents. GBRank [206] and RankNet [30] are examples of pairwise models.
- Listwise approaches aim to optimize the ordering of the entire result list. Algorithms included in this category are Coordinate Ascent [133], ListNet [34], and LambdaMART [196].

We now provide a high-level description of two pointwise learning-to-rank approaches used in this thesis.

The Random Forests algorithm [27] makes use of the Classification and Regression Tree (CART) procedure to obtain uncorrelated trees. In an iterative process, a CART is built from a bootstrap sample of instances in each iteration, using m random features. The mean prediction of the individual trees is taken as the final prediction. These two parameters, the number of trees (or iterations) and the maximum number m of features in each tree, are set empirically. The Random Forests algorithm is widely used as it often performs very well.

The Gradient Boosted Regression Trees algorithm [58] builds a set of trees using few features on each iteration, and takes the prediction average as the final score. Unlike random forests, in GBRT the trees are not randomly created, but each tree in every iteration is obtained to minimize the residual error of the previous iterations. Also here, the number of iterations and the number of features in each tree are free parameters to be set.

2.1.2 Evaluation

The most common way to evaluate an IR system is to compare the result of a given experiment against a standard test collection. A test collection is made of a set of queries and their corresponding relevance labels. For a number of IR tasks, such collections are usually provided by a benchmark campaign held at initiatives such as the Text REtrieval Conference (TREC), NTCIR and CLEF. Additionally, research teams often develop tailor-made test collections for novel problems. The relevance labels in these collections are commonly provided by expert human annotators or via crowdsourcing annotations.

The performance of a system is measured in terms of different evaluation metrics. Two main categories are set-based measures and rank-based measures.

Set-based measures are the most basic measures to evaluate an IR method. As their name suggests, they do not consider the order of the results provided by the system. Rather, given a query, these measures simply take into account two sets of documents: (i) REL, the set of documents known to be relevant for the query in the ground truth, and (ii) RET, the set of documents retrieved by the system. The two set-based measures most commonly used are *precision* and *recall*, denoted respectively by P and R, which are defined as follows:

$$P = \frac{REL \cap RET}{RET} \ , \qquad R = \frac{REL \cap RET}{REL} \ .$$

The second category of evaluation metrics are rank-based measures. These take into account the order of the results in a ranking, and usually are measured at different positions of the ranked list. Precision and recall at a given rank position K, denoted respectively by $P@K$ and $R@K$, consider only the set of the first K ranked results. Precision at different levels of recall can be captured in a single measure using *Average Precision* (AP), which is defined as:

$$AP = \frac{1}{|REL|} \sum_{\substack{i \in \{1,\ldots,n\} \\ d_i \in REL}} P@i \ , \tag{2.13}$$

where only the relevant documents $\{d_i\}$ contribute to the sum. *Mean Average Precision* (MAP) is obtained by averaging the AP per query over all queries in a set of interest. MAP is very useful to report the overall system performance for many cases, in particular, when the relevance judgments are binary.

Beyond these measures, other evaluation metrics focus on the top documents. This is in line with the intuitive observation that users tend to look at only the top part of a ranked result list to find relevant items. Moreover, some search tasks may have a single relevant document. Examples of such tasks are navigational search, where the user looks for a particular website, and question answering. In these cases, recall is not an appropriate measure. Instead, $P@K$ for typical rank positions, e.g., $K \in 5, 10, 20$ are easy to compute and to understand. Yet, they are not informative about rank positions less than K. *Reciprocal Rank* (RR) is defined as the reciprocal of the rank position at which the first relevant document is retrieved. *Mean Reciprocal Rank* (MRR) is the average of the reciprocal ranks over a set of queries. This metric is very sensitive to the rank position. MRR has been widely used in IR, yet some criticism has been raised recently, regarding its usage [59].

When non-binary, i.e., graded relevance judgments are used, a popular measure is the *Normalized DCG* (NDCG), that is, the *Discounted Cumulative Gain* (DCG) [90] normalized with respect to the ideal DCG. Discounted Cumulative Gain is built around two assumptions: (i) highly relevant documents are more useful than marginally relevant ones, and (ii) the lower the ranked position of a relevant document, the less likely to be examined, hence the less useful it is for the user. A main concept in this measure is the one of usefulness, or "gain," of a document, representing how much information is gained when a user examines the document. Given that a user starts

examining documents and stops at rank p, the cumulative information gain can be computed as:

$$CG_p = \sum_{i \in \{1,\ldots,p\}} r_i \ , \tag{2.14}$$

where r_i is the gain (or the relevance label) of the i-th retrieved document. However, CG is not appropriate for capturing the rank position of each retrieved document. The gain is then accumulated starting at the top of the ranking and may be reduced, or "discounted," at lower ranks, as a penalty. The discounted cumulative gain is calculated as follows:

$$DCG_p = r_1 + \sum_{i \in \{2,\ldots,p\}} \frac{r_i}{log_2(i+1)} \ . \tag{2.15}$$

The gain at the top ranked document is not discounted, since it is assumed that the user always examines this document. The DCG values are often normalized into the range [0,1] dividing the DCG at each rank by the DCG value for the ideal ranking, referred to as IDCG:

$$NDCG_p = \frac{DCG_p}{IDCG_p} \ . \tag{2.16}$$

This normalization makes averaging easier for queries with different numbers of relevant documents. NDCG values are usually averaged across a set of queries at specific rank values. Typical rank cut-offs for which NDCG is reported in many applications are 5, 10, or 20.

The *Expected Reciprocal Rank* (ERR) [36] generalizes the reciprocal rank measure to the case of graded relevance. ERR is defined as follows:

$$ERR = \sum_{i \in \{1,\ldots,n\}} \frac{1}{i} S(i) \ , \tag{2.17}$$

where n is the number of documents in the ranking, and $S(i)$ is the probability that a user stops from scanning documents in the ranking after examining the document at position i. The assumption in ERR is that S is defined in terms of the relevance grade of the document under consideration. Some criticism has been raised recently regarding the usage of this measure [59].

Keyword queries, predominant in web search, are often ambiguous and have more than one interpretation [4]. A practice widely adopted by modern search systems is to produce a set of diversified results. Result diversification is an important research

problem. Accordingly, metrics such as α-NDCG [38] have been proposed for evaluating the diversity of search results. These metrics assume that there is a predefined set of topics or categories, and that the user information needs can be modeled at the topical level of this set. Both, queries and documents, may belong to one or more categories.

Several metrics for evaluating diversity present certain shortcomings: they do not take into account how well a document satisfies a category, and they do not consider the relative importance of different categories. *Intent-Aware Expected Reciprocal Rank* (ERR-IA) [1] is a metric proposed to address those shortcomings. This measure assumes that the probability $T(c|q)$ of a given query q belonging to a given category c is known. Furthermore, labels indicating topics assigned to documents are known. The metric also makes use of the idea of the quality value of a document d for a query q when the intended category c is considered. Given the category c, the quality value of a document d for a query q is the relevance label of d, except when d does not have c among its known category labels, in which case the quality value of d for q is made zero. $ERR(c)$ denotes ERR where S in Eq. 2.17 is defined not in terms of the document relevance values but in terms of their respective quality values for each query, given the category c. Then, *ERR-IA* can be defined as follows:

$$ERR\text{-}IA = \sum_{c} T(c|q) ERR(c) \ . \tag{2.18}$$

2.2 Semantic Search

As described in Chapter 1, semantic search, or "search by meaning," encompasses multiple techniques under the main goal of understanding the query, and so providing a better search experience. Users seem to expect current web search engines to be able to provide answers to their varied information needs, from checking the weather, to ordering a taxi, to organizing a party. The traditional ten blue links, where only documents relevant to the query were ranked, has been enhanced with meaningful information objects among the results, in particular, entities, and entity-oriented widgets and displays. The increased availability of structured data sparkled the evolution of web search engines into "answer engines [7]." This section presents a review of entity-oriented search, and then discusses previous work on understanding query intents.

2.2.1 Entities and Knowledge Bases

Entities, such as persons, organizations, and locations, are natural units for organizing information [7]. Entities can provide not only more focused responses, but often immediate answers [134]. Many information needs in web search look for specific entities [153].

A knowledge repository contains a collection of entities in a certain representation. Wikipedia is arguably the most widely known knowledge repository, where an entity is represented as a semi-structured article. That is, most of the document content is raw (or unstructured) text, with few structured information in the infoboxes presented as entity cards within the articles. Its collaborative project allows a large number of contributors to combine their efforts, daily used by millions of users around the world as their trustable reference in many areas of knowledge. Wikipedia is also widely supported and used by the research community.

Wikipedia has been created for human consumption. A *knowledge base* (KB), instead, contains structured knowledge, often extracted from free text, intended to be accessible to machines. Contrasting with the unstructured nature of textual documents, this structured format represents knowledge usually by Resource Description Framework (RDF) triples of the shape *(subject, predicate, object)*. Each of these triples declares a fact about the subject (an entity) by relating it with an object (a literal or another entity). Ultimately, the approaches for extracting structured knowledge from textual content aim to produce a potentially machine-readable representation of information. The alternative denomination of KBs as *knowledge graphs* (KGs) emphasizes the relationships between entities.

A number of general-purpose knowledge bases have been developed along the recent years. DBpedia [107] is widely used by researchers in areas such as Semantic Web and Information Retrieval. Developed by processing the semi-structured data from Wikipedia, it is a hub for multiple KBs to redirect to its large repository of entities. Other cross-domain KBs include YAGO [176], Freebase [24], and Wikidata [192, 179]. Alongside these ones, there also exist many purpose-specific knowledge bases, for example, GeoNames, WordNet, and DBLP, usually created for projects in specific domains. All these KBs are repositories of open data, freely available to everyone.

The Linked Open Data cloud constitutes an initiative that allows to link resources among these KBs.[1] Apart from these open KBs, there exist proprietary initiatives to collect and leverage large amounts of structured data. In particular, tech giants

[1] https://lod-cloud.net/

such as Google [51], Bing and Facebook have all developed their own variation of a knowledge graph.

2.2.2 Entity-oriented Search

Given a representation of structured knowledge centered around entities, it is of key importance to be able to uniquely recognize the occurrences of entities for building these representations, and to retrieve relevant entities for entity-oriented search queries. These two central problems in entity-oriented search are discussed in this section.

Recognizing entity mentions in unstructured text is a core task in the construction of knowledge bases, where facts about entities are expected to be collected and represented in a structured fashion. In particular, this task is very important in the area of Information Extraction (IE) when aiming to automatically build knowledge bases, as it is the goal of projects like NELL [35] and OpenIE [14, 123]. Moreover, in many other scenarios it is convenient to identify entity occurrences with high accuracy. An example is the case of an *online reputation management* [3] service, which monitors through news streams the stock performance of a company or a scandal around a celebrity. The task of *named entity recognition* (NER) consists of identifying the occurrences of name entities, and also assigning a high-level semantic class label among Person, Organization, or Location. Another task, *named entity disambiguation* (NED), addresses the problem of linking a given occurrence with the correct entry in a knowledge base. The task of *entity linking* (EL) comprises both NER and NED, in the full problem of recognizing and disambiguating an entity occurring in unstructured text. These tasks are fundamental entity-centered problems when processing a text document [161, 186].

Entity Retrieval

Entity retrieval is the task of obtaining a ranked list of entities relevant to a search query. Many queries are entity-oriented, that is, queries whose expected result is an entity or a list of entities, or queries that contain entity mentions. Indeed, a large portion of information needs in web search look for specific entities [153]. Ranking entities in response to a search query is an important problem. The INEX 2007–2009 Entity Retrieval track [43, 45, 48] is the first benchmark campaign in studying entity retrieval, using Wikipedia as entity repository of reference. Earlier, the expert finding problem addressed the retrieval of people, considering their expertise in an enterprise context as relevance criteria [8].

A research item of particular interest in entity retrieval is the representation of entities. Hence, a considerable portion of the work in this area deals with how to obtain a term-based representation of entities, which then allows to rank them much like documents, using traditional document retrieval approaches. In particular, recent established methods for entity retrieval adopt a fielded entity representation, and utilize fielded variants of document retrieval methods [164, 143, 102, 101, 207].

Several tasks have been studied in the field of entity retrieval. *Ad hoc entity ranking* is the task of returning a ranked list of entities relevant to the input keyword query. This problem is introduced in the INEX 2007–2009 Entity Retrieval track [43, 45, 48]. Another task was formulated in the same benchmark campaign: *list search*, which further assumes that sought results are semantically related. For example, the results expected for the query "US presidents since 1960" are all related by the condition of being US presidents who were in charge since 1960. These semantic relationships may be specified in the input, alongside the keyword query, either with a set of types of entities sought by a query, or a small set of example entities. *Related entity finding*, a special case of list search, requires the result entities to be of a specific type and stand in a particular relation with a given input entity [10]. As an illustration, given the input query "airlines that currently use Boeing 747 planes," the entities to be returned must be of type *Airlines*, and related with the input entity Boeing 747. This problem was featured in the TREC 2009-2011 Entity track [10, 13]. Finally, answers to many questions in *question answering* are specific entities [115]. For example, the expected result for the query "What is the capital of Norway?" is the entity Oslo. The DBpedia Entity test collection, introduced by Balog and Neumayer [12], contains queries from several benchmarking campaigns, ranging from keyword queries to natural language questions, with relevant results mapped to DBpedia. Hasibi et al. [80] enhanced the collection into the DBpedia Entity v2 collection, which adopts a more recent DBpedia version and incorporates judgements for new entities, as well as it uses graded relevance judgements for all queries.

Entity Types and Target Entity Types

As mentioned above, some entity retrieval tasks involved entity type information. A characteristic property of an entity is its type. For example, the entity Oslo is of type *City*. Other types for this entity are *Settlement*, a more general type than *City*, and *Norwegian city*, more specific than *City*. An entity is assigned one or more types from a type system of reference. Types in a type system are usually arranged in hierarchical

relationships, which makes such a system to be known also as type taxonomy or type hierarchy.

Types such as *City* and *Norwegian city* are intuitive semantic classes that group entities. It is possible to think in many other types for a given entity. For example, Oslo is also a *European city*, and a *Norwegian city with more than 100,000 inhabitants*. Yet, in an actual scenario leveraging entity-oriented structured knowledge, the types available for an entity are usually the ones provided by the type taxonomy of reference. This is, depending on the type system that is used, a type that exists in a given taxonomy might not be available in another. As an example, the DBpedia Ontology is a rather small type hierarchy, manually curated, of a few more than 700 types in its current version. Using this taxonomy as reference, it is only possible to indicate that Oslo is a *City*, but not a *Norwegian city*, since this more specific type does not exist in the DBpedia Ontology. Instead, the Wikipedia category graph contains a category *Cities and towns in Norway*. This very large graph of categories records hierarchical relationships, but is not strictly a taxonomy. Other type hierarchies used by the research community include the YAGO taxonomy and Freebase types.

Entity type information in an knowledge base is often incomplete, imperfect, or missing altogether for some entities. In particular, new entities emerging on a daily basis also need to be mapped to one or more types of the underlying type system. Regarding the challenges of the entity typing problem, Gangemi et al. [60] distinguish between *extensional* coverage, i.e., the number of typed resources, and *intensional* coverage, i.e., conceptual completeness. Furthermore, their approach makes use of the natural language definition of an entity in Wikipedia, inferring types from graph patterns over logical interpretations of the entity definitions. Similar approaches are used in the related task of fine-grained entity typing in context. Lin et al. [112] exploit entity definitions to map entities into Freebase types by analyzing n-grams in the textual relations around entity mentions. Nakashole et al. [140] address typing emerging entities from news streams and social media. Fossati et al. [56] address entity typing by leveraging linguistic hypotheses about the syntactic patterns in Wikipedia category labels. Yaghoobzadeh et al. [197] use multi-instance, multi-label typing algorithms over KB data and annotated contexts of entities in a corpus. Kliegr and Zamazal [103] predict the entity type by linearly combining the output distributions from several techniques: (i) string matching and statistical inference on an external hypernyms dataset extracted from Wikipedia, and (ii) hierarchies of classifiers trained on entity short abstracts and on Wikipedia categories. Corpus-level entity typing is also used for knowledge base completion. A multilayer perceptron approach has been proposed

using word embeddings [198]. SDType [149, 150] utilizes links between instances in a KB to infer entity typing by using a weighted voting approach. The main assumption is that some relationships between entities only occur with particular entity types. As an example, from the triple (x, `dbo:location`, y), it could be inferred with high confidence that y is of type *Place*. Since DBpedia 3.9, the type assignments obtained by SDType, available for a large subset of entities, are distributed with DBpedia. Regarding the task of detecting and typing *emerging entities*, having fine-grained types for new entities is of particular importance for informative knowledge [112, 140]. For example, the type *Guitar player* is more informative than *Person*, and similarly is *Jazz music festival* than *Event*. These fine-grained types are expected to boost the grouping of emerging entities, as well as to strengthen the extraction of relational facts about entities.

Just as types are a typical property of entities, types can also be assigned to a query. The types of entities expected to be returned for a query are known as the *target entity types* of the query. As an illustration, some of the target entity types of the query "US presidents since 1960" may be *Person*, *Politician*, *President*, or *US President*. The INEX Entity Ranking track [48] and the TREC Entity track [13] both featured scenarios where target types are provided by the user. In a realistic setting, target entity types are not provided, but rather need to be automatically identified based on the keyword query. Vallet and Zaragoza [185] introduced this very task as the *entity type ranking* problem, this is, to rank target entity types for a given input query. This task is later referred to also as *target entity type identification* problem. They establish a baseline method, which extracts entity mentions from the set of top relevant passages, then considers the types associated with the top-ranked entities using various weighting functions. Methods that use entities as a bridge to reach entity types from a query are known as *entity-centric* methods. Kaptein et al. [98] similarly use a simple entity-centric model. Balog and Neumayer [11] address a hierarchical version of the task, using the DBpedia Ontology and language modeling techniques. The hierarchical target type identification (HTTI) task is defined as the problem of finding the main target entity types of a query, from a type taxonomy, such that (i) these correspond to the most specific categories of entities that are relevant to the query, and (ii) main types cannot be on the same branch in the taxonomy. The authors further propose two approaches. One of them uses an entity-centric strategy. The other builds a textual type representation by concatenating the descriptions of all the entities assigned to each type.

Type Information in Entity Retrieval

Utilizing target entity type information in ad hoc entity retrieval is a multifaceted research problem. As mentioned, type information involves hierarchical relationships in the type system, or the several possible choices of a reference type taxonomy, and these dimensions of type-based information are to be taken into account when addressing type-aware entity retrieval.

The choice of a particular type taxonomy is mainly informed by the problem setting, for example, DBpedia Ontology is a small yet curated taxonomy, whereas Wikipedia provides a larger number of categories. The most common type system used in prior work is Wikipedia categories [47, 9, 97, 158, 29]. This is in part due to historical reasons, since Wikipedia was the underlying type system used at the INEX Entity Ranking track, where type information was first exploited. Further choices include the DBpedia Ontology [11, 181], YAGO types [47, 169, 181, 140], Freebase [112], and schema.org [181].

Early work represented type information as a separate field in a fielded entity model [208]. Demartini et al. [47] additionally expand type information using the underlying hierarchy. Since then, several approaches have attempted to expand the set of target types based on the hierarchical structure of the type system [151, 9, 29, 11, 181]. In later works, types are typically incorporated into the retrieval method by combining term-based similarity with a separate type-based similarity component. This combination may be done (i) using a linear interpolation [9, 97, 151], where the contribution of each term- and type-based components is controlled, or (ii) in a multiplicative manner, where the type-based component essentially serves as a filter over the ranked results of the term-based part [29]. Different approaches have been proposed to measure the type-based similarity. Vercoustre et al. [187] define it in terms of lexical type label similarity. Kaptein and Kamps [96] take into account the descriptions of entities to which a given type is assigned. Weerkamp et al. [194] measure the overlap ratio of type sets. Zhu et al. [208], Demartini et al. [46] model types added as a separated field in multi-field retrieval. Balog et al. [9] employ a probabilistic approach that represents entity type information as multinomial probability distributions. Regarding the representations of type information, target entity types are commonly considered in two main ways. In one hand, the target types are seen simply as a set where every assigned type has the same importance [151, 47, 158, 97]. In the other hand, target types constitute a bag, this is, a set weighted by a probability function, defining a distribution of target types [185, 9, 169].

2.2.3 Understanding Query Intents

Query understanding is the process of "identifying the underlying intent of the queries" [40]. Understanding the query intent constitutes a key factor to improve the search experience [113]. Different general approaches have been proposed, according to a particular interpretation of what it means to understand the query intents. One kind of approaches aim to identify what the query is about, using topical classification, usually performed with a model trained via supervised learning [170, 110, 85, 33]. In the case of entity-oriented queries, a similar line of approaches understands the query by identifying the target entity types, as previously detailed [185, 11]. Another kind of approaches perform query segmentation, this is, to divide the query into meaningful segments, usually lexical or syntactic phrases [160, 93, 17, 178, 86, 77]. Finally, yet another branch of query understanding consists of identifying the entity occurrences in queries, and classifying them into predefined entity types [78, 79].

Several research works aim to propose or discover query taxonomies. Broder [28] studies query intents according to how people search, and for what reason. He proposes a scheme to classify every given query into one of three possible intent categories: navigational (to reach a specific website), informational (learn by viewing or reading web pages) or transactional (obtain a resource different from information). This seminal scheme for understanding query intents has been extended by further refinements as more online search activities are identified. Rose and Levinson [165] focus on why a search is made, and propose a scheme of eleven finer-grained intents. Specifically, the authors further divide the informational intent category into subcategories (including to learn something in particular about a topic by an either unambiguous or open-ended question, to be told anything about a topic, to find a location, to get advice, and to get a list of suggested web sites). They also subdivide the transactional category into four distintive resource acquisitions: downloading, being entertained, interacting, and getting a product possibly by buying or renting it. Jansen et al. [89] present an approach to classify the intent behind a given query into one of the three main categories. Yin and Shah [201] exploit search logs to build taxonomies of query intents for a set of entities.

Previous work has identified high-level patterns from web search queries, as well as has aimed to build a set of all possible query intents for a given entity, or at least most of them. Within similar approaches of query segmentation, these works often assume that the query consists of an entity name, complemented with context terms, or *refiners*, to express the underlying intent of the user [153]. For example, according to Lin et al. [113], a query can be classified as an entity, an entity plus a refiner (e.g.,

"emma stone 2017"), a category, a category plus a refiner (e.g., *"doctors in barcelona"*), a website, or other sort of query. Such classification relies merely on lexico-syntactic forms and lacks a more semantically-grounded distinction. The authors mine a query log to obtain the distribution of the most frequent refiners per entity. Reinanda et al. [159] explore entity aspects in user interaction log data. There, an entity aspect roughly corresponds to a section heading in a Wikipedia article. The authors obtain clusters of entity aspects and also of query refiners from search logs. With these, they address the tasks of ranking the intents for a given entity, and recommending aspects.

In some research lines, the entity-oriented intents under the transactional category are referred to as *actions*. Lin et al. [113] propose the model of active objects as a representation of an entity associated with a ranked list of potential actions. For example, given a particular flashlight as the given entity, some of the actions found suitable for the entity are "read reviews," "find locally," and "buy." The authors consider a fixed set of actions, for individual entities, where the actions are simple surface forms mined from query logs. The authors also study a generative model able to suggest the intent for a given entity, by "generating" a query like the observed ones.

The *action mining* task at the NTCIR Actionable Knowledge Graph track [21] addresses the problem of ranking potential actions for a given entity of a given type. There, an action comprises of a verb and possibly an object or modifier. In this track, an action is required to be expressed as a verbal phrase, which is not the case for most of the transactional intents. As an example of this last item, *"hilton taxi"* very likely expresses the intent of getting a taxi to the hotel, where the refiner "taxi" is not a verbal phrase.

2.3 Task-based Search

Search is often performed in the context of some larger underlying task [99], involving a nontrivial sequence of steps to be completed. A canonical example is planning a trip to Barcelona. In order to complete her task, the user typically needs to issue a query for booking a flight to Barcelona, another query for booking a hotel, another one to check the things to do in the city. Some of these steps may be already simplified, for example, the interface for booking the hotel might also include suitable offers for booking a flight, or provide guides for visiting points of interest. Even in those cases, the user has to deal with the cognitive workload of comparing offers, realizing that she might also need to rent a car, aggregating costs from multiple services, remembering that she would like to ask for special hotel amenities, or issuing new searches when

exploring alternative dates. Current web search engines are far from supporting task completion. Hence, there is a growing stream of research aimed at making search engines more aware of complex search tasks (i.e., recognizing what task the user is trying to accomplish) and customizing the search experience appropriately.

A theme in complex search is supporting exploratory search, where users pursue an information goal to learn or discover more about a given topic [183, 6]. Log-based studies have been a main area of focus, including the identification of tasks and segmentation of search queries into tasks [118, 82] and mining task-based search sessions in order to understand query reformulations [91] or search trails [195], as well as extracting hierarchies of tasks and subtasks [129] and predicting task satisfaction [127].

As described in the previous section, the high-level categorization of search behavior by Broder [28], classifying the queries in navigational, informational or transactional, has been subsequently extended as more online search activities are identified [165, 89, 201]. Of particular interest are multi-step search tasks [81], which encompass the behavior where complex tasks, involving multiple aspects, are aimed to be fulfilled.

A large body of work targets the detection of search sessions in query logs, with the ultimate aim of identifying search patterns in user behavior and of providing additional support upon them. In the field of query log analysis, a *search session* is usually defined as a sequence of queries where the elapsed time between two consecutive queries is not larger than a delta value, which ranges from 30 minutes [83, 154] to 90 minutes [74]. Beyond session detection, focus was also placed on the challenge of understanding the task-based user interaction with search engines. Here, "task" refers to the overall goal ("work task") of the user that motivated her to search in the first place. A notion of *search mission* is established by Jones and Klinkner [92]. They introduce the concept of hierarchical search, instituting the idea of a search mission as a set of topically-related queries, not necessarily consecutive, composed by simpler search goals. Their work also discusses a method for detecting whether a given query pair belongs to the same goal. Their concept of a search mission is the one assumed in this thesis, to emphasize the coherent nature of search queries motivated by a single underlying goal, different from other lines of work that focus on multi-tasking behavior across sessions.

Several works have extended problems on search behavior in query logs. Mei et al. [131] propose a method for finding user search sequences at multiple levels of granularity within the hierarchical search approach. Moreover, Donato et al. [50] address on-the-fly identification of search tasks, i.e., as the user interacts with an engine. Intent-aware query similarity is introduced by Guo et al. [72], aiming to capture the search intent behind a query and using these underlying tasks for recommending related queries.

Kelly et al. [99] propose a supervised model to analyze user search behavior spanning multiple sessions. They address two problems: identifying related queries for a query in previous sessions by that user, and predicting whether the user will return to a given multi-query task. The problem of search task discovery within a user session is introduced in the work by Lucchese et al. [117], where the authors also propose a method based on graph query representation. Hagen et al. [74] extend task detection to the cross-session modality, using a cascading approach, to account for the user behavior of carrying on a search task across multiple sessions. Further approaches in cross-session search task detection include modeling intrinsic and extrinsic diversity [157], temporal closeness [108], topic membership [109], and embedded trees in the query similarity graph [193]. Mehrotra et al. [125] find a distribution of search sessions with respect to the number of tasks, as well as propose a categorization of users in terms of their multi-tasking behaviour. Lucchese et al. [116] propose a framework for finding search tasks combining a Learning-to-Rank-based query similarity and graph-based query clustering. Verma and Yilmaz [189] leverage entities occurring in queries to build task dictionaries of terms and use them for extracting tasks from a query log. Furthermore, the same authors also study the usage of entity type information for task extraction [190]. Rather than the usual flat representation of tasks, methods to obtain a hierarchy of tasks and their corresponding subtasks have been studied [128, 126, 129]. Context information from query logs is used to learn a continuous distributional representation in the form of a task embedding [130]. Beyond task extraction, Mehrotra et al. [127] extract interaction sequences of user activity with the search engine results, and learn a model to predict user satisfaction at the level of individual queries.

2.3.1 Suggesting Queries to Support Task-Based Search

As mentioned before, completing a complex task often requires the user to issue multiple queries. Yet, focus is often on providing the best results for a single query, and usually it is ignored the task that lead the user to issue her query. The three editions of the TREC Tasks track (2015-2017) [200, 188, 95] formulated two main problems. The first, *task understanding*, consists of providing a complete coverage of subtasks for an initial query, by the means of a ranked list of keyphrases or query suggestions, while avoiding redundancy. The second, *task completion*, requires to return documents for helping the user complete the task behind her initial query, aiming to assess the usefulness of such a task completion system.

Query suggestions, recommending a list of related queries to an initial user input, are an integral part of modern search engines [144]. Most of the previous work in query

suggestions utilizes large-scale query logs. For example, Craswell and Szummer [39] perform a random walk on a query-click graph. Boldi et al. [23] model the query flow in user search sessions via chains of queries. Scenarios in the absence of query logs have also been addressed [104, 18].

The teams participating in the TREC Tasks track have relied on several information sources when addressing the task understanding task. Query suggestions from API suggestion services, provided by commercial web search engines like Google and Bing, have been utilized by teams across the three editions of the track. Another commonly used source for extracting keyphrase candidates are the documents corresponding to the search results from commercial web search engines, and the documents in the ClueWeb12 corpus. Bennett and White [16] make use of the anchor text graph, which is also part of the sources in the system developed by Hagen et al. [76]. In this strategy, they consider the documents anchored by keyphrases similar to the initial query, and then take as candidates the anchors that link to these documents. The Webis team [75, 76] also leverages the AOL query log divided into search sessions and missions, taking as candidates all sessions containing a query whose cosine similarity with the initial query is above a certain threshold. Other information sources taken into account by the different teams are query auto-completions from Google Autocomplete, retrieved topics from Freebase and Wikidata that are similar to the entity annotated in the initial query, and WikiHow articles. The approaches used to generate query suggestions for supporting task-based search combine their respective information sources in a number of ways. Alsulmi et al. [2] propose a clustering step to eliminate redundancy among the candidates and so favor the diversity of suggestions. Each keyphrase is then ranked with respect to its cluster. In the approaches developed by Hagen et al. [75, 76], each information source is a module with a corresponding score, determined by manual testing. Then, each keyphrase candidate is assigned the sum of the scores across all the modules that it comes from. Garigliotti and Balog [62] propose a probabilistic generative model.

2.3.2 Procedural Knowledge

As mentioned before, the increased availability of structured data published in knowledge bases was the pivotal component that sparked the evolution of a broad range of search and recommendation services on the Web [7]. However, supporting task-based search does not benefit necessarily from descriptive or declarative KBs, such as Wikipedia, DBpedia, or the Google Knowledge Graph. Indeed, they lack sufficient coverage of *procedural* knowledge. Procedural knowledge [70] is the knowledge involved

in accomplishing of a task. Unlike descriptive knowledge represented in knowledge repositories and bases, procedural knowledge, usually acquired by experience, is regarded rather implicit and hard to be shared. A few *procedural knowledge bases*, instead, are specialized in semi-structured knowledge [145, 37], that has been extracted from corpora of how-to guides collaboratively created in projects like WikiHow or eHow.

Jung et al. [94] automatically create a situation ontology from how-to instructions through a model of daily situational knowledge with goals, action sequences, and contextual items. They also envisage a schema mapping actions to actual service providers.

Pareti et al. [145] identify properties that define community-centric tasks, that is, tasks with a strong social aspect, for example, the task of collaborating to develop a project. They then propose a framework to represent the semantic structure of procedural knowledge in web communities. The proposed representation operates distributed across different KBs, and comprises a process ontology intended to express in a structured fashion the tasks, sub-tasks, requirements, and execution status.

A very different approach for constructing a procedural KB is proposed by Chu et al. [37]. They normalize the how-to tasks from WikiHow into a hierarchical taxonomy that also represents the temporal order of sub-tasks, as well as the attributes of the involved items. Their method relies on hierarchical agglomerative clustering of task frames obtained by open-domain semantic role labeling.

Chapter 3

Utilizing Entity Type Information

Today, the practice of returning entities from a knowledge base in response to search queries has become widespread. Entities can improve the user experience throughout the entire search process, by enabling techniques of query assistance, content understanding, result presentation, and contextual recommendations [7]. Major web search engines also shaped users' expectations about search applications; the single-search-box paradigm has become widespread, and ordinary users have little incentive (or knowledge) to formulate structured queries. The task of *ad hoc entity retrieval* [153] corresponds to this setting: returning a ranked list of entities from a knowledge base in response to a keyword user query.

One of the unique characteristics of entity retrieval that distinguishes it from document retrieval is that entities are typed. *Entity types* (or *types* for short) are semantic categories that group together entities with similar properties. Types are typically organized in a hierarchy, which we will refer to as *type taxonomy* hereinafter. Each entity in the knowledge base can be associated with (i.e., is an *instance of*) one or more types. For example, using the DBpedia Ontology, the type of the entity Albert Einstein is *Scientist*; according to Wikipedia's category system, that entity belongs to the types *Theoretical physicists* and *People with acquired Swiss citizenship*, among others. By identifying the types of entities sought by a query (*target types*, from now on), one can use this information to improve entity retrieval performance [208, 47, 151, 29, 9, 158, 97]; see Fig. 3.1 for an illustration. Moreover, in an envisaged search interface, direct answers such as widgets and verticals could take into account the types of entities targeted by the search query. The main high-level research question we are concerned with in this chapter is the following: *How can one exploit entity type information to improve ad hoc entity retrieval?*

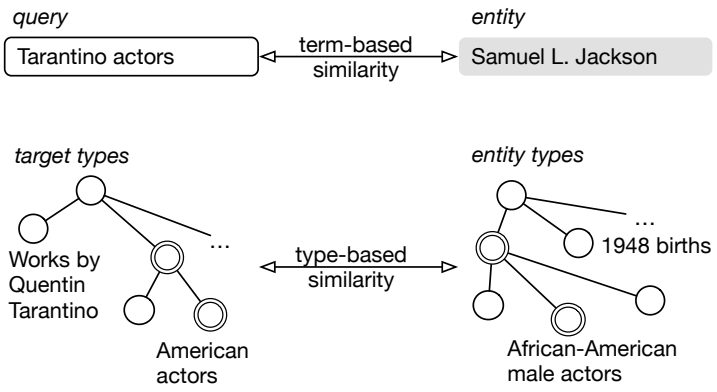


Fig. 3.1 Illustration of type-aware entity retrieval, where the target types sought by a query (left) are matched against the types that are assigned to the given entity in the knowledge base (right).

The concept of entity types, while seemingly straightforward, turns out to be a multifaceted research problem that has not yet been thoroughly investigated in the literature. Most of the research related with the usage of type information has been conducted in the context of the INEX Entity Ranking track [48]. There, it is assumed that the user complements the keyword query with one or more target types, using Wikipedia categories as the type system. The focus has been on expanding the set of target types based on hierarchical relationships and dealing with the imperfections of the type system [47, 9, 151, 97]. Importantly, these developments have been motivated and driven by the peculiarities of Wikipedia's category system. It is not known whether the same methods prove effective, and even if these issues persist at all, in case of other type taxonomies. We consider and systematically compare multiple type taxonomies (DBpedia, Freebase, Wikipedia, and YAGO). Additionally, there is the issue of representing entity type information, more specifically, to what extent the hierarchy of the taxonomy should be preserved. Yet another question is how to combine type-based and text-based matching in the retrieval model. In this chapter, we present the first comprehensive study on the usage of entity type information for entity retrieval.

The remainder of this chapter is organized as follows. In Section 3.1, we review related research. Section 3.2 introduces type-aware entity retrieval models. Next, in Section 3.3 we discuss alternative ways of representing hierarchical entity type information and present different type taxonomies. Section 3.4 describes our experimental setup. We report and discuss the experimental results in Section 3.5.

3.1 Related Work

The task of entity ranking has been studied in different flavors. *Ad hoc entity ranking* takes a keyword query as input and returns a ranking of relevant entities [153, 141]. *List search* further assumes that sought results are semantically related (e.g., "US presidents since 1960" or "Axis powers of World War II"); these semantic relationships may be specified with a set of target types, or a (small) set of example entities [13, 48]. *Related entity finding*, a special case of list search, requires result entities to be of a specific type and stand in a particular relation with a given input entity (e.g., "airlines that currently use Boeing 747 planes") [10]. Finally, answers to many questions in *question answering* are specific entities (e.g., "Who is the mayor of Berlin?") [115]. Our interest in this work lies in the usage of type information for general-purpose entity retrieval against a knowledge base, where queries may belong to any of the above categories.

3.1.1 Using Type Information in Entity Ranking

Early work represented type information as a separate field in a fielded entity model [208]. Demartini et al. [47] additionally expand type information using the underlying hierarchy. In later works, types are typically incorporated into the retrieval method by combining term-based similarity with a separate type-based similarity component. This combination may be done (i) using a linear interpolation [9, 97, 151] or (ii) in a multiplicative manner, where the type-based component essentially serves as a filter [29]. Raviv et al. [158] introduce a particular version of interpolation linearly aggregating each of the scores for the joint distribution of the query with entity document, type, and name. All the mentioned works have consistently reported significant performance improvements when a type-based component is incorporated into the (term-based) retrieval model. However, type-aware approaches had not been systematically compared prior to the work that we present in this chapter. We formalize these two general combination strategies, interpolation and filtering, in Section 3.2, and then compare them experimentally in Section 3.5.

Different approaches have measured the type-based similarity by using lexical type label similarity [187], descriptions of entities [96], overlap ratio of type sets [194], and even types added as a separated field in multi-field retrieval [208, 46]. In this work we use a state-of-the-art solution proposed by Balog et al. [9] (cf. Section 3.2.3).

3.1.2 Type Taxonomies

The choice of a particular type taxonomy is mainly motivated by the problem setting, depending on whether a wide-coverage type system (like Wikipedia categories) or a curated taxonomy (e.g., the DBpedia Ontology) is desired. The most common type system used in prior work is Wikipedia categories [47, 9, 97, 158, 29]. This is in part due to historical reasons, as this was the underlying type system used at the INEX Entity Ranking track, where type information was first exploited. Further choices include the DBpedia Ontology [11, 181], YAGO types [47, 169, 181, 140], Freebase [112], and schema.org [181]. To the best of our knowledge, ours is the first study to compare different type taxonomies for entity retrieval.

3.1.3 Representations of Type Information

Target types are commonly considered either as a set [151, 47, 158, 97] or as a bag (weighted set) [185, 9, 169]. Various ways of measuring type-based similarity have been proposed [187, 96, 194, 208, 46]. In this work we employ a probabilistic approach that represents entity type information as multinomial probability distributions [9] (cf. Section 3.2.3). Within a taxonomy, types are arranged in a hierarchy. (Wikipedia represents a special case here, as its categories do not form a well-defined "is-a" hierarchy.) Several approaches have attempted to expand the set of target types based on the hierarchical structure of the type system [151, 9, 29, 47, 11, 181]. Crucially, the investigation of type hierarchies has been limited to Wikipedia, and, even there, mixed results are reported [187, 208, 46, 88]. It remains an open question whether considering the hierarchical nature of types benefits retrieval performance. We aim to fill that gap.

3.1.4 Entity Type Assignments

A further complicating issue is that type information associated with entities in the knowledge base is incomplete, imperfect, or missing altogether for some entities. Gangemi et al. [60] distinguish between *extensional* coverage, i.e., the number of typed resources, and *intensional* coverage, i.e., conceptual completeness. Automatic typing of entities is a possible solution for alleviating some of these problems. For example, approaches to extend entity type assignments in DBpedia include mining associated Wikipedia articles for link relations [142], patterns over logical interpretations of the deeply parsed natural language definitions [60], or linguistic hypotheses about category classes [56]. Several works have addressed entity typing over progressively larger taxonomies with finer-grained types [54, 71, 156, 114, 202]. Regarding the task of

detecting and typing *emerging entities*, having fine-grained types for new entities is of particular importance for informative knowledge [112, 140].

3.2 Type-aware Entity Retrieval

In this section, we formally describe the type-aware entity retrieval models that we will be using for aiming to answer the research question (RQ1) *How can entity type information be utilized in ad-hoc entity retrieval?* Our contributions do not lie in this part; the techniques we present were shown to be effective in prior research. Table 3.1 presents the notation that we use throughout this chapter.

We formulate the retrieval task in a generative probabilistic framework. Given an input query q, we rank entities e according to

$$P(e|q) \propto P(q|e)\, P(e) \ . \tag{3.1}$$

When uniform entity priors are assumed, the final ranking of entities boils down to the estimation of $P(q|e)$. We consider the query in the term space as well as in the type space. Hence, we write $q = (q_w, q_t)$, where q_w holds the query terms (words) and q_t holds the *target types*. Two ways of factoring the probability $P(q|e)$ are presented in Section 3.2.1. All models share two components: term-based similarity, $P(q_w|e)$, and type-based similarity, $P(q_t|e)$. These are discussed in Sections 3.2.2 and 3.2.3, respectively.

3.2.1 Retrieval Models

We present two alternative approaches for combining term-based and type-based similarity.

Filtering

Assuming conditional independence between the term-based and type-based components, the final score becomes a multiplication of the components:

$$P(q|e) = P(q_w|e)\, P(q_t|e) \ . \tag{3.2}$$

This approach is a generalization, among others, of the one used by Bron et al. [29] (where the term-based information itself is unfolded into multiple components,

Symbol	Description
e	Entity ($e \in \mathcal{E}$)
$\mathcal{E}$	Set of all entities in the knowledge base
$\mathcal{E}_t$	Set of all entities typed with t
q	Query
t	Type ($t \in \mathcal{T}$)
$\mathcal{T}$	Set of all types in the taxonomy
$\mathcal{T}_e$	Set of all types assigned to e ($\mathcal{T}_e \subset \mathcal{T}$)
w	Term (word)
$a(e,t)$	Entity-type association weight for e and t
$f(w,e)$	Frequency of w in (the description of) e
$n(t,e)$	Entity-type association importance for e and t
$\pi(t)$	Parent type of t in the taxonomy
Rel_q	Set of relevant entities for q according to the ground truth
$rel(q,e)$	Relevance level of e for q according to the ground truth
$R_k(q)$	Set of top-k ranked entities for q
$score_M(e,q)$	Retrieval score of entity e for query q, given by model M
$score_{M_{(\phi_i)}}(t,q)$	Target type score of t for query q, given by model M, with array (ϕ_i) of underlying retrieval model parameters (omitted if empty)
$w2v(w)$	Pre-trained word2vec word embedding vector for w
$\boldsymbol{v}^{w2v}_{content}$	Centroid of word2vec vectors for all content words in v
$\mathbb{1}(p)$	Binary indicator function which returns 1 iff p is true
θ_e	Entity types distribution for e
θ_q	Target types distribution for q
λ_t	Weight of type-based component in interpolation model
k	Target types ranking cut-off in strict filtering model

Table 3.1 Glossary of the notation used in Chapters 3 and 4.

considering not only language models from textual context but also estimations of entity co-occurrences). We consider two specific instantiations of this model:

- **Strict filtering**, where $P(q_t|e)$ is 1 if the sets of target types and entity types have a non-empty intersection, and is 0 otherwise.
- **Soft filtering**, where $P(q_t|e) \in [0..1]$ and is estimated using the approach detailed in Section 3.2.3.

Interpolation

Alternatively, a mixture model may be used, which allows for controlling the importance of each component. Nevertheless, the conditional independence between q_w and q_t is still imposed by this model:

$$P(q|e) = (1 - \lambda_t)P(q_w|e) + \lambda_t P(q_t|e) \ , \tag{3.3}$$

where $P(q_t|e)$ is estimated using the approach detailed in Section 3.2.3. Several works use the interpolation model [151, 9, 158, 97].

3.2.2 Term-based Similarity

We base the estimation of the term-based component, $P(q_w|e)$, on statistical language modeling techniques since they have shown to be an effective approach in prior work [9, 97, 29, 12, 79]. Specifically, we employ the Sequential Dependence Model (SDM) [132]. Following what is done in the literature ([80]), we set the default parameters 0.8, 0.1, and 0.1 for terms, ordered, and unordered bigrams, respectively. We note that the term-based component is not the focus of this work; any other approach could also be plugged in (provided that the retrieval scores are mapped to probabilities).

3.2.3 Type-based Similarity

Rather than considering types simply as a set, we assume a distributional representation of types, also referred to as *bag-of-types*. Namely, a type in the bag may occur with repetitions, naturally rendering it more important. Following the literature ([9]), we represent type information as a multinomial probability distribution over types, both for queries and for entities. Specifically, let θ_q denote the target type distribution for the query q (such that $\sum_t P(t|\theta_q) = 1$). We assume that there is some mechanism in place that estimates this distribution; in our experiments, we will rely on an "oracle" that provides us exactly with this information (cf. Section 3.4.2). Further, let θ_e denote the target type distribution for entity e. We assume that a function $n(t, e)$ is provided, which returns 1 if e is assigned to type t, otherwise 0. We present various ways of setting $n(t, e)$ based on the hierarchy of the type taxonomy in Section 3.3. We note that $n(t, e)$ is not limited to having a binary value; this quantity could, for example, be used to reflect how important type t is for the given entity e. We use a multinomial distribution to allow for such future extensions. The type-based representation of an

entity e is estimated using Dirichlet smoothing:

$$P(t|\theta_e) = \frac{n(t,e) + \mu P(t)}{\sum_{t'} n(t',e) + \mu} \ , \tag{3.4}$$

where the background type model is obtained by a maximum-likelihood estimate:

$$P(t) = \frac{\sum_{e'} n(t,e')}{\sum_{t'} \sum_{e'} n(t',e')} \ . \tag{3.5}$$

The smoothing parameter μ in Eq. (3.4) is set to the average number of types assigned to an entity. In Eqs. (3.4) and (3.5), t' is any type in the taxonomy ($t' \in \mathcal{T}$) and e' is any entity in the knowledge base ($e' \in \mathcal{E}$).

With both θ_q and θ_e in place, we estimate type-based similarity using the Kullback-Leibler (KL) divergence of the two distributions:

$$P(q_t|e) = z \left(\max_{e'} KL(\theta_q \parallel \theta_{e'}) - KL(\theta_q \parallel \theta_e)\right) \ , \tag{3.6}$$

where z is a normalization factor:

$$z = 1 \Big/ \sum_{e} \max_{e'} (KL(\theta_q \parallel \theta_{e'}) - KL(\theta_q \parallel \theta_e)) \ .$$

Note that the smaller the divergence the more similar the distributions are, therefore in Eq. (3.6) we subtract it from the maximum KL-divergence, in order to obtain a probability distribution. For further details we refer to the literature ([9]).

3.3 Entity Type Representation

This section introduces alternative ways of representing hierarchical entity type information (Section 3.3.1) and the different type taxonomies that are considered in our experimental evaluation (Section 3.3.2).

3.3.1 Hierarchical Entity Type Representation

We consider different ways of representing hierarchical entity type information. Specifically, we investigate how to set the quantity $n(t,e)$, which is needed for estimating type-based similarity between target types of the query and types assigned to the entity in the knowledge base. Before proceeding further, we introduce some terminology and notation.

- $\mathcal{T}$ is a type taxonomy that consists of a set of hierarchically organized entity types, and $t \in \mathcal{T}$ is a specific entity type.
- $\mathcal{E}$ is the set of all entities in the knowledge base, and $e \in \mathcal{E}$ is a specific entity.
- $\mathcal{T}_e$ is the set of types that are assigned to the entity e in the knowledge base. We refer to this as a set of *assigned types*. Note that $\mathcal{T}_e$ might be an empty set.

We impose the following constraints on the type taxonomy.

- There is a single root node t_0 that is the ancestor of all types (e.g., `<owl:Thing>`). Since all entities belong to this type, it is excluded from the set of assigned types by definition.
- We restrict the type taxonomy to subtype-supertype relations; each type t has a single parent type denoted as $\pi(t)$.
- Type assignments are transitive, i.e., an entity that belongs to a given type also belongs to all ancestors of that type: $t \in \mathcal{T}_e \wedge \pi(t) \neq t_0 \implies \pi(t) \in \mathcal{T}_e$.

We further note that an entity might belong to multiple types under different branches of the taxonomy. Assume that t_i and t_j are both types of e. It might be then that their nearest common ancestor in the type hierarchy is t_0.

While $\mathcal{T}_e$ holds the types assigned to entity e, there are multiple ways of defining a value, $n(t, e)$, which reflects the type's importance with respect to the given entity. This importance is taken into account when building the type-based entity representation in Eq. (3.4). In this work, we treat all types equally important for an entity, i.e., use binary values for $n(t, e)$.

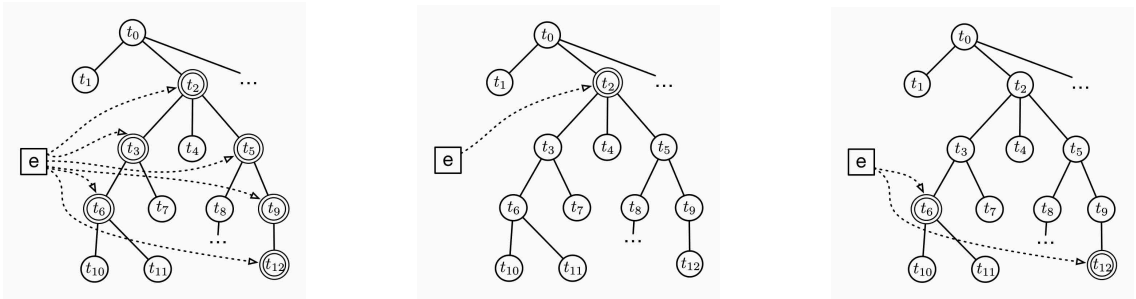
We consider the following three options for representing hierarchical type information; see Fig. 3.2 for an illustration. In our definitions, we use $\mathbb{1}(x)$ as an indicator function, which returns the value 1 if condition x is true and returns 0 otherwise.

- **Types along path-to-top**: It considers all types that are assigned to the entity in the knowledge base, excluding the root (from constraint (iii) it follows that $\mathcal{T}_e$ contains all the types in the path to the top-level nodes):

$$n(t, e) = \mathbb{1}\big(t \in \mathcal{T}_e\big) \ .$$

- **Top-level type(s)**: Only top-level types are considered for an entity, that is, types that have the root node as their parent:

$$n(t, e) = \mathbb{1}\big(t \in \mathcal{T}_e \wedge \pi(t) = t_0\big) \ .$$



(a) Type(s) along path (b) Top-level type(s) (c) Most specific type(s)

Fig. 3.2 Alternative ways of representing entity-type assignments with respect to the type taxonomy. The dashed arrows point to the types that are assigned to entity e. The root node of the taxonomy is labeled with t_0.

- **Most specific type(s)**: From each path, only the most specific type is considered for the entity:

$$n(t,e) = \mathbb{1}\big(t \in \mathcal{T}_e \wedge \nexists\, t' \in \mathcal{T}_e : \pi(t') = t\big) \ .$$

Even though there may be alternative representations, these three are natural ways of encoding hierarchical information.

3.3.2 Entity Type Taxonomies

We study multiple type taxonomies from various knowledge bases: DBpedia, Freebase, Wikipedia, and YAGO. These vary a lot in terms of hierarchical structure and in how entity-type assignments are recorded. We normalize these type taxonomies to a uniform structure, adhering to the constraints specified in Section 3.3.1. Table 3.2 presents an overview of the type systems (after normalization). The number of type assignments are counted according to the representation along path-to-top. Properties of the four type systems and details of the normalization process are discussed below.

Type Taxonomies

Wikipedia categories. The Wikipedia category system, developed and extended by Wikipedia editors, consists of textual labels known as categories. This categorization is not a well-defined "is-a" hierarchy, but a graph; a category may have multiple parent categories and there might be cycles along the path to ancestors [98]. Also, categories often represent only loose relatedness between articles; category assignments are neither consistent nor complete [48].

Type system	DBpedia	Freebase	Wikipedia categories	YAGO
Number of types	713	1,719	423,636	568,672
Number of top-level types	22	92	27	61
Number of leaf-level types	561	1,626	303,956	549,754
Height	7	2	35	19
Number of types used	408	1,626	359,159	314,632
Number of entities with type	4.87M	3.27M	3.52M	2.88M
Avg. number of types per entity	2.8	4.4	20.8	13.4
Mode depth	2	2	11	4

Table 3.2 Overview of the normalized type taxonomies and their statistics. The top block is about the taxonomy itself; the bottom block is about type assignments of entities.

We transformed the Wikipedia category graph, consisting of over 1.16M categories, into a type taxonomy as follows. First, we selected a set of 27 top-level categories covering most of the knowledge domains.[1] These became the top-level nodes of the taxonomy, all with a single common root type `<owl:Thing>`. All super-categories that these selected top-level categories might have in the graph were discarded. Second, we removed multiple inheritances by selecting a single parent per category. For this, we considered the population of a category to be the set of its assigned articles. Each category was linked in the taxonomy with a single parent in the graph whose intersection between their populations is the maximal among all possible parents; in case of a tie, the most populated parent was chosen. Under this criterion, and for the purpose of understanding hierarchical relations, any category without a parent was discarded. Lastly, from this partial hierarchy (which is still a graph, not a tree), we obtained the final taxonomy by performing a depth-first exploration from each top-level category, and avoiding to add those arcs that would introduce cycles. This depth-first approach was previously used by Fossati et al. [56] for enforcing taxonomic constraints on Wikipedia categories. The resulting taxonomy contains over 423k categories and reaches a maximum depth of 35 levels.[2]

[1]The selected top-level categories are the main categories for each section of the portal https://en.wikipedia.org/wiki/Portal:Contents/Categories. (As an alternative, we also considered the categories from https://en.wikipedia.org/wiki/Category:Main_topic_classifications, and found that it comprises a similar category selection).

[2]We have confirmed experimentally that enforcing the Wikipedia category graph to satisfy the taxonomical constraints does not hurt retrieval performance. In fact, it is the opposite: it results in small, but statistically significant improvements [65].

DBpedia ontology. The DBpedia Ontology is a well-designed hierarchy since its inception; it was created manually by considering the most frequently used infoboxes in Wikipedia. It continues to be properly curated to address some weaknesses of the Wikipedia infobox space. While the DBpedia Ontology is clean and consistent, its coverage is limited to entities that have an associated infobox. It consists of 713 classes, including the root, organized in a hierarchy of 7 levels.

YAGO taxonomy. YAGO is a huge semantic knowledge base, derived from Wikipedia, WordNet, and GeoNames [176]. Its type classification schema is constructed by taking leaf categories from the category system of Wikipedia and then using WordNet synsets to establish the hierarchy of classes. The result is a deep subsumption hierarchy, consisting of over 568k classes. We work with the YAGO taxonomy from the current version of the ontology (3.0.2). We normalized it by adding a root node, `<owl:Thing>`, as a parent to every top-level type.

Freebase types. Freebase has a two-layer categorization system, where types on the bottom level are grouped under high-level domains. We used the latest public Freebase dump (2015-03-31), discarding domains meant for administering the Freebase service itself (e.g.; `base`, `common`). Additionally, we made `<owl:Thing>` the common root of all the domains, and finally obtained a taxonomy of 1,719 types.

Entity-Type Assignments

Now that we have presented the four type taxonomies, we also need to discuss how type assignments of entities are obtained. We use DBpedia 2015-10 as our knowledge base, which makes DBpedia types, Wikipedia categories, and YAGO type assignments readily available. For the fourth type taxonomy, Freebase, we followed same-as links from DBpedia to Freebase (which exist for 95% of the entities in DBpedia) and extracted type assignments from Freebase. It should be noted that entity-type assignments are provided differently for each of these taxonomies; DBpedia and Freebase supply a single (most specific) instance type for an entity, Wikipedia assignments include multiple categories for a given entity (without any restriction), while YAGO adheres to the representation along path. We treat all entity-type assignments transitively, adhering to Constraint (iii) in Section 3.3.1.

Category	Description	Example
INEX-LD	General keyword queries	"Guitar origin blues"
ListSearch	Entity list queries	"Products of Medimmune, Inc."
QALD-2	Natural language queries	"Who was called Scarface?"
SemSearch ES	Named entity queries	"Brooklyn bridge" or "Ashley Wagner"

Table 3.3 Query categories in the DBpedia-Entity v2 collection.

3.4 Experimental Setup

We base our experiments on the DBpedia knowledge base (version 2015-10). DBpedia [107], as a central hub in the Linked Open Data cloud, provides a large repository of entities, which are mapped—directly or indirectly; cf. Section 3.3.2—to each of the type taxonomies of interest.

3.4.1 Test Collection

Our experimental platform is based on the DBpedia-Entity v2 test collection[3] developed by Hasibi et al. [80]. The dataset contains 467 queries, synthesized from various entity-related benchmarking evaluation campaigns. These range from short keyword queries to natural language questions; see Table 3.3.

3.4.2 Target Entity Types Oracle

Throughout the set of experiments of this chapter (in Section 3.5), we make use of a so-called *target entity types oracle*. We assume that there is an "oracle" process in place that provides us with the (distribution of) correct target types for a given query. This corresponds to the setting that was employed at previous benchmarking campaigns (such as the INEX Entity Ranking track [48] and the TREC Entity track [13]), where target types are provided explicitly as part of the topic definition. We employ this idealized setting to ensure that our results reflect the full potential of using type information, without being hindered by the imperfections of an automated type detector.

For a given query q, we take $\mathcal{T}_q = \bigcup_{e \in Rel_q} \mathcal{T}_e$, the union of all types of all entities that are judged relevant for that query. Each of these types $t \in \mathcal{T}_q$ becomes a target type, and its probability $P(t|\theta_q)$ is set proportional to the number of relevant entities that have that type. Formally, the oracle O scores target types as follows:

[3] http://tiny.cc/dbpedia-entity

$$score_O(t, q) = \sum_{e \in Rel_q \cap \mathcal{E}_t} rel(e, q) \ , \tag{3.7}$$

where $\mathcal{E}_t$ denotes the set of entities that are assigned to type t and $rel(e, q)$ is the relevance score of entity e for query q, according to the ground truth. Then, the oracle target distribution is given by:

$$P(t|\theta_q) = \frac{score_O(t, q)}{\sum_{t'} score_O(t', q)} \ . \tag{3.8}$$

3.4.3 Entity Retrieval Models

As our baseline, we use a term-based approach, specifically the Sequential Dependence Model (SDM) [132], which we described in Section 3.2.2. We compare three type-aware retrieval models (cf. Section 3.2.1): strict filtering, soft filtering, and interpolation. For the latter, we perform a sweep over the possible type weights $\lambda_t \in [0, 1]$ in steps of 0.05, and use the best performing setting when comparing against other approaches. (Automatically estimating the λ_t parameter is outside the scope of this work.)

3.4.4 Type Assignments

In the default setting, we include all entities from the knowledge base and use the original set of relevance assessments. By doing so, some entities and queries do not have types assigned from one or more taxonomies. Therefore, we introduce an additional experimental setting, referred to as *1TT*, to ensure that the differences we observe are not a result of missing type assignments.

In the 1TT setting, for each type taxonomy, we restrict our set of entities to those that have at least one type assigned in the taxonomy. We also restrict the set of queries to those that have target types in that type system; queries without any relevant results (as a consequence of these restrictions) are filtered out. This leaves us with a total of 446 queries for DBpedia, 454 for Freebase, 463 for Wikipedia, and 450 for YAGO.

3.5 Results and Analysis

In this section, we present evaluation results for all combinations of the three proposed dimensions: type taxonomies, type representation modes, and retrieval models. When discussing the results, we use the term *configuration* to refer to a particular combination of type taxonomy, type representation, and retrieval model.

Model	Strict filtering		Soft filtering		Interpolation		
	@10	@100	@10	@100	@10	@100	λ_t
Baseline [132]	0.4185	0.5143	0.4185	0.5143	0.4185	0.5143	-
DBpedia							
along-path	0.3998	0.4947	**0.4440**†	**0.5279**†	0.4549‡	0.5337‡	0.25
top-level	0.3998	0.4947	0.4307	0.5187	0.4414‡	0.5253‡	0.25
most specific	**0.4389**	**0.5186**	0.4404†	0.5259	**0.4579**‡	**0.5366**‡	0.15
Freebase							
along-path	0.4113	0.5003	**0.4766**‡	**0.5486**‡	**0.4702**‡	**0.5453**‡	0.35
top-level	0.4113	0.5003	0.4758‡	0.5461‡	0.4690‡	0.5428‡	0.40
most specific	**0.4306**	**0.5127**	0.4734‡	0.5467‡	0.4664‡	0.5432‡	0.35
Wikipedia							
along-path	0.4310‡	0.5170	0.4256	0.5215	0.4283‡	0.5211‡	0.05
top-level	0.1102	0.3243	0.2707	0.4271	0.4185	0.5143	0.00
most specific	**0.5362**‡	**0.5775**‡	**0.4742**‡	**0.5506**‡	**0.4603**‡	**0.5432**‡	0.25
YAGO							
along-path	0.3814	0.4770	**0.4718**‡	0.5483‡	**0.4647**‡	0.5421‡	0.35
top-level	0.3814	0.4770	0.4186	0.5129	0.4314‡	0.5223‡	0.25
most specific	**0.4235**	**0.5038**	0.4685‡	**0.5492**‡	0.4561‡	**0.5429**‡	0.20

Table 3.4 Entity retrieval performance using oracle target types, returning all entities from the knowledge base (ALL). For the interpolation model, λ_t is value of the best empirically found interpolation parameter. Performance is measured in terms of NDCG@10 and NDCG@100. Statistical significance, tested using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$ and $p < 0.0005$, is denoted by † and ‡, respectively.

Recall that we distinguish between two settings (cf. Section 3.4.4): ranking all entities in the knowledge base (ALL) and considering only entities that have types assigned to them in a given type taxonomy (1TT). Tables 3.4 and 3.5 show results corresponding to these two settings, respectively. Our main evaluation metric is normalized discounted cumulative gain with a cut-off of 10 (NDCG@10); we also report on NDCG@100. We test statistical significance to measure our confidence in rejecting the null hypothesis that states that our improvements occur by chance. Specifically, we use a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$ and 0.0005. For an easier visual inspection, the NDCG@10 scores are also plotted in Fig. 3.3, where the red line corresponds to the term-based baseline.

Model	Strict filtering		Soft filtering		Interpolation		
	@10	@100	@10	@100	@10	@100	λ_t
DBpedia							
Baseline [132]	0.3036	0.4119	0.3036	0.4119	0.3036	0.4119	-
along-path	0.4600‡	0.5079‡	**0.4353**‡	**0.4894**‡	**0.4172**‡	**0.4746**‡	0.65
top-level	0.4600‡	0.5079‡	0.4196‡	0.4779‡	0.4141‡	0.4725‡	0.70
most specific	**0.5092**‡	**0.5393**‡	0.4309‡	0.4864‡	0.4075‡	0.4696‡	0.55
Freebase							
Baseline [132]	0.3322	0.4403	0.3322	0.4403	0.3322	0.4403	-
along-path	0.4513‡	0.5106‡	0.4471‡	**0.5085**‡	0.4408‡	0.5021‡	0.65
top-level	0.4513‡	0.5106‡	**0.4492**‡	0.5076‡	**0.4443**‡	**0.5031**‡	0.70
most specific	**0.4736**‡	**0.5251**‡	0.4393‡	0.5034‡	0.4300‡	0.4966‡	0.60
Wikipedia							
Baseline [132]	0.3666	0.4727	0.3666	0.4727	0.3666	0.4727	-
along-path	0.4121‡	0.5000‡	0.4145‡	0.5014‡	0.3944‡	0.4877	0.40
top-level	0.0963	0.2950	0.2193	0.3777	0.3666	0.4727	0.00
most specific	**0.5874**‡	**0.6071**‡	**0.4741**‡	**0.5393**‡	**0.4474**‡	**0.5180**‡	0.65
YAGO							
Baseline [132]	0.3076	0.4180	0.3076	0.4180	0.3076	0.4180	-
along-path	0.4325‡	0.4904‡	**0.4453**‡	**0.5041**‡	**0.4313**‡	**0.4879**‡	0.75
top-level	0.4325‡	0.4904‡	0.3630‡	0.4476‡	0.3807‡	0.4513‡	0.85
most specific	**0.4843**‡	**0.5231**‡	0.4347‡	0.4998‡	0.4211‡	0.4850‡	0.70

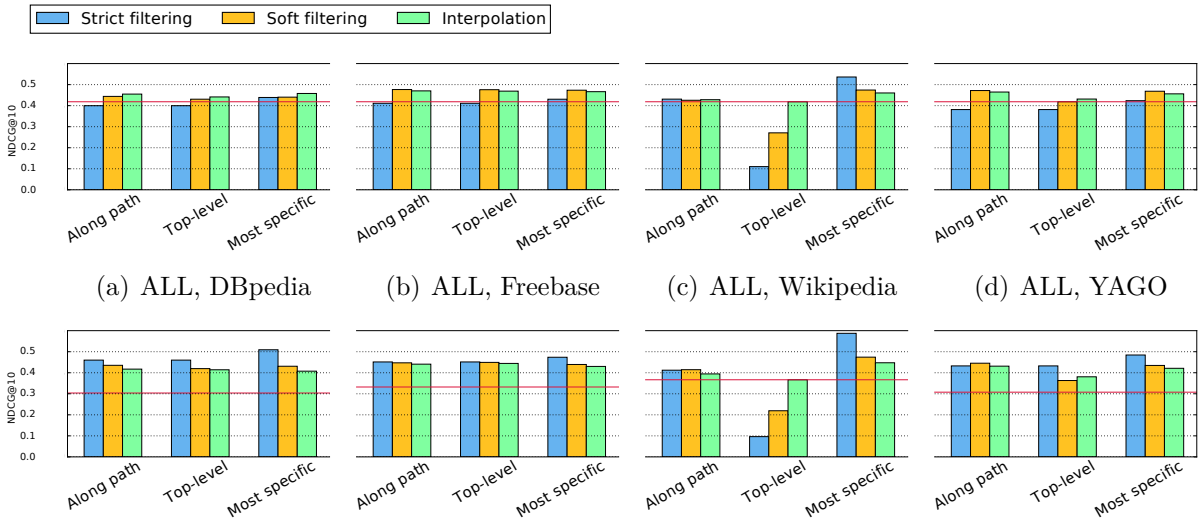
Table 3.5 Entity retrieval performance using oracle target types, considering only entities that have types assigned to them in the respective type taxonomy (1TT). For the interpolation model, λ_t is value of the best empirically found interpolation parameter. Performance is measured in terms of NDCG@10 and NDCG@100. Statistical significance, tested using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$ and $p < 0.0005$, is denoted by † and ‡, respectively.

3.5.1 Type Taxonomy

Our first research question (RQ1a) concerns the impact of the particular choice of type taxonomy.

- **RQ1a** What is the impact of the particular choice of type taxonomy on entity retrieval performance?

It is clear that Wikipedia, in combination with the *most specific* type representation, performs best for both settings (ALL and 1TT, Figs. 3.3(c) and 3.3(g)), and yields



(a) ALL, DBpedia (b) ALL, Freebase (c) ALL, Wikipedia (d) ALL, YAGO

(e) 1TT, DBpedia (f) 1TT, Freebase (g) 1TT, Wikipedia (h) 1TT, YAGO

Fig. 3.3 Entity retrieval performance for all combinations of type taxonomies, type representation modes, and retrieval models. Top: all entities in the knowledge base (ALL); bottom: only entities with types from the given type taxonomy (1TT). The red line corresponds to the term-based baseline(s). Performance is measured by NDCG@10.

substantial and significant improvements for all three retrieval models. As for the other two type representations for Wikipedia, performance slightly improves for *along-path* (significant for 1TT). *top-level* Wikipedia types do not contribute when using the interpolation model ($\lambda_t = 0$), and are rather harmful when using either strict or soft filtering.

DBpedia and Freebase also show improvements for the ALL setting in all configurations, except the strict filtering model (Figs. 3.3(a) and 3.3(b)). The improvements for these smaller, shallower taxonomies are significant for all configurations in the 1TT setting (Figs. 3.3(e) and 3.3(f)). The case of YAGO is similar: all but the strict filtering configurations improve in the ALL setting (Figs. 3.3(d)), and all 1TT configurations yield significant improvements (Fig. 3.3(h)).

Comparing the results for the *top-level* representation between YAGO and Wikipedia, it is clear that the *top-level* Wikipedia categories that were chosen for enforcing taxonomic constraints are not appropriate for conveying entity type information.

3.5.2 Type Representation

The second research question (RQ1b) is about type representation.

- **RQ1b** How can one represent hierarchical entity type information for entity retrieval?

The question has a clear answer: keeping only the *most specific* types in the hierarchy provides the best performance across the board, for all configurations. This fact is also in line with findings in past work (cf. Section 3.1). As for the other two representations, *along-path* is the better performing representation. The difference between *along-path* and *top-level*, however, are generally small, in particular for the smaller taxonomies, DBpedia and Freebase.

Overall, we have verified that hierarchical relationships from ancestor types result in improved retrieval effectiveness, but simply resorting to the *most specific* type assignments in the knowledge base is the most effective way of representing entity type information.

3.5.3 Type-Aware Retrieval Model

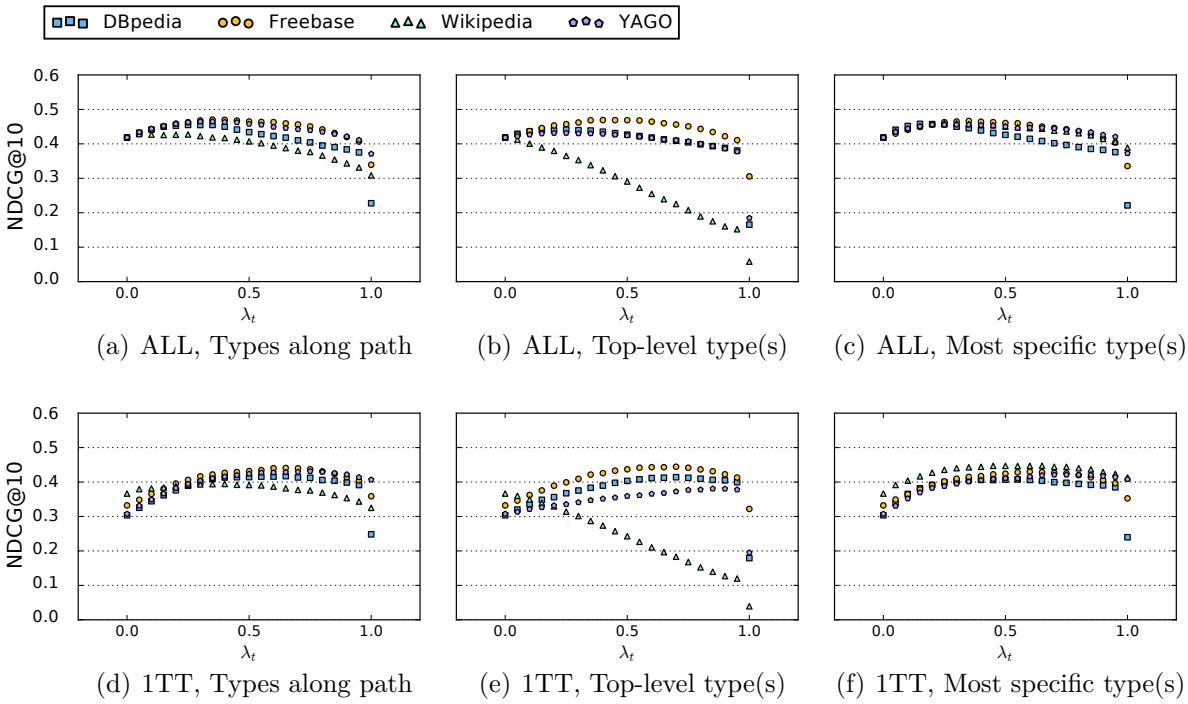
Our third research question (RQ1c) concerns the type-aware retrieval model.

- **RQ1c** How can one combine term-based and type-based matching for entity retrieval?

According to the 1TT setting (Table 3.5), strict filtering with the *most specific* type representation is the best retrieval model for all configurations (achieving, in particular, a relative improvement of 67% in terms of NDCG@10 on DBpedia types), significantly outperforming the baseline in all cases. This no longer holds in the ALL setting (Table 3.4). The soft filtering and interpolation models perform best for all taxonomies, with small differences between the two, depending on the type representation. In the ALL setting, strict filtering always performs the worst and is often even below the baseline when *along-path* or *top-level* type representation is used.

Comparing the respective λ_t type weights in Tables 3.4 and 3.5, it is noticeable that the interpolation model relies more on the type component in the 1TT than in the ALL setting. This makes perfect sense since in the ALL setting many entities lack type information in the knowledge base.

Figure 3.4 shows the performance of the interpolation model when varying the value of λ_t. We observe that with the exception of Wikipedia using the *top-level* type representation, type information always improves over the baseline. In the ALL setting, performances generally peak in the 0.2–0.4 range, while for 1TT it is higher, around 0.5–0.7.



(a) ALL, Types along path (b) ALL, Top-level type(s) (c) ALL, Most specific type(s)

(d) 1TT, Types along path (e) 1TT, Top-level type(s) (f) 1TT, Most specific type(s)

Fig. 3.4 Retrieval performance, in terms of NDCG@10, using the interpolation model with different type weights, λ_t. Top: all entities in the knowledge base (ALL); bottom: only entities with types from the given type taxonomy (1TT). The leftmost data points ($\lambda_t = 0$) correspond to the term-based baseline.

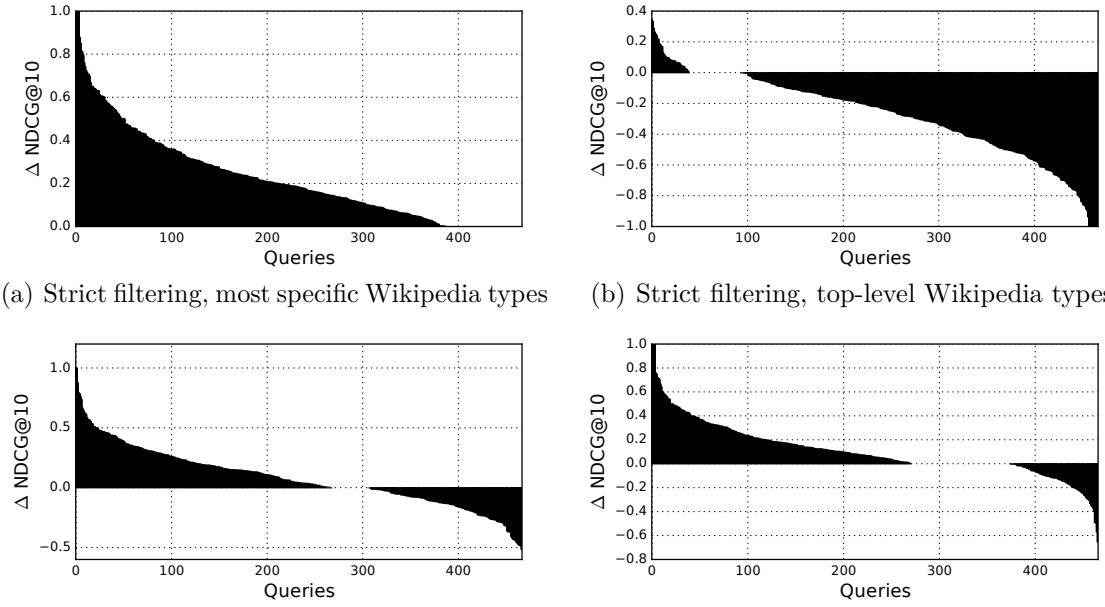
3.5.4 Analysis

We perform a more detailed analysis of particular configurations in order to gain a deeper understanding of each of the dimensions of entity type information. We focus on the bottom row of bar plots in Fig. 3.3 (e)–(h), that is, the 1TT experimental setting. There, as we previously explained, it is ensured that the differences we observe are not a result of missing type assignments. Figure 3.5 shows, for each of the selected configurations, the differences in NDCG@10 scores (ΔNDCG@10 hereinafter) on the level of individual queries between a given configuration and the corresponding (term-based) baseline. For the ease of visual comprehension, queries are ordered by ΔNDCG@10. Table 3.6 lists specific queries with the largest ΔNDCG@10 differences, along with their "best" oracle target types (according to the scoring function defined by Eq. 3.7).

The best performing configuration uses *most specific* Wikipedia types with the strict filtering model (Fig. 3.3 (g)). As it can be observed in Fig. 3.5 (a), most of the queries are improved while none of them are hurt. The queries with the highest

Δ	Query	Best oracle target type(s)
Strict filtering, most specific, Wikipedia		
+1.0	SemSearch_ES-129 (“pizza populous detroit mi”)	American pizza, Pizza varieties
+1.0	SemSearch_ES-75 (“sagemont church houston tx”)	Lists of churches in the US, Cathedrals in the US
+1.0	QALD2_te-53 (“is the ruling party in Lisbon”)	Liberal-conservative parties
+1.0	SemSearch_ES-104 (“bourbonnais il”)	Amtrak accidents, Railway accidents in Illinois
+0.88	SemSearch_ES-96 (“New England Coffee”)	Whitbread divisions and subsidiaries
Strict filtering, top-level, Wikipedia		
+0.34	SemSearch_ES-99 (“University of York”)	Geography, Social sciences
+0.33	INEX_XER-63 (“Hugo awarded best novels”)	Social sciences, Arts
+0.29	SemSearch_ES-137 (“steak express”)	Social sciences, Philosophy
+0.26	QALD2_tr-91 (“organizations were founded in 1950”)	Social sciences, Society
+0.25	INEX_LD-20120131 (“vietnam travel national park”)	Social sciences, Geography
-1.0	QALD2_te-76 (“List the children of Margaret Thatcher”)	Social sciences, Philosophy
-1.0	SemSearch_ES-111 (“eagle rock ca”)	Society, History
-1.0	QALD2_tr-13 (“classis does the Millepede belong to”)	Social sciences, Nature
-1.0	QALD2_te-90 (“is the residence of the prime minister of Spain”)	Arts, Philosophy
-1.0	SemSearch_ES-107 (“concord steel”)	Social sciences, Philosophy
Interpolation, most specific, Wikipedia		
+1.0	SemSearch_ES-129 (“pizza populous detroit mi”)	American pizza
+1.0	QALD2_te-53 (“is the ruling party in Lisbon”)	Liberal-conservative parties
+0.88	SemSearch_ES-124 (“motorola bluetooth hs850”)	Bluetooth, Electronics companies of the US
+0.8	SemSearch_ES-50 (“laura steele bob and tom”)	American comedy radio programs
+0.79	SemSearch_ES-96 (“New England Coffee”)	Whitbread divisions and subsidiaries
-0.47	SemSearch_ES-22 (“city of charlotte”)	City councils in the US, Years in North Carolina
-0.48	SemSearch_ES-98 (“University of Texas at Austin”)	Association of American Universities
-0.48	QALD2_tr-54 (“was the wife of US president Lincoln”)	First Ladies of the US, Lincoln family
-0.49	TREC_Entity-10 (“Campuses of Indiana University”)	Joint Venture Schools
-0.52	QALD2_tr-16 (“the capitals of all countries in Africa”)	Capitals in Africa
Interpolation, most specific, DBpedia		
+1.0	SemSearch_ES-129 (“pizza populous detroit mi”)	Food
+1.0	QALD2_te-43 (“all breeds of the German Shepherd dog”)	Book
+1.0	INEX_LD-2012309 (“residents small island city state Malay Peninsula Chinese”)	Settlement
+1.0	QALD2_te-55 (“Greek goddesses dwelt on Mount Olympus”)	Mythological Figure
+0.76	QALD2_te-42 (“is the husband of Amanda Palmer”)	Writer
-0.4	QALD2_te-27 (“Sean Parnell is the governor of US state”)	Office Holder
-0.49	QALD2_tr-54 (“was the wife of US president Lincoln”)	Office Holder
-0.5	QALD2_tr-4 (“river does the Brooklyn Bridge cross”)	Bridge
-0.57	QALD2_te-90 (“is the residence of the prime minister of Spain”)	Building
-0.65	SemSearch_ES-89 (“university of north dakota”)	University

Table 3.6 Queries with the largest ΔNDCG@10 differences with respect to the term-based baseline, using oracle target types and the 1TT setting.



(a) Strict filtering, most specific Wikipedia types

(b) Strict filtering, top-level Wikipedia types

(c) Interpolation, most specific Wikipedia types

(d) Interpolation, most specific DBpedia types

Fig. 3.5 Differences in NDCG@10 per query between a given type-aware entity retrieval configuration and its corresponding (term-based) baseline, using only entities with types from that type taxonomy (1TT).

ΔNDCG@10 are mostly named entity queries, with a very suitable best oracle target type to be used as strict filter (first block of Table 3.6).

By changing type representation from *most specific* to *top-level*, the performance for Wikipedia types with strict filtering goes from best to worst (Fig. 3.3 (g)). As we can see in Fig. 3.5 (b), the vast majority of queries is negatively affected. Queries that are most harmed are natural language queries like "classis does the Millepede belong to" and "is the residence of the prime minister of Spain" (second block of Table 3.6). For queries that are improved, we observe that the best oracle target types are high-level and with a large descendants subtree like "Social sciences," corresponding more to broad knowledge domains that result in more permeable filters.

We also observe the results after changing another dimension of the best performing setting: *most specific* Wikipedia types now with the *interpolation* model (Fig. 3.3 (g)). Figure 3.5 (c) shows the distribution of ΔNDCG@10 values. Four out of the five most improved queries are named entity queries (third block of Table 3.6), whereas the most hurt queries belong to diverse categories. The λ_t parameter is set here to 0.65, which means that the type component is given clearly more weight than the term component. Yet, mean performance is much lower for the interpolation model than for strict filtering (0.4474 vs. 0.5874).

Avg. Δ across taxos.	Query	ΔNDCG@10			
		DB-pedia	Free-base	Wiki-pedia	YAGO
+1.0	SemSearch_ES-129 (“pizza populous detroit mi”)	+1.0	+1.0	+1.0	+1.0
+0.69	SemSearch_ES-4 (“NAACP Image Awards”)	+0.79	+0.56	+0.85	+0.56
+0.68	SemSearch_ES-115 (“goodwill of michigan”)	+0.64	+0.64	+0.63	+0.78
+0.6	INEX_XER-140 (“Airports in Germany”)	+0.59	+0.59	+0.57	+0.63
+0.59	QALD2_tr-42 (“are the official languages of the Philippines”)	+1.0	+0.49	+0.5	+0.36
+0.59	SemSearch_ES-78 (“sharp pc”)	+0.65	+0.33	+0.71	+0.65
+0.58	SemSearch_ES-1 (“44 magnum hunting”)	+0.32	+1.0	+0.47	+0.52
+0.58	SemSearch_ES-124 (“motorola bluetooth hs850”)	+0.03	+0.7	+0.65	+0.92
+0.57	SemSearch_ES-50 (“laura steele bob and tom”)	+0.49	+0.67	+0.8	+0.32
+0.56	QALD2_te-60 (“a list of all lakes in Denmark”)	+0.54	+0.51	+0.61	+0.57

Table 3.7 Queries with the largest average ΔNDCG@10 improvements across all type taxonomies, using most specific types with strict filtering and the 1TT setting. Differences greater than 0.05 are shown in green.

Finally, from this last configuration (interpolation with *most specific* Wikipedia types), we compare against the configuration where the third dimension, type taxonomy, is changed to a smaller, shallower taxonomy like DBpedia (Fig. 3.3(e)). The best performing λ_t is set here to 0.55, which represents a balanced interpolation. The ΔNDCG@10 values in Fig. 3.5 (d) are positive for a large proportion of queries. Around 100 queries are unaffected, while a few are moderately negatively impacted by the type-aware model. It is difficult to identify any patterns here (last block of Table 3.6), apart from noting that most queries with the largest impact (either positive or negative) are natural language queries. Moreover, given that the DBpedia taxonomy has a depth of six levels, even *most specific* types are not specific enough.

Additionally, to answer the question whether the same queries are helped/hurt using different taxonomies, we show in Table 3.7 the top ten queries according to average ΔNDCG@10 across all four taxonomies. All the queries in this configuration result to have non-negative ΔNDCG@10 in every type taxonomy. We can observe that most of the queries that are helped are named entity queries.

3.6 Summary

In this chapter, we have furthered our understanding on the usage of target type information for entity retrieval over structured data sources. Specifically, we have conducted a systematic comparison of four well-known type taxonomies (DBpedia, Freebase, Wikipedia, and YAGO) across three dimensions of interest: the representation of hierarchical entity type information, the way to combine term-based and type-

based information, and the impact of choosing a particular type taxonomy. Using an idealized "oracle" setting for identifying the target entity types, we have found that type information can significantly and substantially improve a strong text-only entity retrieval baseline.

In a realistic setting, target entity types are not provided, but need to be automatically identified based on the keyword query. Hence, in the next chapter, we will discuss the problem of automatically identifying target entity types, as well as assess how automatically obtained target type information impacts entity retrieval performance.

Chapter 4

Identifying Target Entity Type Information

The previous chapter described a systematic comparison of dimensions in type-aware entity retrieval. There, we considered target entity type identification using an oracle process, based on the set of known relevant entities (cf. Section 3.4.2). Indeed, target entity types may be provided by the user explicitly as part of the search request, for example, via faceted user interfaces. Often, however, users would prefer to use simple keyword queries as input. In that case, target entity types need to be identified automatically based on the keyword query. In this chapter, we discuss how to assign target entity types to queries from a type taxonomy.

We start by revisiting the task of hierarchical target type identification (Section 4.1). The chapter also describes previous work on this problem (Section 4.2). In Section 4.3, we present methods for identifying target entity types, in particular, we introduce our approach that constitutes the state of the art for this task. Section 4.4 details the test collection we developed for hierarchical target type identification. We then discuss the experimental results on this task, and on the usage of automatically identified target entity types for the main problem of type-aware entity retrieval (Section 4.5).

4.1 Hierarchical Target Entity Type Identification

As our starting point, we take the definition of the *hierarchical target type identification* (HTTI) task, as introduced by Balog and Neumayer [11]: "finding the single most specific type within the ontology that is general enough to cover all relevant entities." We point out two major limitations with this definition and suggest ways to overcome them.

First, it is implicitly assumed that every query must have a *single* target type, which is not particularly useful in practice. Take, for example, the query "finland car industry manufacturer saab sisu," where both *Company* and *Automobile* are valid types. We shall allow for possibly multiple main types, if they are sufficiently different, i.e., lie on different branches in the taxonomy. Second, it can happen—and in fact it does happen for 33% of the queries considered in the work by Balog and Neumayer [11]—that a query cannot be mapped to any type in the given taxonomy (e.g., "Vietnam war facts"). However, those queries were simply ignored in the work by Balog and Neumayer [11]. Instead, we shall allow a query not to have any type (or, equivalently, to be tagged with a special `NIL`-type). This relaxation means that we can now take any query as input. Our revised task definition is thus as follows.

Definition 1. *Hierarchical target entity type identification (*HTTIv2*) is the task of finding the main target types of a query, from a type taxonomy, such that (i) these correspond to the most specific category of entities that are relevant to the query, and (ii) main types cannot be on the same branch in the taxonomy. If no matching type can be found in the taxonomy then the query is assigned a special* `NIL`*-type.*

We note that detecting `NIL`-types is a separate task on its own account, which we are not addressing in this work. For now, the importance of the `NIL`-type distinction is restricted to how the query annotations are performed.

4.2 Related Work

The INEX Entity Ranking track [48] and the TREC Entity track [13] both featured scenarios where target types are provided by the user. When explicit target type information is lacking, one might attempt to infer types from the keyword query. This subtask was introduced by Vallet and Zaragoza [185] as the *entity type ranking* problem. They extract entity mentions from the set of top relevant passages, then consider the types associated with the top-ranked entities using various weighting functions. Kaptein et al. [98] similarly use a simple entity-centric model. Manually assigned target types tend to be more general than automatically identified ones [97]. Having a hierarchical structure, therefore, makes it convenient to assign more general types. Balog and Neumayer [11] address a hierarchical version of the *target entity type identification* task, using the DBpedia Ontology and language modeling techniques. One approach uses an entity-centric strategy. The other builds a textual type representation by concatenating the descriptions of all the entities assigned to each type. We present a

detailed description of these models in Sections 4.3.1 and 4.3.2, and further expand on them in Section 4.3.3. Sawant and Chakrabarti [169] focus on telegraphic queries and assume that each query term is either a type hint or a "word matcher." They consider multiple interpretations of the query and tightly integrate type detection within the ranking of entities. Their approach further relies on the presence of a large-scale web corpus.

4.3 Approach

In this section, we describe target type identification models from the literature, to be used as baselines, and introduce our approach for learning to rank target entity types.

4.3.1 Entity-Centric Model

The entity-centric model can be regarded as the most common approach for determining the target types for a query in previous work [98, 11, 185]. This model also fits the *late fusion* design pattern for object retrieval [205]. The idea is simple: first, rank entities based on their relevance to the query, then look at what types the top-K ranked entities have. The final score for a given type t is the aggregation of the relevance scores of entities with that type. Formally:

$$score_{EC_{(M,K)}}(t,q) = \sum_{e \in R_K(q)} score_M(e,q)\, a(e,t) \ ,$$

where $R_K(q)$ is the set of top-K ranked entities for the keyword query q. The retrieval score of entity e is denoted by $score_M(e,q)$. In our experiments, we consider both Language Models and BM25 as the underlying entity retrieval model M. The rank cut-off threshold K is set empirically. The entity-type association weight, $a(e,t)$, is set uniformly across entities that are typed with t, and is 0 otherwise:

$$a(e,t) = \begin{cases} \frac{1}{|\mathcal{E}_t|} & e \in \mathcal{E}_t \\ 0 & \text{otherwise} \ . \end{cases}$$

We denote this entity-centric target type score by $EC_{M,K}(t,q)$.

4.3.2 Type-Centric Model

Alternatively, one can also build for each type a direct term-based representation (pseudo type description document), by aggregating descriptions of entities that belong to the given type into a pseudo-document representation of the type. Then, those type representations can be ranked much like documents. This model has been presented by Balog and Neumayer [11] using Language Models, and has been generalized to arbitrary retrieval models (and referred to as the *early fusion* design pattern for object retrieval [205]). The pseudo-frequency of word w given type t is defined as:

$$\tilde{f}(w,t) = \sum_{e} f(w,e)\, a(e,t) \ , \tag{4.1}$$

where $f(w,e)$ is the frequency of the term w in (the description of) entity e and $a(e,t)$, as before, denotes the entity-type association weight. The relevance score of a type for a given query $q = \langle q_1, \ldots, q_n \rangle$ is then calculated as the sum of the individual query term scores:

$$score_{TC_{(M)}}(t,q) = \sum_{i=1}^{n} score_M(q_i, \tilde{f}, \varphi) \ ,$$

where $score_M(q_i, \tilde{f}, \varphi)$ is the underlying term-based retrieval model M (e.g., LM or BM25), parameterized by φ. This model assigns a score to each query term q_i, based on the word pseudo-frequencies $\tilde{f}$. We denote this type-centric target type score by $TC_M(t,q)$.

4.3.3 Learning to Rank

The entity-centric and type-centric models capture different aspects of target entity type identification, and it is therefore sensible to combine the two. While this idea has been previously suggested [11], to the best of our knowledge, our work is the first to realize it, using a learning-to-rank (LTR) approach [68]. In addition, there are other signals that one could leverage, including knowledge base features and type label similarities. Table 4.1 summarizes the features we use for target entity type identification.

Knowledge base features

We assume that a knowledge base provides a type system of reference along with entity-type mappings. In this setting, features related to the hierarchy of the type taxonomy emerge naturally. In particular, instead of using absolute depth metrics of

#	Feature	Description	Kind	Value
Baseline features				
1–5	$EC_{BM25,K}(t,q)$	Entity-centric score (cf. Section 4.3.1) of type t for query q, with $K \in \{5,10,20,50,100\}$ using BM25	entity-centric	$[0..\infty)$
6–10	$EC_{LM,K}(t,q)$	Entity-centric score (cf. Section 4.3.1) of type t for query q, with $K \in \{5,10,20,50,100\}$ using LM	entity-centric	$[0..1]$
11	$TC_{BM25}(t,q)$	Type-centric score (cf. Section 4.3.2) of type t for query q, using BM25	type-centric	$[0..\infty)$
12	$TC_{LM}(t,q)$	Type-centric score (cf. Section 4.3.2) of type t for query q, using LM	type-centric	$[0..1]$
Knowledge base features				
13	$DEPTH(t)$	The hierarchical level of type t, normalized by the taxonomy depth	taxonomy	$[0..1]$
14	$CHILDREN(t)$	Number of children of type t in the taxonomy	taxonomy	$\{0,\ldots,\infty\}$
15	$SIBLINGS(t)$	Number of siblings of type t in the taxonomy	taxonomy	$\{0,\ldots,\infty\}$
16	$ENTITIES(t)$	Number of entities assigned to type t	coverage	$\{0,\ldots,\infty\}$
Type label features				
17	$LENGTH(t)$	Length of (the label of) type t in words	statistical	$\{1,\ldots,\infty\}$
18	$IDFSUM(t)$	Sum of IDF for terms in (the label of) type t	statistical	$[0..\infty)$
19	$IDFAVG(t)$	Avg of IDF for terms in (the label of) type t	statistical	$[0..\infty)$
20-21	$JTERMS_n(t,q)$	Query-type Jaccard similarity for sets of n-grams, for $n \in \{1,2\}$	linguistic	$[0..1]$
22	$JNOUNS(t,q)$	Query-type Jaccard similarity using only nouns	linguistic	$[0..1]$
23	$SIMAGGR(t,q)$	Cosine sim. between the q and t word2vec vectors aggregated over all terms of their resp. labels	distributional	$[0..1]$
24	$SIMMAX(t,q)$	Max. cosine sim. of word2vec vectors between each pair of query (q) and type (t) terms	distributional	$[0..1]$
25	$SIMAVG(t,q)$	Avg. of cosine sim. of word2vec vectors between each pair of query (q) and type (t) terms	distributional	$[0..1]$

Table 4.1 Features for learning to rank target entity types.

a type like done in the work by Tonon et al. [182], we use a normalized depth with respect to the height of the taxonomy (Feature 13). We also take into account the number of children and siblings of a type (Features 14 and 15). Intuitively, the more specific a type, the deeper it is located in the type taxonomy, and the less its number of children, while the more its number of siblings. Hence all three of these features capture how specific a type is according to its context in a type taxonomy. The type coverage (Feature 16) is also directly related to the intuition of type specificity: the more general the type, the larger the number of entities it tends to cover.

Type label features

We consider several signals for measuring the similarity between the surface form of the type label and the query. The type label length (Feature 17) and the IDF-related statistics (Features 18 and 19) are closely related to type specificity. The Jaccard similarities (Features 20 and 21) capture shallow linguistic similarities by n-gram

matches between the set of n consecutive terms in the query and the type labels, where $n \leq 2$, since the textual phrases in any of these labels are expected to be of short length. In particular, the bigram match ($n = 2$) makes sense for capturing some typical type label patterns, e.g., $\langle adjective \rangle$ $\langle noun \rangle$ in `German physicists`. A more constrained version, defined in Feature 22, measures the query-type Jaccard similarity over single terms ($n = 1$), which are nouns.

We use pre-trained word embeddings provided by the *word2vec* toolkit [135]. However, we only consider *content words* (linguistically speaking, i.e., nouns, adjectives, verbs, or adverbs). Feature 23 captures the compositional nature of words in type labels:

$$SIMAGGR(t, q) = cos(\boldsymbol{q}^{w2v}_{content}, \boldsymbol{t}^{w2v}_{content}) \ ,$$

where the query and type vectors are taken to be the $w2v$ centroids of their content words. Feature 24 measures the pairwise similarity between content words in the query and the type label:

$$SIMMAX(t, q) = \max_{w_q \in q, w_t \in t} cos(w2v(w_q), w2v(w_t)) \ ,$$

where $w2v(w)$ denotes the word2vec vector of term w. Feature 25 $SIMAVG(t)$ is defined analogously, but using avg instead of max.

4.4 Experimental Setup

We now describe the steps followed to build our test collection of target entity types, as well as the experimental setup in which we will use the test collection to evaluate the proposed approach.

4.4.1 Choice of Type Taxonomy

In our experiments, we focus only on DBpedia, for the following main reason. Both for evaluation and for supervised learning, one needs relevance assessments for target entity types of queries. For the two large taxonomies, Wikipedia and YAGO, human assessments are problematic; Wikipedia is not a proper type taxonomy and is huge, while YAGO does not provide human-readable labels for types. As we have seen in the previous section, Freebase behaves similarly to DBpedia, but it is not particularly interesting in the taxonomical sense, given that it has only two levels. This leaves us

with DBpedia. The DBpedia Ontology is small enough to be manageable by humans, and is a proper taxonomy.

We note that none of the elements of our our supervised learning approach are specific to this taxonomy, and our methods for target entity type identification can be applied on top of any type taxonomy.

4.4.2 Test Collection of Target Entity Types

We build a test collection for the revised hierarchical target type identification task (cf. Section 4.1). Having the DBpedia Ontology (version 2015-10) as our type taxonomy, we collect relevance labels via crowdsourcing for all the 485 queries in the DBpedia-Entity v1 collection [12] (which is a superset of the DBpedia-Entity v2 queries that we use for evaluating entity ranking).

Using the well-established top-N pooling technique [191], a pool of target entity types is constructed from four baseline methods, taking the top 10 types from each: entity-centric (cf. Section 4.3.1) using K=100, and type-centric (cf. Section 4.3.2), with both BM25 and LM as underlying retrieval methods. Additionally, we included all types returned by the target entity types oracle (cf. Section 3.4.2), to ensure that all reasonable types are considered when collecting human annotations.

We obtained target type annotations via the CrowdFlower (now: Figure Eight) crowdsourcing platform. Specifically, crowd workers were presented with a search query (along with the narrative from the original topic definition, where available), and a list of candidate types, organized hierarchically according to the taxonomy. We asked them to "select the single most specific type, that can cover all results the query asks for" (in line with [11]). If none of the presented types are correct, they were instructed to select the "None of these types" (i.e., `NIL`-type) option.

The annotation exercise was carried out in two phases. In the first phase, we sought to narrow down our pool to the most promising types for each query. Since the number of candidate types for certain queries was fairly large, they were broken down to multiple micro-tasks, such that for every top-level type, all its descendants were put in the same micro-task. Each query-type batch was annotated by 6 workers. In the second phase, all candidate types for a query were presented in a single micro-task; candidates include all types that were selected by at least one assessor in phase one, along with their ancestors up to the top level of the hierarchy. Each query was annotated by 7 workers. The Fleiss' Kappa inter-annotator agreement for this phase was 0.71, which is considered substantial [55].

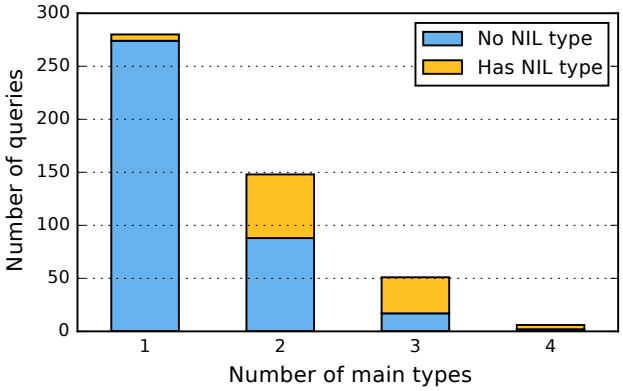


Fig. 4.1 Distribution of the number of main target types in our test collection.

Note that according to our *HTTIv2* task definition, main target types of a query cannot lie on the same path in the taxonomy. To satisfy this condition, if two types were on the same path, we merged the more specific type into the more generic one (i.e., the more generic type received all the "votes" of the more specific one). This affected 120 queries. Figure 4.1 shows the distribution of queries according to the number of main types. In the resulting collection, 280 of all queries (57.73%) have a single target type, while the remainder of them have multiple target types. It is noticeable that as the number of main types increases, so does the proportion of `NIL`-type annotations.

4.4.3 Parameter Settings

For the entity-centric (Section 4.3.1) and type-centric (Section 4.3.2) models, the Language Modeling (LM) approach uses Dirichlet prior smoothing with the smoothing parameter set to 2000; for BM25, we use $k1 = 1.2$ and $b = 0.75$.

For the LTR approach (Section 4.3.3), we employ the Random Forest algorithm for regression as our supervised ranking method. We set the number of trees (iterations) to 1000, and the maximum number of features in each tree, m, to (the ceiling of the) 10% of the size of the feature set.

4.5 Results and Analysis

In the previous chapter, we have presented results using an idealized setting, where target types were provided by an oracle (cf. Section 3.5). We now instead use the methods introduced in Section 4.3 to automatically identify target entity types (Section 4.5.1), and subsequently use these for type-aware entity retrieval (Section 4.5.2).

Method	NDCG@1	NDCG@5
EC, BM25 ($k = 10$)	0.1335	0.2657
EC, LM ($k = 10$)	0.1039	0.2625
TC, BM25	0.2305	0.3216
TC, LM	0.2508	0.3757
LTR	**0.4420**	**0.5968**

Table 4.2 Target entity type identification performance, measured in terms of NDCG@1 and NDCG@5.

4.5.1 Target Entity Type Identification

The research question we seek to answer concerns the automatic identification of target entity types:

- **RQ2a** How can one automatically determine the target entity types of a query from a type taxonomy?

First, we evaluate target entity type identification intrinsically. We follow the literature ([11]) and approach the task as a ranking problem and report on NDCG at rank positions 1 and 5. Also following the literature ([68]), the `NIL`-type labels are ignored in our experimental evaluation. For the LTR method, we used 5-fold cross-validation.

Table 4.2 presents the evaluation results, on basis of which we answer RQ2a. For each of the underlying retrieval models, BM25 and LM, we select only a single entity-centric (EC) method according to the best performing cut-off K (here, $K = 10$ for both models). We find that our supervised learning (LTR) approach significantly and substantially outperforms all baseline methods (with $p < 0.0005$ using a two-tailed paired t-test with Bonferroni correction [137]), in particular, achieving an NDCG@5 score of 0.6.

Analysis of LTR features

We analyze the discriminative power of our features, by sorting them according to their individual information gain, measured in terms of Gini importance. Table 4.3 presents the resulting features order, with their respective information gains; this is also shown as the vertical bars in Fig. 4.2. The top 3 features are: $SIMMAX(t, q)$, $SIMAVG(t, q)$, and $SIMAGGR(t, q)$. This underlines the effectiveness of textual similarity, enriched with distributional semantic representations, measured between the query and the type label. Then, we incrementally add features, one by one, according

Added feature	NDCG@5	Gain
$SIMMAX(t,q)$	0.3672	0.1138
+ $SIMAVG(t,q)$	0.3863	0.1097
+ $SIMAGGR(t,q)$	0.4186	0.1045
+ $ENTITIES(t)$	0.4888	0.0479
+ $EC_{LM,5}$	0.5551	0.0461
+ $EC_{LM,10}$	0.5519	0.0444
+ $EC_{BM25,100}$	0.5555	0.0443
+ $EC_{BM25,50}$	0.5700	0.0417
+ $TC_{BM25}(t,q)$	0.5644	0.0416
+ $EC_{BM25,20}$	0.5757	0.0385
+ $EC_{LM,20}$	0.5661	0.0385
+ $EC_{LM,50}$	0.5782	0.0371
+ $CHILDREN(t)$	0.5905	0.0362
+ $EC_{BM25,10}$	0.5856	0.0340
+ $SIBLINGS(t)$	0.5868	0.0326
+ $EC_{LM,100}$	0.5882	0.0307
+ $IDFSUM(t)$	0.5992	0.0301
+ $EC_{BM25,5}$	0.6003	0.0276
+ $IDFAVG(t)$	0.5924	0.0240
+ $DEPTH(t)$	0.5983	0.0190
+ $JNOUNS(t,q)$	0.5898	0.0185
+ $JTERMS_1(t,q)$	0.5944	0.0178
+ $LENGTH(t)$	0.5967	0.0126
+ $TC_{LM}(t,q)$	0.6001	0.0078
+ $JTERMS_2(t,q)$	0.5980	0.0008

Table 4.3 Performance of our LTR approach, measured by NDCG@5, after incrementally adding features proportional to their individual information gain, measured by Gini score.

to their importance and report on performance in NDCG@5 metric in Table 4.3 (and also shown as the line plot in Fig. 4.2). In each iteration, we set the m parameter of the Random Forests algorithm to 10% of the size of the feature set.

4.5.2 Type-Aware Entity Retrieval

Next, we turn to extrinsic evaluation of the automatically identified target types, by using them for ad hoc entity retrieval (that is, our end-to-end task). (Note that this extrinsic evaluation is similar to the evaluation conducted in the previous chapter, cf. Section 3.4.2, where we used oracle target types.) Here our research question is:

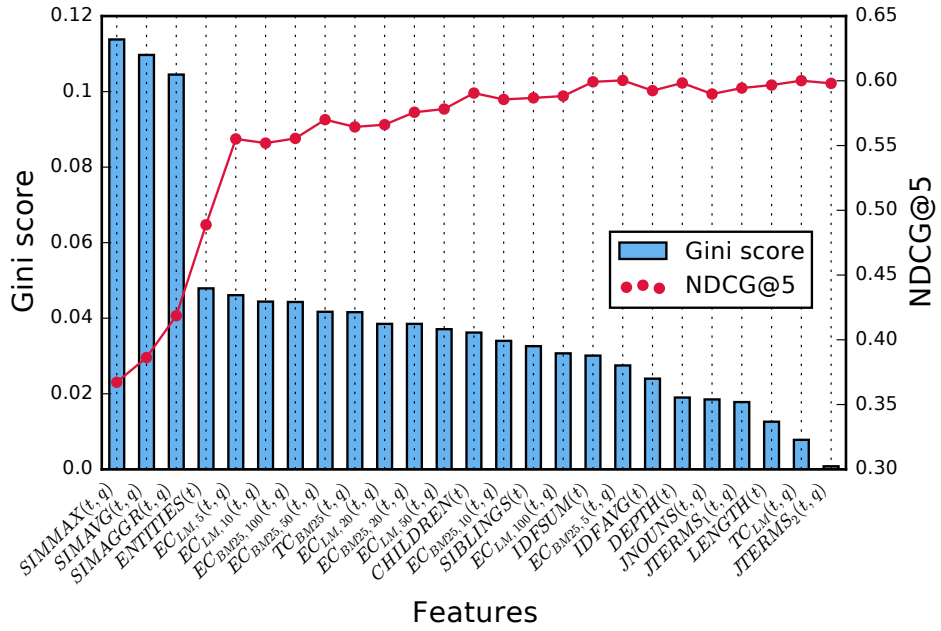


Fig. 4.2 Performance of our LTR approach, measured by NDCG@5, when incrementally adding features according to their individual information gain, measured by Gini score.

- **RQ2b** How does type-aware entity retrieval perform using automatic target entity type identification, compared to an "oracle" setting?

In this section, we present evaluation results for all combinations of retrieval models, type representation modes, and target entity type identification models. We refer to the latter models simply as *identification models* when discussing the results. We then use the term *configuration* now to refer to a particular combination of retrieval model, type representation, and identification model. Throughout this section, we will focus on the 1TT setting (cf. Section 3.4.4).

Table 4.4 shows the results for all configurations. We also present results using the target type labels provided by the human assessors as target entity types, referred to as *Oracle 2*. Additionally, we report on our original oracle as well (which uses the type assignments of relevant results), referred to as *Oracle 1*. Again, our main evaluation metric is NDCG@10, and we also report on NDCG@100. The NDCG@10 scores are also plotted in Fig. 4.3 for an easier visual inspection, with the red line corresponding to the term-based baseline.

For the strict filtering model, we introduce a cut-off parameter k. This parameter controls how many of the highest ranked types we consider as target types. Clearly, the larger the cut-off value k, the more lenient the filtering gets. Hence, we perform a

Model	Strict filtering			Soft filtering		Interpolation		
	@10	@100	k	@10	@100	@10	@100	λ_t
Baseline [132]	0.3036	0.4119	-	0.3036	0.4119	0.3036	0.4119	-
along-path								
EC, BM25	0.3501‡	0.4048	65	**0.3210**	**0.4154**	**0.3214**	**0.4204**	0.30
TC, LM	0.4304‡	0.5043‡	20	0.2678	0.3805	0.3066	0.4116	0.10
LTR	**0.4378**‡	**0.5151**‡	20	0.2988	0.4029	0.3126	0.4150	0.15
Oracle 2	0.4245‡	0.4797‡	-	0.3638	0.4406‡	0.3587‡	0.4384‡	0.40
Oracle 1	0.4600‡	0.5079‡	-	0.4353‡	0.4894‡	0.4172‡	0.4746‡	0.65
top-level								
EC, BM25	0.3501‡	0.4048	65	0.3348†	0.4251†	0.3304‡	0.4226	0.50
TC, LM	0.4295‡	0.5038‡	65	0.3254	0.4159	0.3313†	0.4228	0.55
LTR	**0.4371**‡	**0.5157**‡	5	**0.3705**‡	**0.4453**‡	**0.3748**‡	**0.4460**‡	0.80
Oracle 2	0.4245‡	0.4797‡	-	0.3791‡	0.4477‡	0.3732‡	0.4430‡	0.60
Oracle 1	0.4600‡	0.5079‡	-	0.4196‡	0.4779‡	0.4141‡	0.4725‡	0.70
most specific								
EC, BM25	0.3075	0.3350	65	**0.3048**	**0.4038**	0.3119	**0.4159**	0.15
TC, LM	0.3078	0.3352	65	0.2274	0.3500	0.3036	0.4119	0.00
LTR	**0.3880**‡	**0.4367**	45	0.2997	0.3980	**0.3161**	**0.4159**	0.20
Oracle 2	0.3064	0.4009	-	0.2792	0.3809	0.3185	0.4176	0.15
Oracle 1	0.5092‡	0.5393‡	-	0.4309‡	0.4864‡	0.4075‡	0.4696‡	0.55

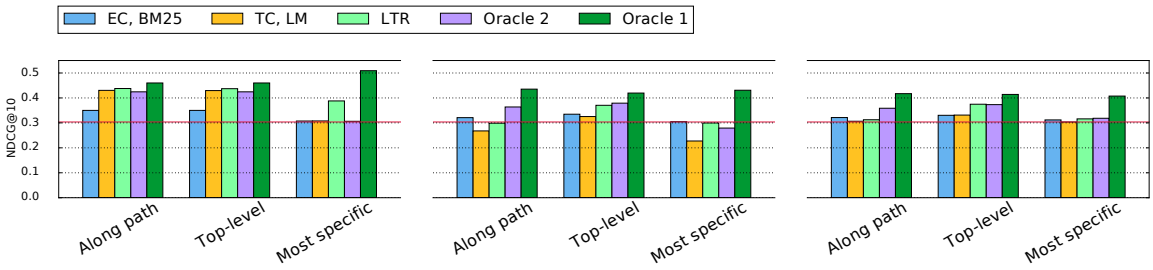
Table 4.4 Entity retrieval performance using automatically identified target types from DBpedia. λ_t and k are the best empirically found interpolation and strict filtering parameters, respectively. For reference comparison, the results using human annotated target type collection ("Oracle 2") and the original "Oracle 1" (cf. Section 3.4.2) are also included. Performance is measured in terms of NDCG@10 and NDCG@100. Statistical significance, tested using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$ and 0.0005, is denoted by † and ‡, respectively.

sweep over the possible values $k \in \{5, 10, 15, \ldots, 100\}$ to calculate $P(q_t|e)$, and use the best performing setting when comparing against other approaches.

Type Representation

We now revisit our research question (RQ1b) about type representation, and answer it for the case when target entity types are automatically identified:

- **RQ2c** What is the impact of the particular choice of type representation on entity retrieval performance with automatically identified target entity types?



(a) Strict filtering (b) Soft filtering (c) Interpolation

Fig. 4.3 Entity retrieval performance for all combinations of retrieval models and type representation modes, using automatically identified target types from DBpedia. The red line corresponds to the term-based baseline. Performance is measured by NDCG@10.

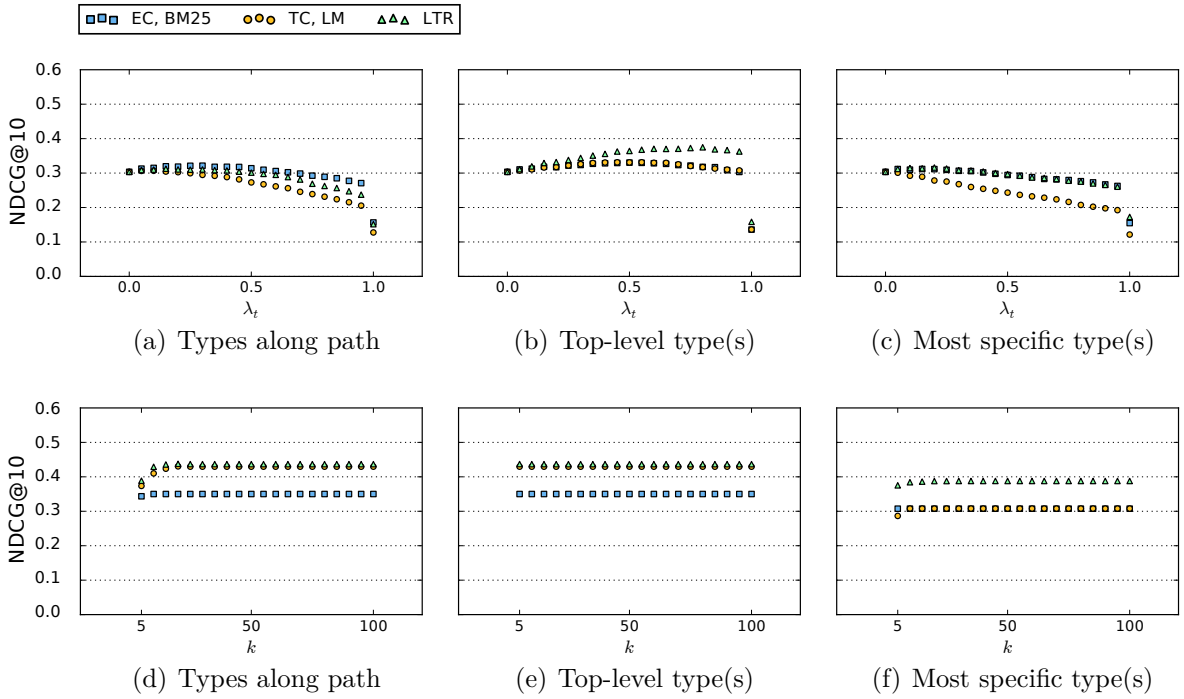
We observe that for all methods, *along-path* and *top-level* representations provide better performance compared to the Most specific representation. The difference between these representations is small, which is similar to our observations using oracle types (cf. Section 3.5.2). The *most specific* representation brings significant improvements only for the LTR method, which is the top performing method in Table 4.2. Overall, our results show that when target entity types are identified automatically, using hierarchical relationships from ancestor types is the most effective way of representing entity type information; keeping only the most specific types is helpful only when an accurate target entity type identification method is employed.

Type-Aware Retrieval Model

Revisiting our research question (RQ1c) about type-aware retrieval model, we can ask an analogous research question in the settings of automatic target entity types:

- **RQ2d** How can one combine term-based and type-based matching for entity retrieval using with automatically identified target entity types?

We observe that strict filtering achieves the best performance among all configurations, and the best results are obtained when it is combined with the LTR identification model. Specifically, by using this retrieval model with the along-path representation, we can outperform the text-only baseline by 44% in terms of NDCG@10. The soft filtering and interpolation models perform best when used with *top-level* representation, significantly outperforming the baseline for the LTR model. Similar to the oracle setting for 1TT (Table 3.5), the λ_t type weight is the highest for *top-level* representation,



(a) Types along path (b) Top-level type(s) (c) Most specific type(s)

(d) Types along path (e) Top-level type(s) (f) Most specific type(s)

Fig. 4.4 Retrieval performance, in terms of NDCG@10, using automatically identified target types from DBpedia. Top: interpolation model with different type weights, λ_t; bottom: strict filtering model with different ranking cut-offs k. The leftmost data points in the top row ($\lambda_t = 0$) correspond to the term-based baseline.

showing that the type component contribution to the interpolation model is high in this configuration.

Target Entity Type Identification Method

The top row of Fig. 4.4 shows the performance of the interpolation model, when varying the value of the λ_t parameter. We observe that assigning any weight greater than 0.2 to type-based information is harmful for the *most specific* and *along-path* representations. On the other hand, when using *top-level* representation, the retrieval performance improves until it reaches a peak (ranging between 0.5 to 0.8), and then it gradually decreases.

The bottom row of Fig. 4.4 presents the retrieval performance while varying the cut-off value k. We find that retrieval performance for the strict filtering model plateaus quickly by increasing the number of ranked types, especially when using the LTR model. This verifies that an effective target entity type identification method, returning

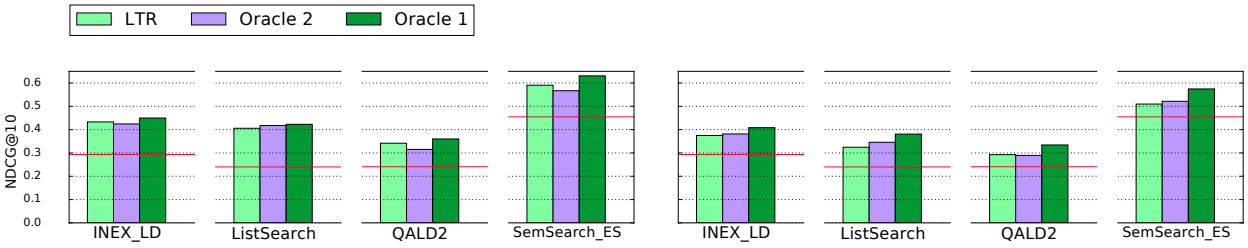
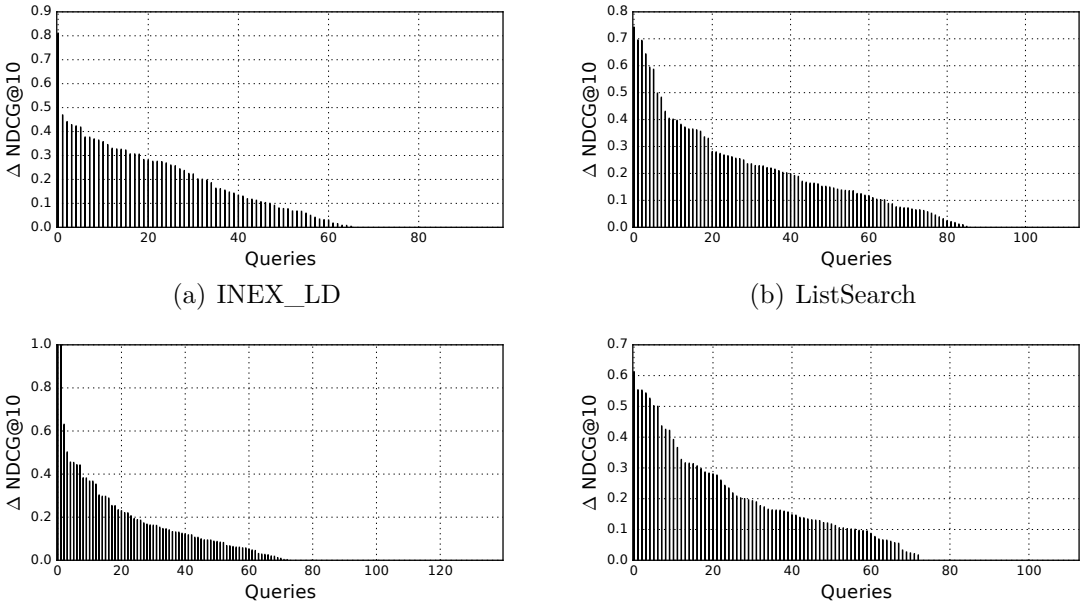


Fig. 4.5 Entity retrieval performance per query category (cf. Table 3.3) for the strict filtering (Left) and soft filtering (Right) retrieval models, using automatically identified top-level target types from DBpedia. The red line corresponds to the term-based baseline. Performance is measured by NDCG@10.



(a) INEX_LD (b) ListSearch

(c) QALD2 (d) SemSearch_ES

Fig. 4.6 Differences in NDCG@10 per query between type-aware entity retrieval and its corresponding (term-based) baseline, using the strict filtering model with top-level DBpedia target types automatically detected by LTR, grouped by query categories (cf. Table 3.3).

relevant types at the top ranks, can bring considerable retrieval improvements using the strict filtering retrieval model.

In order to better understand the effects of automatic target type identification, we break down the entity retrieval results into the four query categories present in the DBpedia-Entity v2 collection (cf. Table 3.3). Figure 4.5 shows retrieval performance per query category for each of the two filtering retrieval models using top-level DBpedia

Δ	Query	Best target type(s)	
		LTR	Oracle 2
Strict filtering, top-level, INEX_LD			
+1.0	INEX_LD-2012335 ("US president authorise nuclear weapons against Japan")	Agent, Device, Event	Agent
+0.55	INEX_LD-20120312 ("tango culture countries")	Place, Topical Concept	Place
+0.5	INEX_LD-20120322 ("tango music instruments")	Topical Concept, Agent, Work	Work
+0.38	INEX_LD-2010057 ("Einstein Relativity theory")	Agent, Work, Place	Work
+0.32	INEX_LD-20120122 ("vietnamese food blog")	Food, Work, Agent	Work
-0.18	INEX_LD-2012369 ("most famous civic military airports")	Place, Agent, Work	Place
-0.19	INEX_LD-2012347 ("seat Florida country Dade")	Place, Agent	Place
-0.23	INEX_LD-20120311 ("tango culture movies")	Work, Agent, Topical Concept	Work
-0.4	INEX_LD-2010043 ("List of films from the surrealist category")	Work, Agent	Work
-0.57	INEX_LD-2010004 ("Indian food")	Food, Agent, Place	Food
Strict filtering, top-level, ListSearch			
+0.52	INEX_XER-96 ("Pure object oriented programing languages")	Work, Agent	-
+0.35	SemSearch_LS-7 ("Branches of the US military")	Agent, Place	Place
+0.32	INEX_XER-99 ("Computer systems that have a recursive acronym for the name")	Work, Device, Agent	Device
+0.29	INEX_XER-125 ("countries have won the FIFA world cup")	Event, Place, Agent	Place
+0.26	INEX_XER-121 ("US presidents since 1960")	Agent, Event	Agent
-0.27	SemSearch_LS-13 ("five great epics of Tamil literature")	Work, Agent, Topical Concept	Work
-0.28	SemSearch_LS-49 ("invented the python programming language")	Work, Agent, Device	Agent
-0.41	SemSearch_LS-3 ("astronauts landed on the Moon")	Agent, Place	Agent
-0.63	SemSearch_LS-14 ("gods dwelt on Mount Olympus")	Agent, Place	Agent
-0.64	TREC_Entity-5 ("Products of Medimmune, Inc")	Agent, Unit Of Work, Chemical Substance	Chemical Substance

Table 4.5 Queries with the largest ΔNDCG@10 differences, using the strict filtering model with top-level DBpedia target types automatically detected by the LTR method, in comparison with target types from human annotators (Oracle 2), grouped per query category. (Here, we only present INEX_LD and ListSearch query categories; see Table 4.6 for the other two categories.) Respective best target type(s), assigned by both the automatic detector and the human annotators, are also shown.

types. We find that type-aware retrieval using the LTR method significantly and consistently outperforms the term-based baseline for all query categories. Moreover, target types identified by our LTR method yield better performance than human type annotations, for three out of the four categories using strict filtering, and for QALD2 (natural language) queries in case of soft filtering.

Furthermore, we conduct an analysis at the query level for each of these query categories. It can be seen in Fig. 4.6 that every category exhibits a similar distribution, with all ΔNDCG@10 values being positive. Specially, the ListSearch category has the largest proportion of improved queries as well as the largest differences. Finally, Tables 4.5 and 4.6 list queries with the largest ΔNDCG@10 differences, either positive

Δ	Query	Best target type(s)	
		LTR	Oracle 2
Strict filtering, top-level, QALD2			
+1.0	QALD2_tr-42 ("are the official languages of the Philippines")	Place, Agent	-
+0.7	QALD2_te-98 ("country does the creator of Miffy come from")	Place, Agent, Work	Place
+0.63	QALD2_te-43 ("all breeds of the German Shepherd dog")	Agent, Species	Species
+0.63	QALD2_te-89 ("In city was the former Dutch queen Juliana buried")	Agent, Species	Place
+0.63	QALD2_te-97 ("painted The Storm on the Sea of Galilee")	Place, Work, Agent	Agent
-0.18	QALD2_te-17 ("all cars that are produced in Germany")	Mean Of Transportation, Work, Agent	Mean Of Transportation
-0.18	QALD2_tr-24 ("mountain is the highest after the Annapurna")	Place, Agent	Place
-0.22	QALD2_te-100 ("produces Orangina")	Agent, Food, Work	Agent
-0.28	QALD2_te-63 ("all Argentine films")	Work, Agent	Work
-0.33	QALD2_te-40 ("List all boardgames by GMT")	Agent, Activity, Event	Activity
Strict filtering, top-level, SemSearch_ES			
+1.0	SemSearch_ES-3 ("Bookwork")	Agent, Place	-
+0.74	SemSearch_ES-26 ("disney orlando")	Place, Agent	Place
+0.42	SemSearch_ES-41 ("joan of arc")	Agent, Place, Work	Agent
+0.39	SemSearch_ES-18 ("canasta cards")	Activity, Work	Activity
+0.34	SemSearch_ES-80 ("sonny and cher")	Work, Agent	Agent
-0.37	SemSearch_ES-67 ("ovguide movies")	Work, Agent	Work
-0.39	SemSearch_ES-79 ("shobana masala")	Agent, Work, Food	Agent
-0.41	SemSearch_ES-125 ("nokia e73")	Device, Place, Work	Device
-0.43	SemSearch_ES-58 ("mason ohio")	Agent, Place	Place
-0.48	SemSearch_ES-4 ("NAACP Image Awards")	Award, Agent, Work	Award

Table 4.6 Queries with the largest ΔNDCG@10 differences, using the strict filtering model with top-level DBpedia target types automatically detected by the LTR method, in comparison with target types from human annotators (Oracle 2), grouped per query category. (Here, we present the other two query categories: QALD2 and SemSearch_ES.) Respective best target type(s), assigned by both the automatic detector and the human annotators, are also shown.

or negative, along with their respective types using automatic detection (LTR) and human annotators (Oracle 2). Comparing with the top-level ancestor(s) of the target types provided by the human oracle, it is clear that in most of the cases the set of LTR-detected query types contains one or more types than the oracle. The additional types relax the filtering criterion by allowing some non-relevant entities to be kept in the ranking and then hurting the performance. Particularly, when the additional type is Agent, the one with largest coverage in the knowledge base, it potentially introduces many entities including all persons and organizations. On the other hand, the positively affected queries result to have more target types detected by LTR. The more lenient filtering given by the additional type(s) allows to retain relevant entities, and even outperforms the contributions of the human oracle target types. The QALD2 category exhibits the distribution of ΔNDCG@10 differences where LTR-detected target types perform the best in comparison with the human oracle. This fact is in line with our

observations in Section 4.5.1, since the most important LTR features, the continuous semantic representations, are more effective in these longer, natural language queries.

4.6 Summary

In realistic scenarios, target entity types are not provided. This chapter has focused on automatically identifying these target types and then leveraging them for entity retrieval. To this end, we have first presented a revision of the hierarchical target type identification task. We have then developed a method for identifying target entity types automatically, which combines a variety of information signals. Our experiments have shown that using automatic target entity types not only improves entity retrieval performance, but also brings the same, or even higher, level of performance achievable by human target type annotations.

Throughout Chapters 3 and 4, we have focused on entity types as a key property of entities for improving entity retrieval. In the next chapter, we will once again leverage entity type information, this time in our quest for understanding and modeling entity-oriented query intents.

Chapter 5

Understanding and Modeling Entity-Oriented Search Intents

As we have mentioned previously, many information needs behind people's searches revolve around specific entities [153]. In the previous chapters, our focus was on improving the ranking of results for these entity-oriented queries by understanding the usage of types of entities targeted by the search query in entity retrieval. Here, we are interested in studying the intents of these queries. Specifically, we wish to answer this research question: (RQ3) *What do entity-oriented queries ask for, and how can they be fulfilled?* As natural units for organizing information, entities can provide not only more focused responses, but often immediate answers [134]. An example of this kind of responses is returning an entity card of Henrik Ibsen for the search query "a doll's house playwright," i.e., related to the entity A Doll's House by the playwright relationship.

A particular kind of entity-oriented queries is more transaction-oriented. Either trying to book a flight or looking for tickets for a concert, just to mention two popular examples, users are often engaged to fulfill information needs by interacting with a third-party service or application. There has been an increasing focus on supporting task-based search [99], and on modeling actionable knowledge; see, e.g., the dedicated vocabulary for actions in the schema.org ontology, and the NTCIR AKG task [22]. These developments display the interest and efforts towards transforming search engines into actions-guided task completion assistants [6]. As in the scenario of a user trying to book a taxi, discussed in Chapter 1 (cf. Fig. 1.4), an envisaged user interface would recognize the service-oriented search intent, and then allow to perform the required transaction, including the elicitation of the travel details and the request for ordering a taxi. Hence, our main research question in this chapter can be refined in subquestions

including: Which of those searches can be fulfilled by looking up direct answers from a knowledge base?, and Which would require to interact with external services?

Most entity-oriented queries consist of an entity name, complemented with context terms, i.e., *refiners*, to express the underlying intent of the user [153]. Examples of these queries are *"the rock movies"* and *"london book a hotel."* Our main objective is to understand entity-related search intents by studying those refiners. In the first part of this chapter, we describe the process we followed in our study of entity-oriented search intents (Section 5.2). Specifically, we represent refiners on the level of entity types. In the Chapters 3 and 4 we have leveraged type-based information to improve entity retrieval. Just like entity types can also aid the disambiguation of known entities and the grouping of emerging ones [140], these type-level characterizations of entity refiners enable knowledge abstraction and generalization. As an example, by representing with *[city]* any entity of the type city, we want to categorize a refiner, e.g., "rentals", in the type-level query *"[city] rentals"*. We then categorize these type-level refiners using an intent classification scheme. Our classification scheme comprises four main categories: property, website, service, and other. After acquiring query suggestions for entities of a given type, they are aggregated to extract type-level refiners. Then, for a representative sample of 50 Freebase types, we collect human annotations for those refiners with respect to the classification scheme we developed. Section 5.3 presents our findings.

A follow-up question we aim to answer is (RQ4): *How can we build a knowledge base of entity-oriented search intents?* In the second part of this chapter, we propose to build IntentsKB, a knowledge base (KB) of entity-oriented search intents, for representing in a structured fashion the main search intents that are commonly associated with a given entity type. These machine-readable statements can be used for automatic querying and reasoning about search intents, and represent one step towards making entities *actionable*. Table 5.1 shows an excerpt from IntentsKB, which represents the intent of customer service for an airline. First, we give a formal definition of the problem of constructing this KB (Section 5.4.1). Then, we propose a pipeline framework to build the knowledge base, consisting of refiner acquisition, refiner categorization, intent discovery, and knowledge base construction stages (Section 5.5). We evaluate the components of our pipeline against editorial judgments, and, using a small seed of labeled data, generate a knowledge base comprising over 30k intents (Section 5.6).

Predicate	Object	Conf.
searchedForType	Aviation/Airline	1
ofCategory	Service	0.866
expressedBy	customer service	0.688
expressedBy	customer care	0.656

Table 5.1 An excerpt from our knowledge base (IntentsKB), for IntentID: `<aviation.airline-65-customer_service>`.

5.1 Related Work

Understanding the query intent constitutes a key factor to improve the search experience [113]. Different general approaches have been proposed, according to a particular interpretation of what it means to understand the query intents, for example, query "aboutness" [170, 33] and query segmentation [160, 17, 86, 77].

Broder's categorization of information needs is broadly accepted and is the most commonly used one for web search, with further refinements [165, 89]. We strive for a similar high-level categorization of intents, but specifically for entity-oriented search queries. Previous work has identified high-level patterns from web search queries. For example, according to Lin et al. [113], a query can be classified as an entity, an entity plus a refiner (e.g., *"emma stone 2017"*), a category, a category plus a refiner (e.g., *"doctors in barcelona"*), a website, or other sort of query. Such classification relies merely on lexico-syntactic forms and lacks a more semantically-grounded distinction.

Search intents have been studied in previous work. Reinanda et al. [159] explore entity aspects in user interaction log data. There, an entity aspect roughly corresponds to a section heading in a Wikipedia article. The authors obtain clusters of entity aspects and also of query refiners from search logs. With these, they address the ranking of intents for a given entity (independently from a query), and the recommendation of aspects. Unlike them, we (i) work on individual query refiners, (ii) model entity intents at the level of types, (iii) consider always a query context of entities, (iv) use approaches defined entirely in absence of log data, and (v) obtain uniquely identified intents.

Actions are a particular kind of entity intent, which we represent using the *service* category. The schema.org ontology is equipped with a dedicated actions vocabulary. Yet, the current schema.org model does not allow to represent actions at the entity type level. Lin et al. [113] propose the model of active objects as a representation of an entity associated with a ranked list of potential actions. These actions are simple surface forms mined from query logs. While our envisaged scenario is the same, we

differ from them in that we (i) model the spectrum of actions at the level of entity types, (ii) do not work with a fixed set of mined actions but rather discover them automatically from query intents, and (iii) obtain structured triples with uniquely identified actions and their surface forms. The *action mining* task at the NTCIR Actionable Knowledge Graph track [21] addresses the problem of ranking potential actions for a given entity of a given type. There, an action comprises of a verb and possibly an object or modifier. While this effort points in the same direction as our work, we note the following limitations: (i) they consider entity type only as input signals, but the output actions are required at the entity level, (ii) they do not account for actions uniquely identified in a KB, and (iii) they force an action to be expressed as a verbal phrase, which is not the case for most of the service-oriented intents we observed. As an example of this last item, *"hilton taxi"* very likely expresses the intent of getting a taxi to the hotel, where the refiner "taxi" is not a verbal phrase.

5.2 Understanding Entity-Oriented Search Intents

This section describes the process we followed for understanding entity-oriented search intents. In this chapter, an *entity-bearing query* is a query that consists of an entity name possibly complemented with a refiner, usually as a suffix. More than just a syntactic complement, a *refiner* is a complementary surface form expressing an underlying user *intent* in relation with the entity. This kind of queries are part of a broader set of *entity-oriented* queries (queries whose expected result is an entity, or list of entities, or a literal entity attribute), that is, queries revolving around entities, like the ones we addressed in the previous chapters. As an example, consider the entity *keens steakhouse* (a restaurant) in the entity-bearing query *"keens steakhouse menu."* The refiner "menu" expresses the intent of reading the restaurant's menu. To understand what these entity-bearing queries ask for, we characterize the refiners on the level of *entity types*, taking natural advantage of an entity type as a semantic class that groups entities together with common characteristics.

Our approach, to be detailed in the next subsections, can be summarized as follows. We collect refiners for a set of prominent entities, and aggregate them across entity types to obtain type-level refiners. Next, we develop a classification scheme of *intent categories*, with a focus on how to fulfill the intent expressed by a type-level refiner. Finally, we annotate a representative sample of entity types with intent categories, and obtain a corpus of prominent type-level refiners assigned to those categories.

5.2.1 Collecting Refiners

We use the type system of Freebase. It is a two-layer categorization system, where types on the leaf level are grouped under high-level domains. Specifically, we use the latest public Freebase dump (2015-03-31), discarding domains meant for administering the Freebase service itself (e.g., `base`, `common`).

We focus on prominent entities, since in this way we benefit from observing a larger and more representative selection of information needs. As the criterion of an entity prominence, we rely on Wikistats page views.[1] This dataset registers the number of times each English Wikipedia article has been requested. We set empirically a prominence threshold of 3,000 page views per article over a span of one year (from June 2015 to May 2016). Given a Freebase type, we select it if it covers at least 100 entities with a prominence above the threshold. Applying these criteria, the selected set contains 634 types.

In a second step, we exploit query suggestions from a major search engine API. This strategy, employed successfully in previous work for various applications [57, 15], is used here to overcome the lack of direct access to past usage data or query logs. Specifically, we collect query suggestions from the Google Suggestions API for at most top 1,000 entities per type according to the above prominence criteria. Then, we replace the name of the entity by its type in each query suggestion. This can be viewed as getting queries where a refiner complements a type. For example, the type-level query *"[travel destination] map"* is obtained from all queries for popular travel destinations, e.g., *"sydney map"* and *"paris map."* Finally, we retain only those refiners that occur in at least 5 suggestions for the given type. This leads to a total of 63,148 distinct type-level refiners for 631 types.

Hereinafter, *type-level refiner*, or simply *refiner*, refers to the query suffix in a type-level query pattern.

5.2.2 Classification Scheme

To address our main goal of understanding entity-related search intents, we need a suitable scheme to classify the entity intents. After a close inspection of the type-level refiners, we define the following scheme of *intent categories*. These categories are focused on how (and from which type of source) the information need can be fulfilled.

- **Property**: The refiner is about getting a specific entity property or attribute that can be looked up in a knowledge base. For example, "children" in the query

[1] https://dumps.wikimedia.org/other/pagecounts-ez/

"*angelina jolie children*," or "opening times" in "*at&t stadium opening times*." The criterion does not require the refiner to exist as a property in an actual knowledge base, but rather its existence to be reasonable.

- **Website**: The refiner looks to reach a specific website or application, which is a rough equivalent of navigational queries in the scheme by Broder [28]. For example, "twitter" in the query "*karpathy twitter*."
- **Service**: The refiner expresses the need to interact with a service, possibly by redirecting to an external site or app. For example, "menu" in the query "*keens steakhouse menu*" would indicate the need to access to an external site for reading the restaurant's menu. As another example, "new album" in "*eric clapton new album*" looks for a service to read about, or listen to, or buy the new album. The interaction would possibly involve further parameters, like "from" and "to" values for "ticket price" in the query "*jpass bullet train ticket price*."
- **Other**: None of the previous ones is applicable. For example, "jr" in the query "*tim hardaway jr*" merely serves to disambiguate the person from other people with the same name.

5.2.3 Annotation

Since it is unfeasible to annotate all types in the knowledge base, we need to sample a set of representative types. From the set of 631 types, we perform stratified sampling as follows. The types are sorted by the total aggregated frequencies of the refiners. We delimit 5 roughly equally-sized intervals by the splitting values of 1,500, 3,000, 6,000, and 8,500 refiners per type; we randomly pick 10 types from each interval. We then annotate data for this final set of 50 representative Freebase types.

Via crowdsourcing, we annotated type-level refiners with intent categories. Specifically, using the Crowdflower (now: Figure Eight) platform, for each annotation instance we presented workers with the query, indicating its entity type and refiner, and asked them to select one of the four intent categories. A total of 5,301 unique instances (type-level refiners) were required to be annotated, each by at least 3 judges (5 at most, if necessary to reach a majority agreement, using dynamic judgments). We paid ¢5 per batch, comprising 11 annotation instances. We ensured quality by requiring a minimum accuracy of 80%, a minimum time of 20 seconds per batch, and a minimum confidence threshold of 0.7. For each type, we only retain an annotated refiner if at least three annotators agreed on the majority category. This leads to a total of 4,490

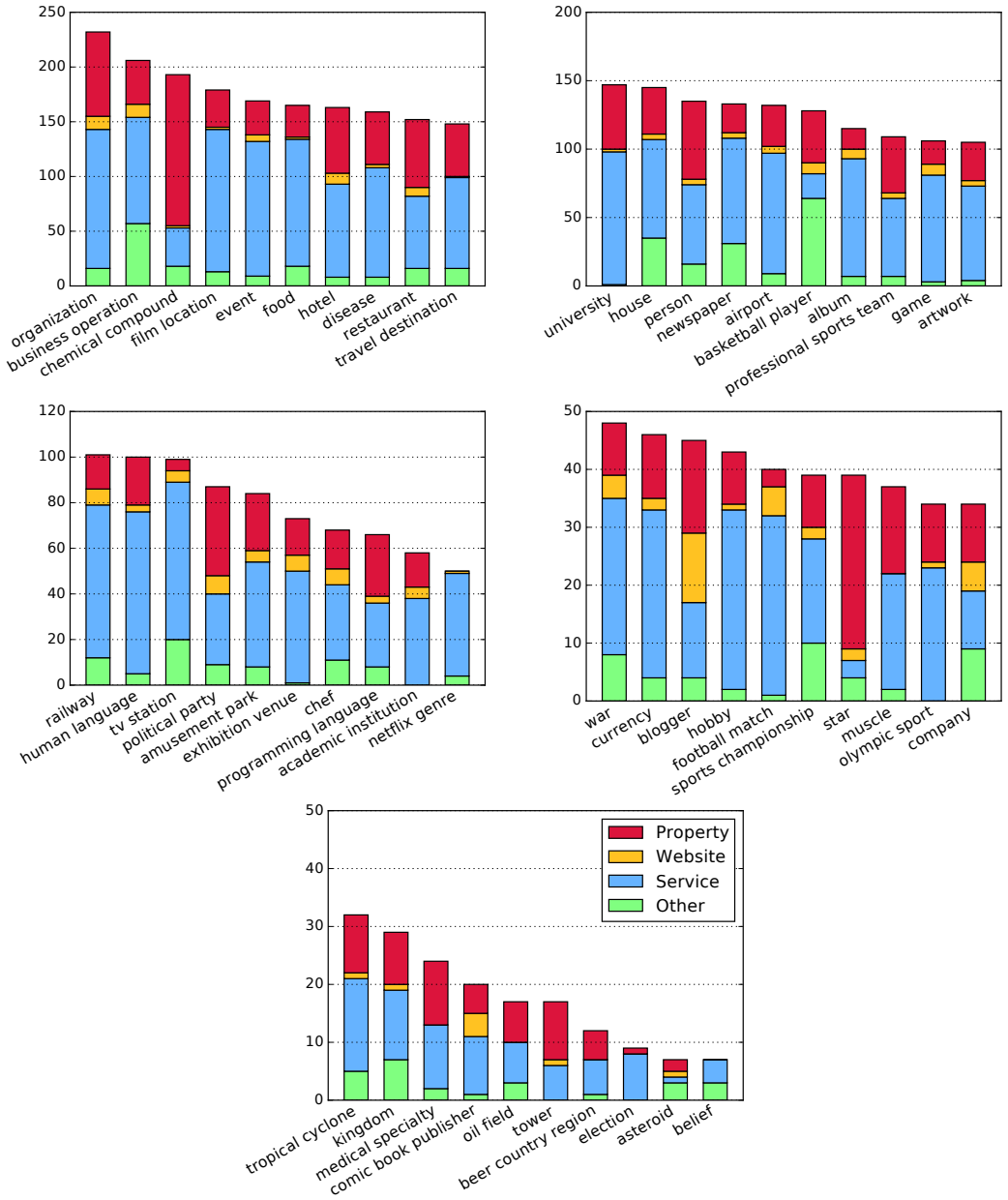


Fig. 5.1 Distributions of intent categories for the sampled types. Note that the y-axis scales differ.

instances, consisting of 2,353 unique refiners. The Fleiss' Kappa coefficient κ for this assessment results to be $\kappa = 0.6503$, which is considered a substantial inter-annotator agreement [55].

5.3 Results and Analysis

Figure 5.1 presents the number of refiners classified per category, for the 50 sampled types, grouped in one plot for each of the 5 intervals of the stratified sampling. Since the final set of types was sampled from types with prominent entities, this ordering, given by the number of refiners, in a way also reflects the prominence of types.

We obtain a distribution of entity intent categories per type after normalizing the frequency of each category by the total number of refiners for that type. From the average proportions in these distributions, we can answer our research question: (RQ3) What do entity-oriented queries ask for, and how can they be fulfilled? A 54.06% of unique entity-oriented queries are to be fulfilled by interacting with some external service or app, meanwhile, 28.6% look for direct answers from a knowledge base. Further, 5.34% of the type-level refiners represent an attempt to reach a website, while 12.08% of them do not fit into any of the previous three categories.

The types with the largest proportion of *service* intents are `netflix_genre` (with refiners, e.g., "videos," "live"), `election` ("map," "polls"), `football_match` ("video," "highlights"), and `music_album`. The *property* intent category covers refiners that are of a more static nature, e.g., `chemical compound` (with refiners like "structural formula," "molecular weight"), `political_party` ("slogan," "president"), `star` ("type of star," "temperature"), or `tower` ("hours," "height"); only the first one is a very prominent type. Most of the entity types exhibit a non-empty proportion of *website* intents. Among all the types, this category exceeds the average proportion, e.g., for `organization`, `business_operation`, `hotel` and `blogger`. The most frequent website refiners in the whole corpus are "wikipedia," "twitter," "facebook," and "youtube." For a few types like `muscle`, `election`, `belief`, or `medical_speciality`, all in the lowest populated groups, no website refiner is present. A marginal proportion of refiners are classified as having the *other* intent. A few exceptional cases with large proportions of other intents are, e.g., `business_operation` and `house` (where the refiner is usually a location), or `basketball_player` (for which many refiners refer mostly to an NBA franchise, e.g., "lakers"). Table 5.2 provides additional examples for a selection of types.

Entity type	Intent category			
	Property	Website	Service	Other
`comic_book_publisher`	logo, address	wiki, website, twitter	submissions, publishing, comics	movies
`tower`	height, address, opening hours	wiki	tickets, restaurant	collapse
`war`	deaths, results, cause	youtube, wikipedia, reddit, quizlet	video, uniforms, pictures, documentary	ap euro, in hindi
`academic_institution`	logo, email, notable alumni	wiki, login, twitter, portal	scholarships, ranking, map, library, jobs	baseball
`automotive_company`	stock, logo, ceo, address	wikipedia, website, linkedin, facebook	parts, careers, investor relations	india, inc
`programming_language`	syntax, ide	wikipedia, wiki, github	jobs, examples, interview questions	3, 2017
`restaurant`	phone number, owner, location	yelp, twitter, app, tripadvisor, groupon	wine list, vouchers, recipes, menu prices	sf, nj, nyc
`music_album`	value, cast, release date	youtube, wikipedia, amazon, imdb	zip download, video, ukulele chords, tracklist	2015, lp
`person`	son, salary, real name	youtube, instagram, snapchat	tour, quotes, photos, new album	sr, now, ww2
`travel_destination`	zip code, train station	craigslist	weather radar, vacation, tours, things to do	today, nj

Table 5.2 Examples of refiners for each intent category, for each (stratified) type group.

5.4 A Knowledge Base of Entity-Oriented Search Intents

We now formally define the problem of constructing a knowledge base of entity-oriented search intents, and design a model for structured representation of this knowledge. As we did for studying them in the first part of this chapter, we propose to represent search intents at the level of entity types. This allows us to capture intents common to many entities, resulting in a representation that is space-efficient and generalizes well to long-tail and emerging entities [140, 113].

5.4.1 Problem Definition

Our goal is to build a knowledge base of entity-oriented search intents (*intents* for short), at the level of entity types. Each search intent is uniquely identified by an *intentID* and is described by an *intent profile*. The KB consists of a set of (*subject*, *predicate*, *object*, *confidence*) quadruples. We formally define the KB as a relational knowledge representation model: $IntentsKB = Q_T \cup Q_C \cup Q_L$, where the three sets of quadruples are partitioned as follows:

- $Q_T \subseteq I \times \{\text{SEARCHEDFORTYPE}\} \times TYPES \times [0, 1]$;

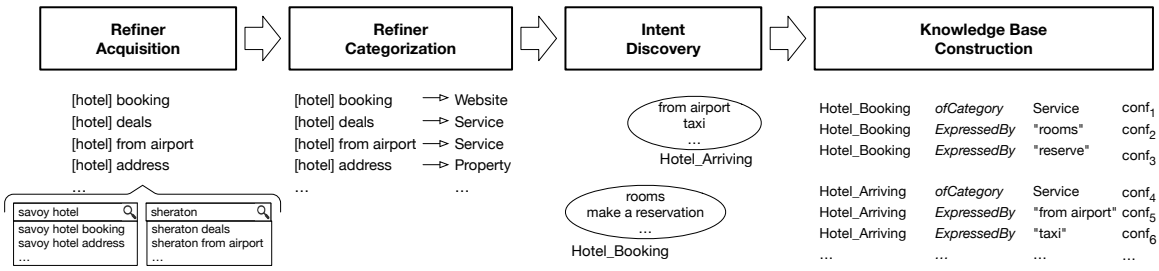


Fig. 5.2 Overview of our proposed framework. Top: pipeline architecture. Bottom: an example through all its stages.

- $Q_C \subseteq I \times \{\text{OFCATEGORY}\} \times CATEGORIES \times [0, 1]$;
- $Q_L \subseteq I \times \{\text{EXPRESSEDBY}\} \times STRINGS \times [0, 1]$.

The set I consists of the unique intent identifiers. SEARCHEDFORTYPE, OFCATEGORY, and EXPRESSEDBY are predicates used for associating an intent with an entity type, intent category, and possible lexicalizations, respectively. $TYPES$ is the set of entity types in the reference KB, and $CATEGORIES$ is a scheme of intent categories (that is, {Property, Website, Service, Other}; cf. Section 5.2.2).

5.5 Proposed Framework

Figure 5.2 displays an overview of our framework for constructing a knowledge base of entity-oriented search intents. It is a pipeline, consisting of four main components.

First, we build upon our previous refiners collection: entity-bearing queries of given types, and from them, refiners aggregated to obtain type-level query patterns (e.g., "*[hotel] booking*" or "*[airline] customer service*"). Second, we classify each of these type-level refiners into our four main *intent categories*, property, website, service, or other. Third, type-level refiners that express the same underlying intent are clustered together. A cluster of refiners corresponds to the various ways of expressing that intent in an actual query (like "*[hotel] booking*," "*[hotel] reservations*," "*[hotel] book a room*," etc.). Fourth, structured representations of intents are created by assigning each intent a unique intentID. This structured representation then contains the type of the entity, the category of the intent (according to the four main intent categories), and the different ways that intent may be expressed. Additionally, each piece of information is assigned a confidence score.

#	Description	Value
Group I - Lexical features		
1	Length of the top result title, in characters	$\{1, \ldots, \infty\}$
2	Avg. length of the 10 result titles, in characters	$\{1, \ldots, \infty\}$
3	Length of the top result snippet, in characters	$\{1, \ldots, \infty\}$
4	Avg. length of the 10 result snippets, in characters	$\{1, \ldots, \infty\}$
5	Number of '/' in the top result URL	$\{0, \ldots, \infty\}$
6	avg. number of '/' in the top 10 result URLs	$\{0, \ldots, \infty\}$
7	size of the URL domains set among all top 10 result URLs.	$\{1, \ldots, 10\}$
8	Jaro distance between the refiner and the top result URL	[0..1]
9	Avg. Jaro distance between the refiner and each of the top 10 result URLs	[0..1]
10	Avg. Jaro distance between all the top 10 result URLs	[0..1]
11–13	{Max., Min., Mean} aggregated term-to-term Jaro distance between the refiner and the top result title	[0..1]
14–16	Avg. {max., min., mean} aggregated term-to-term Jaro distance between the refiner and the top 10 result titles	[0..1]

Table 5.3 Lexical features for refiner categorization.

5.5.1 Refiner Acquisition

In this first stage, we obtain popular type-level query patterns. This stage corresponds to the acquisition of refiners detailed previously in Section 5.2.1.

5.5.2 Refiner Categorization

Next is the assignment of intent categories to refiners. We approach this task, referred to as *refiner categorization*, as a single-class classification problem using supervised learning. Each type-refiner pair constitutes an instance, and the four categories defined above are the possible classes.

Lexical features

We compute attributes at the lexical level of a refiner. Many of these signals are detected in the search results obtained from a major Web search engine. For every type-refiner pair, we take the most prominent entity assigned to that type (according to the entity prominence criteria described in Section 5.2.1). We then search for the corresponding actual entity-bearing query. For example, for the type-level query "*[travel destination] map*," the corresponding entity-bearing query is "united states

map." We exploit the top 10 search results to quantify, among others, the size of the set of their URL domains, or, following Reinanda et al. [159], the average Jaro distance between a refiner and each result URL. Table 5.3 presents the detailed description of the full set of lexical features.

Semantic features

The semantic similarity between a refiner r and a type t is defined as the cosine similarity $cos(\boldsymbol{r}, \boldsymbol{t})$ between the centroid word embedding vectors of the refiner terms ($\boldsymbol{r}$) and type terms ($\boldsymbol{t}$). This measure captures the compositional nature of words in both the type label and the refiner, which was shown to be an effective attribute of a phrase [136]. We use pre-trained word embeddings provided by the word2vec toolkit [135].

5.5.3 Intent Discovery

Once each type-level refiner is mapped to a category as described above, we proceed to discover the intents underlying those refiners. We achieve this by clustering the refiners that express the same user intent. We make use of the intent categories that have been assigned in the previous step, that is, two refiners in the same cluster must have the same intent category. Following the literature ([159]), we apply hierarchical agglomerative clustering (HAC) over the distributional semantic space of the refiners for each type. As before, we use pre-trained word2vec word embeddings. For each refiner, we take its vector if the refiner is in the embedding vocabulary, otherwise we assign it the normalized centroid of the vectors of its terms.

The clusters merging step of HAC is stopped when all the inter-cluster distances are above a certain cut-off threshold. This threshold, $\gamma_{c,t}$, is chosen for each combination of intent category c and entity type t. However, to avoid overfitting, we only learn a single parameter ϵ_c for each intent category, and then set $\gamma_{c,t} = \epsilon_c M_{c,t}$, where $M_{c,t}$ is the maximum distance between any pair of refiners belonging to intent category c and type t. This may be seen as a way of normalization, to account for various cluster sizes. To find the best ϵ_c value, we perform a grid search over [0..1] and pick the value that maximizes the evaluation score against the ground truth clusters, available as training data.

5.5.4 Knowledge Base Construction

In the last step, we construct the full knowledge base representation of intents, i.e., create *intent profiles*. An intent profile i consists of the set of refiners $R(i)$ that were clustered together. Recall that all these refiners have previously been assigned the same intent category, therefore, there exists a single intent category for the profile. The profile is assigned a unique *intentID*. In the interest of readability, it is a concatenation of the entity type, a numerical ID, and the label of the refiner which is closest to the intent centroid.

Recall, that each intent profile has three types of predicates. We define the confidence for each as follows:

- SEARCHEDFORTYPE: since entity type information comes from a curated KB, the confidence in the assigned type is always 1.
- OFCATEGORY: we take the average categorization confidence of the refiners in the profile: $\frac{1}{|R(i)|}\sum_{r\in R(i)}\alpha(r)$, where $\alpha(r)$ is the associated confidence score from the intent categorization step.
- EXPRESSEDBY: the confidence is the similarity between the refiner r and the centroid of all refiners in $R(i)$. In our case, refiners are represented as embedding vectors and cosine is used to measure their similarity (cf. Section 5.5.2).

We also assign a confidence score to the intent profile itself, by taking a linear mixture of the category confidence $\alpha(c)$ and the average refiner confidence $\alpha(r)$:

$$\alpha(i) = \frac{1}{2}\Big(\alpha(c) + \frac{1}{|R(i)|}\sum_{r\in R(i)}\alpha(r)\Big)\ . \tag{5.1}$$

5.6 Experimental Evaluation

This section presents the evaluation of our approach, aiming to answer the main research question: (RQ4) How can we build a knowledge base of entity-oriented search intents?

5.6.1 Refiner Categorization

Every instance in this component is, as before, a type-refiner pair, collected in the first stage of our pipeline (cf. Section 5.5.1). Our dataset consists of 4,490 training instances (cf. Section 5.2.3), each labelled with one of the four intent categories. This comprises

Feature Set	Accuracy
Feature group I (lexical)	0.5920
Feature group II (semantic)	**0.7024**‡
Combined (I + II)	0.6150

Table 5.4 Refiner categorization results, in terms of accuracy. Statistical significance is tested against the combined feature set (line 3), using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.001$, denoted by ‡.

Category	Homogeneity	Completeness	V-measure
Oracle categories	0.9494	0.7270	0.8201
Automatic categorization	0.8872	0.6872	0.7710

Table 5.5 Clustering refiners using oracle vs. automatically assigned intent categories, measured in terms of homogeneity, completeness, and V-measure scores.

all the instances corresponding to the 50 sampled types. For obtaining the search results exploited by many features, we utilize the Google Search API. In a preliminary round, we experimented with a variety of classifiers. Results are reported only for the best performing classifier, which is Random Forests. We used the following parameters: the number of trees is 100 and the maximum depth of the trees is approximately the square root of the feature set used in that setting. We train the model using five-fold cross-validation. Table 5.4 reports the results measured by accuracy. We find that semantic features perform much better than lexical features. When combining the two, the resulting performance is still significantly inferior to using semantic features alone. Therefore, we only use the semantic feature group in the remainder of our experiments. The final model is applied to the 60,795 test instances corresponding to the remaining 581 types.

5.6.2 Intent Discovery

In order to construct ground truth clusters, two expert annotators manually grouped all the refiners for each of the 50 types in the refiner categorization dataset. The annotators worked independently and then discussed the conflicting cases to reach perfect agreement. We evaluate the clustering approach in two evaluation settings: (i) an "oracle" setting, which uses the ground truth intent categories, and (ii) a realistic setting, where the categories are automatically assigned by our refiner categorization component. We perform five-fold cross-validation for each evaluation instance (type-refiner pair),

with the same partition of folds used for evaluating refiner categorization. The training folds are used to optimize ϵ_c for each category as described in Section 5.5.3. The clusters obtained for a given type are "flattened" by ignoring the intent categories, and are evaluated against the ground truth clusters flattened in the same way. This serves to eliminate, to some extent, errors that originate from incorrect refiner categorization. Table 5.5 presents the results, measured in terms of three clustering evaluation metrics: homogeneity (which, intuitively, assesses that each cluster contains only members of a single class), completeness (i.e., all members of a given class are assigned to the same cluster), and their harmonic mean called V-measure [166]. We find that our automatic setting can achieve a V-measure that is only about 6% lower than using oracle categories.

5.6.3 Fact Validation

In this last part, we evaluate the end-to-end approach by applying the pipeline, which we trained on 50 types (cf. Section 5.2.3), on the remaining 581 types. As a result, we obtain 31,724 intent profiles that comprise a total of 155,967 quadruples. We proceed to estimate its expected overall quality by taking a stratified sample from the generated quadruples. Considering the confidence scores associated with the profiles (cf. Eq. (5.1)), we partition the range [0..1] into 5 equally-sized buckets. For each bucket, we take 25 random types without repetition and select 5 intent profiles from each type. The resulting sample has a total of 2,010 quadruples (around 1.29% of the size of the KB). For the following experiment, we ignore the confidence scores, so as not to bias annotators. Thus, we refer to triples, not quadruples from now on.

We conduct an annotation experiment to decide whether a triple is correct w.r.t. its intent profile. Three expert annotators each manually labeled the sampled triples. Note that for every profile, the searchedForType triple is trivially correct. Also, if there is only a single expressedBy triple in the profile, then it is trivially correct. We measure the inter-annotator agreement using Fleiss' Kappa coefficient κ [55], separately on ofCategory and expressedBy triples. For the set of ofCategory triples, $\kappa = 0.2206$, indicating a fair agreement; for expressedBy triples, $\kappa = 0.8606$, thus it is almost perfect. We take the majority vote of the annotators as the ground truth. Among the ofCategory triples, 88% of them are correct. As for the expressedBy triples, 42% of them are correct. Over all the triples, 54% of them results to be correct (ignoring the trivially correct searchedForType triples).

We then ask the question: (RQ4a) How well do the estimated confidence scores correspond to the actual correctness of triples? Figure 5.3 displays the number of

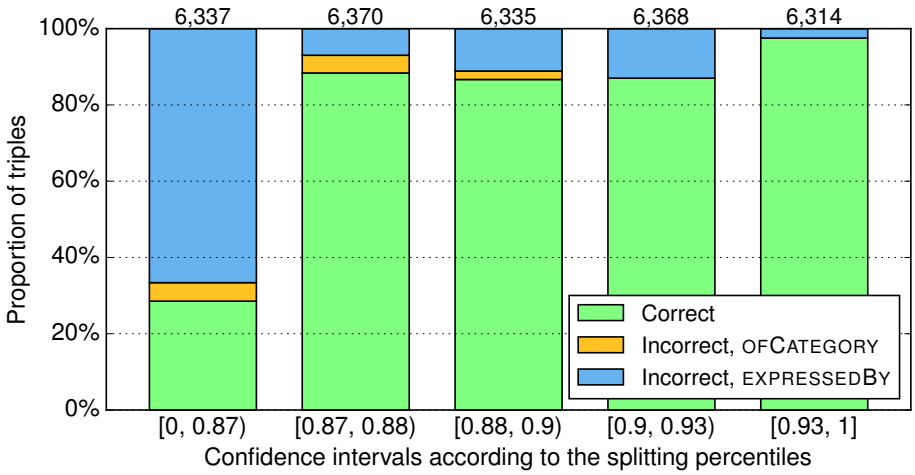


Fig. 5.3 Proportion of triples in the annotated sample (y-axis), and number of intent profiles in the KB (on top of each bar), per confidence bucket.

correct and incorrect triples with a break down per confidence bucket. We find that, indeed, the higher the associated confidence score, the more likely it is that the triple is correct. It is worth pointing out that accuracy generally is quite high. Apart from the leftmost bucket, triples overall have over 90% accuracy based on the manually labeled sample.

5.7 Summary

The study performed in this chapter has lead to a better understanding of what entity-oriented queries ask for. We have developed a classification scheme to categorize entity-oriented search intents and annotated a representative sample of type-level refiners using this scheme. We have found that, on average, more than a half of the queries are to be fulfilled by interaction with services; also, a large proportion of information needs look for direct answers from a knowledge base.

In this chapter we have also addressed the problem of constructing a knowledge base of entity-oriented search intents and proposed a pipeline approach. We have performed an experimental evaluation on the level of individual components as well as on the end-to-end task. Using a sample of 4.5k labeled instances for 50 types as training data, we have generated a knowledge base of over 30k intents for almost 600 unseen types.

The first two grand themes of this thesis focus on entity-oriented queries. Whereas in Chapters 3 and 4 we have investigated entity retrieval models that leverage type-based information identified in entities and in queries, in this chapter we have achieved

a better understanding of the search intents in these queries by studying their refiners on the level of entity types, as well as we represented them in a structured fashion. Moving into the ending theme of this work, the next chapter will present our study on query suggestions for supporting task-based search.

Chapter 6

Suggesting Queries to Support Task-Based Search

Query suggestions, recommending a list of related queries to an initial user input, are an integral part of modern search engines [144]. Accordingly, this task has received considerable attention, particularly over the last decade [138, 18, 32]. Search, however, is often performed in the context of some larger underlying task [99], and traditional approaches for suggesting queries do not consider the task behind the initial query. There is a growing stream of research aimed at making search engines more task-aware (i.e., recognizing what task the user is trying to accomplish) and customizing the search experience appropriately. In this chapter, we focus our attention on one particular tool for supporting task-based search: query suggestions.

We envisage a user interface where task-based query suggestions are presented once the user has issued an initial query. As illustrated in Fig. 6.1, these query suggestions actually come in two flavors: *query completions* and *query refinements*. The difference is that the former are prefixed by the initial query, while the latter are not. Note that this is different from query autocompletion, which tries to recommend various possible completions while the user is still typing the query. The task-aware query suggestions we propose are intended for exploring various aspects (subtasks) of the given task after inspecting the initial search results. Selecting them would allow the user to narrow down the scope of the search. The Tasks track at the Text REtrieval Conference (TREC) has introduced an evaluation platform for this very problem, referred to as *task understanding* [200]. Specifically, given an initial query, the system should return a ranked list of keyphrases "that represent the set of all tasks a user who submitted the query may be looking for" [200]. The goal is to provide a complete coverage of subtasks for an initial query, while avoiding redundancy. We use these keyphrases

(a) Query completion (b) Query refinement

Fig. 6.1 Query suggestions to support task-based search.

as *query suggestions*. The main research question that we address in this chapter is: (RQ5) *How can we generate query suggestions for supporting task-based search?*

Following the pipeline architecture widely adopted in general query suggestion systems [138, 175], we propose a two-step approach consisting of *suggestion generation* and *suggestion ranking* steps. In the first part of this chapter, our aim is to generate suggestions in a setting where past usage data and query logs are not available or cannot be utilized. This would be typical for systems that have a smaller user base (e.g., in the enterprise domain) or when a search engine has been newly deployed [18]. One possibility is to use query suggestion APIs, which are offered by all major web search engines. These are indeed one main source type we consider. Additionally, we can use the initial query to search for relevant documents, and extract keyphrases from search snippets and from full text documents. Finally, given the task-based focus of our work, we lend special treatment to the WikiHow site,[1] which is an extensive database of how-to guides. In a first part we aim to address task-based query suggestion in an end-to-end fashion regarding the two-step pipeline. For this, we approach it with a probabilistic generative framework that consists of four components: source importance, document importance, keyphrase relevance, and query suggestion generation. We propose different estimators for these components, and experimentally compare them using the datasets from the TREC Tasks track.

Later in this chapter, we focus on the first step of the main pipeline approach, namely suggestion generation, dedicated to creating suggestion candidates. There, our study is interested in additional challenges: Can a unified method produce both query completions and query refinements?, and, Is it possible to do it so without relying on resources such as a major search engine suggestions or query logs? We consider

[1] http://www.wikihow.com/

several methods and multiple information sources, as well as we collect a large number of annotations via crowdsourcing.

A review of related literature is presented in Section 6.1. Section 6.2 details the statement of the task understanding problem. We propose an architecture that uses generative probabilistic modeling for extracting keyphrases from a variety of sources and generating query suggestions from them (Section 6.3). Making use of the benchmark datasets of the 2015 and 2016 editions of the TREC Tasks track, we provide a detailed analysis of the components of our framework using different estimation methods (Section 6.4). Later in Section 6.5 we address the generation of high-quality query suggestion candidates, for which we develop a test collection and perform an experimental evaluation of our approach (Section 6.6).

6.1 Related Work

There is a large body of work on understanding and supporting users in carrying out complex search tasks. Log-based studies have been one main area of focus, including the identification of tasks and segmentation of search queries into tasks [118, 82] and mining task-based search sessions in order to understand query reformulations [91] or search trails [195].

Another theme is supporting exploratory search, where users pursue an information goal to learn or discover more about a given topic. Recent research in this area has brought the importance of support interfaces into focus [5, 183, 6].

Our main interest is in query suggestions, as a distinguished support mechanism. Most of the related work utilizes large-scale query logs. For example, Craswell and Szummer [39] perform a random walk on a query-click graph. Boldi et al. [23] model the query flow in user search sessions via chains of queries. Scenarios in the absence of query logs have been addressed [104, 18], where query suggestions are extracted from the document corpus. However, their focus is on query auto-completion, representing the completed and partial terms in a query. Kelly et al. [100] have shown that users prefer query suggestions, rather than term suggestions. We undertake the task of suggesting queries to users, related to the task they are performing, as we shall explain in the next section.

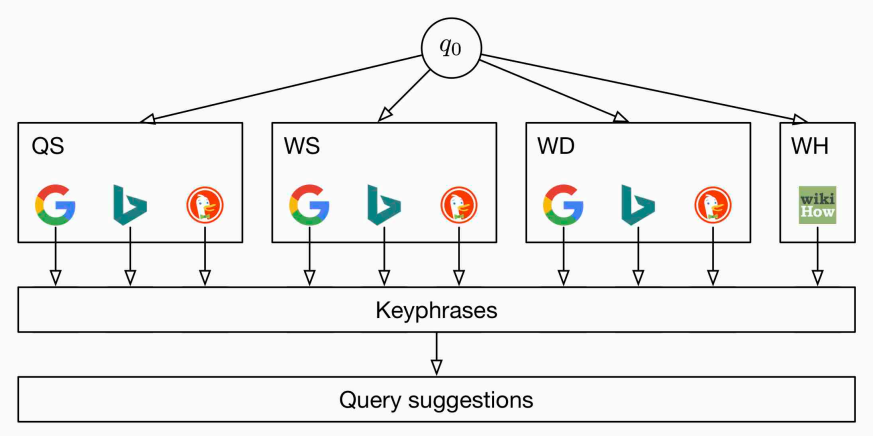


Fig. 6.2 High-level overview of our approach.

6.2 Problem Statement

Throughout this chapter, we adhere to the problem definition of the task understanding task of the TREC Tasks track [200]. Given an initial query q_0, the goal is to return a ranked list of query suggestions $\langle q_1, \ldots q_n \rangle$ that cover all the possible subtasks related to the task the user is trying to achieve. In addition to the initial query string, the entities mentioned in it are also made available (identified by their Freebase IDs).

For example, for the query "low wedding budget," subtasks include (but are not limited to) "buy a used wedding gown," "cheap wedding cake," and "make your own invitations." These subtasks have been manually identified by the track organizers based on information extracted from the logs of a commercial search engine. The suggested keyphrases are judged with respect to each subtask on a three point scale (non-relevant, relevant, and highly relevant). Note that subtasks are only used in the evaluation, these are not available when generating the keyphrases.

6.3 Approach

We now present our approach for generating query suggestions. As Figure 6.2 illustrates, we obtain keyphrases from a variety of sources, and then construct a ranked list of query suggestions from these. Many systems that participated in the TREC Tasks track have relied on strategic combinations of different sources to produce query suggestions [75, 62, 76]. However, no systematic comparison of the different source types has been performed yet—we fill this gap. Additional components include estimating a

document's importance within a given source, extracting keyphrases from documents, and forming query suggestions from these keyphrases.

6.3.1 Generative Modeling Framework

We introduce a generative probabilistic model for scoring the candidate query suggestions according to $P(q|q_0)$, i.e., the probability that a query suggestion q was generated by the initial query q_0. Formally:

$$\begin{aligned} P(q|q_0) &= \sum_s P(q|q_0, s)P(s|q_0) \\ &= \sum_s \big(\sum_d P(q|q_0, s, d)P(d|q_0, s)\big)P(s|q_0) \\ &= \sum_s \big(\sum_d \Big(\sum_k P(q|q_0, s, k)P(k|s, d)\Big)P(d|q_0, s)\big)P(s|q_0) \ . \end{aligned}$$

This model has four components: (i) $P(s|q_0)$ expresses the importance of a particular information source s for the initial query q_0; (ii) $P(d|q_0, s)$ represents the importance of a document d originating from source s, with respect to the initial query; (iii) $P(k|d, s)$ is the relevance of a keyphrase k extracted from a document d from source s; and (iv) $P(q|q_0, s, k)$ is the probability of generating query suggestion q, given keyphrase k, source s, and the initial query q_0. Below, we detail the estimation of each of these components.

6.3.2 Source Importance

We collect relevant information from four kinds of sources: query suggestions (QS), web search snippets (WS), web search documents (WD), and WikiHow (WH). For the first three source types, we use three different web search engines (Google, Bing, and DuckDuckGo), thereby having a total of 10 individual sources. In this work, we assume conditional independence between a source s and the initial query q_0, i.e., set $P(s|q0) = P(s)$.

6.3.3 Document Importance

From each source s, we obtain the top-K ($K = 10$) documents for the query q_0. We propose two ways of modeling the importance of a document d originating from s: (i) uniform and (ii) inversely proportional to the rank of d among the top-K documents,

that is:

$$P(d|q_0, s) = \frac{K - r + 1}{\sum_{i=1}^{K}(K - i + 1)} = \frac{K - r + 1}{K(K + 1)/2} \quad ,$$

where r is the rank position of d ($r \in \{1, ..., K\}$).

6.3.4 Keyphrase Relevance

We obtain keyphrases from each document, using an automatic keyphrase extraction algorithm. Specifically, we use the RAKE keyword extraction system [124]. For each keyphrase k, extracted from document d, the associated confidence score is denoted by $c(k, d)$. Upon a manual inspection of the extraction output, we introduce some data cleansing steps. We only retain keyphrases that meet these selection criteria: (i) they have an extraction confidence above an empirically set threshold of 2; (ii) they are at most 5 terms long; (iii) each of the terms has a length between 4 and 15 characters, and is either a number of max. 4 digits or a term (which is not a term in {"–", "ll", "/", "javascript", "wikihow", "quora", "google"}, nor it starts with a character in {"-", "<", "/", "jpg"}, nor it contains the substring "&amp"); (iv) the first term in the keyphrase is not "-" or "/;" and (v) the keyphrase is not "[citation needed]" or "–." Finally, we set the relevance of k as $P(k|d, s) = c(k, d)/\sum_{k'} c(k', d)$.

In case s is of type QS, each returned document actually corresponds to a query suggestion. Thus, we treat each of these documents d as a single keyphrase k, for which we set $P(k|d, s) = 1$.

6.3.5 Query Suggestions

In the final step, we need to generate query suggestions from the extracted keyphrases. As a baseline option, we take each raw keyphrase k as-is, i.e., with $q = k$ we set $P(q|q_0, s, k) = 1$.

Alternatively, we can form query suggestions by *expanding keyphrases*. Here, k is combined with the initial query q_0 using a set of expansion rules: (i) adding k as a suffix to q_0; (ii) adding k as a suffix to an entity e mentioned in q_0; and (iii) using k as-is. In particular, rule (ii) aims to leverage the entity mentioned in the initial query, which is provided as part of the dataset for the task understanding task (cf. Section 6.2). If the initial query contains multiple entities, then (ii) is performed for each of the mentioned entities.

Each query suggestion q, that is generated from keyword k, has an associated confidence score $c(q, q_0, s, k)$, based on which rule generated it:

$$c(q, q_0, s, k) = \begin{cases} \alpha, & \text{if } q = q_0 \oplus k \text{ (i.e., rule (i))} , \\ \beta, & \text{if } q = e \oplus k \text{ (i.e., rule (ii))} , \\ \gamma, & \text{otherwise} , \end{cases} \tag{6.1}$$

where $\alpha + \beta + \gamma = 1$. When a keyphrase could be generated by multiple rules, we take the one with the highest weight.

Rules (i) and (ii) further involve a custom string concatenation operator, $\oplus$, which first removes the longest common subphrase from the right-side concatenation term. For example: "*aa bb*" $\oplus$ "*bb cc*" = "*aa bb cc*" and *"choose bathroom decor"* $\oplus$ *"bathroom decor style"* = *"choose bathroom decor style."* Such a removal strategy is motivated by the fact that in English most of the usual (non-adjectival) refinements are syntactically performed as an addition to the right.

We then set $P(q|q_0, s, k) = c(q, q_0, s, k) / \sum_{q'} c(q', q_0, s, k)$. By conditioning the suggestion probability on s, it is possible to apply a different approach for each source. Like in the previous subsection, we treat sources of type QS distinctly, by simply taking $q = k$ and setting $P(q|q_0, s, k) = 1$.

We note that it is possible that multiple query suggestions have the same final probability $P(q|q_0)$. We resolve ties using a deterministic algorithm, which orders query suggestions by length (favoring short queries) and then sorts them alphabetically.

6.4 Task Understanding Results

In this section we present our experimental setup and results.

6.4.1 Experimental Setup

We use the test suites of the TREC 2015 and 2016 Tasks track [200, 188]. These contain 34 and 50 queries with relevance judgments for suggestions, respectively. We report on the official evaluation metrics used at the TREC Tasks track, which are ERR-IA@20 and α-NDCG@20. In accordance with the track's settings, we use ERR-IA@20 as our primary metric. (For simplicity, we omit mentioning the cut-off rank of 20 in all the table headers.) We noticed that in the ground truth the initial query itself has been judged as a highly relevant suggestion in numerous cases. We removed these cases, as they make little sense for the envisioned scenario; we note that this might lead to a

$P(q\|q_0, s, k)$	S	2015		2016	
		ERR-IA	α-NDCG	ERR-IA	α-NDCG
Using raw keyphrases	QS	0.0755	0.1186	**0.4114**	**0.5289**
	WS	**0.2011**	**0.2426**	0.3492	0.4038
	WD	0.1716	0.2154	0.2339	0.2886
	WH	0.0744	0.1044	0.1377	0.1723
Using expanded keyphrases	QS	0.0751	0.1182	**0.4046**	**0.5233**
	WS	**0.1901**	**0.2274**	$0.2927^{\dagger}$	$0.3467^{\dagger}$
	WD	0.1551	0.2097	$0.1045^{\ddagger}$	$0.1667^{\ddagger}$
	WH	0.0849	0.1090	$0.0789^{\dagger}$	$0.0932^{\dagger}$

Table 6.1 Comparison of query suggestion generators across the different types of sources. Performance is measured in terms of ERR-IA@20 and α-NDCG@20. Statistical significance is tested against the corresponding line in the top block, using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$ and $p < 0.0005$, denoted by † and ‡, respectively.

drop in terms of performance. We report on statistical significance using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$ and $p < 0.0005$.

In a series of experiments, we evaluate each component of our approach, in a bottom-up fashion. For each query set (i.e., 2015 and 2016 versions of the TREC Tasks track), we pick the configuration that performed best on that query set, which is an idealized scenario. Note that our focus is not on absolute performance figures, but on answering the following research questions:

RQ5a What are the most useful information sources?

RQ5b What are effective ways of (i) estimating the importance of documents and (ii) generating query suggestions from keyphrases?

RQ5c Are our findings consistent across the two query sets?

6.4.2 Query Suggestion Generation

We start our experiments by focusing on the generation of query suggestions and compare the two methods described in Section 6.3.5 (as-is versus expanded keyphrases). The document importance is set to be uniform in this experiment. We report performance separately for each of the four source types S (that is, we set $P(s)$ uniformly among sources $s \in S$ and set $P(s) = 0$ for $s \notin S$). After testing estimations with different weight combinations, we set the weights of expansion rules empirically to be

$P(d\|q_0, s)$	S	2015		2016	
		ERR-IA	α-NDCG	ERR-IA	α-NDCG
Uniform	QS	0.0755	0.1186	**0.4114**	**0.5289**
	WS	**0.2011**	**0.2426**	0.3492	0.4038
	WD	0.1716	0.2154	0.2339	0.2886
	WH	0.0849	0.1090	0.1377	0.1723
Rank-based decay	QS	0.0891†	0.1307†	**0.4288**	**0.5455**
	WS	**0.1906**	**0.2315**	0.3386	0.4011
	WD	0.1688	0.2119	0.1964	0.2608
	WH	0.0935	0.1225	0.1195	0.1495

Table 6.2 Comparison of document importance estimators across the different types of sources. Performance is measured in terms of ERR-IA@20 and α-NDCG@20. Statistical significance is tested against the corresponding line in the top block, using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$, denoted by †.

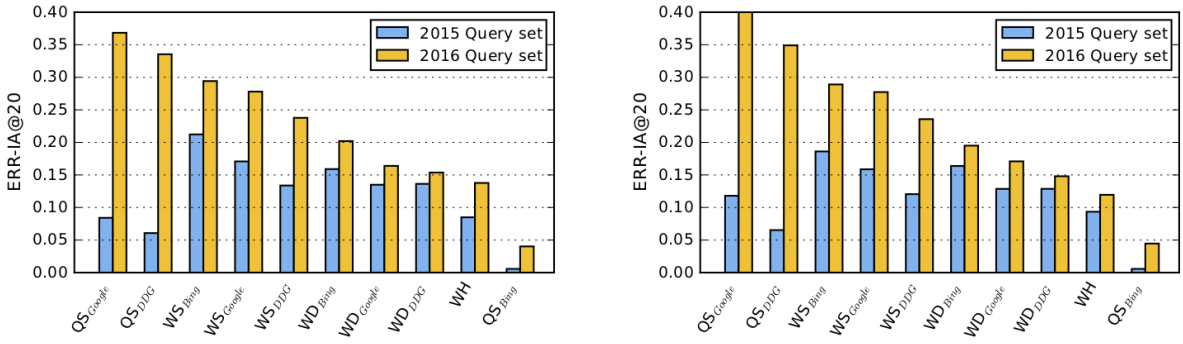
$\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.5$. Table 6.1 presents the results. It is clear that, with a single exception (2015 WH), it is better to use the raw keyphrases, without any expansion. The differences are significant on the 2016 query set for all source types but QS. In particular, we do not find the contribution of the entity mentioned in the initial query to be useful in the set of expansion rules.

Regarding the comparison of different source types, we find that QS > WS > WD > WH on the 2016 query set, meanwhile for 2015, the order is WS > WD > QS, WH.

6.4.3 Document Importance

Next, we compare the two document importance estimator methods, uniform and rank-based decay (cf. Section 6.3.3), for each source type. Table 6.2 reports the results. We find that rank-based document importance is beneficial for the query suggestion (QS) source types, for both years, and for WikiHow (WH) on the 2015 topics. However, the differences are only significant for QS 2015. For all other source types, the uniform setting performs better.

We also compare performance across the 10 individual sources. Figure 6.3(a) shows the results, in terms of ERR-IA@20, using the uniform estimator. We observe a very similar pattern using the rank-based estimator (Fig. 6.3(b)). On the 2016 query set, the individual sources follow the exact same patterns as their respective types (i.e., QS > WS > WD > WH), with one exception. The Bing API returned an empty set of search suggestions for many queries, hence the low performance of QS_{Bing}. We can



(a) Uniform (b) Rank-based

Fig. 6.3 Performance of individual sources, in terms of ERR-IA@20, sorted by performance on the 2016 query set, using (a) uniform and (ii) rank-based document importance estimator.

$P(s)$	2015		2016	
	ERR-IA	α-NDCG	ERR-IA	α-NDCG
Uniform	0.2219	0.2835	0.4561	0.5793
Source-type	0.2381	0.2905	**0.4570**	**0.5832**
Individual	**0.2518**†	**0.3064**	0.4562	**0.5832**

Table 6.3 Combination of all sources using different source importance estimators. Performance is measured in terms of ERR-IA@20 and α-NDCG@20. Statistical significance is tested against the uniform setting (line 1), using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.025$, denoted by †.

observe a similar pattern on the 2015 topics, with the exception of sources of type QS, which are the least effective here.

6.4.4 Source Importance

Finally, we combine query suggestions across different sources; for that, we need to set the importance of each source. We consider three different strategies for setting $P(s)$: (i) uniformly; (ii) proportional to the importance of the corresponding source type (QS, WS, WD, and WH) from the previous step (cf. Table 6.2); (iii) proportional to the importance of the individual source (cf. Figure 6.3). The results are presented in Table 6.3. Firstly, we observe that the combination of sources performs better than any individual source type on its own. As for setting source importance, on the 2015 query set we find that (iii) delivers the best results, which is in line with our expectations.

On the 2016 query set, only minor differences are observed between the three methods, none of which are significant.

6.4.5 Summary of Findings

Answering RQ5a, we observe that query suggestions provided by major web search engines are unequivocally the most useful information source on the 2016 queries. We presume that these search engine suggestions are already diversified, which we can directly benefit from. These are followed, in order, by keyphrases extracted from (i) web search snippets, (ii) web search results, i.e., full documents, and (iii) WikiHow articles. On the 2015 query set, query suggestions proved much less effective; see RQ5c below.

We answer RQ5b by noting that using the raw keyphrases, as-is, performs, with a single exception, better than expanding them by taking the original query into account. For web query suggestions it is beneficial to consider the rank order of suggestions, while for web search snippets and documents the uniform setting performs better. For WikiHow, it varies across query sets.

Our main observations for answering RQ5c are consistent across the 2015 and 2016 query sets, regarding document importance estimation and suggestion generation methods. It is worth noting that some of our methods were officially submitted to TREC 2016 [62] and were included in the assessment pools. This is not the case for 2015, where many of our query suggestions are missing relevance assessments (and, thus, are considered irrelevant). This might explain the low performance of QS sources on the 2015 queries.

6.5 Generating Query Suggestion Candidates

Our goal is to investigate query suggestions as a tool for supporting task-based search. Specifically, we address the task-based query suggestion problem stated in Section 6.2. As mentioned before, and illustrated in Fig. 6.1, the query suggestions that our envisaged interface would present once the user has issued an initial query actually come in two flavors, query completions and query refinements. It is an open question whether a unified method can produce suggestions in both flavors, or rather specialized models are required. Also, the probabilistic generative model that combines keyphrase-based suggestions extracted from multiple information sources, discussed in previous sections, relies heavily on query suggestion services from major web search engines.

Then, another main challenge in our work is to solve this task without relying on suggestions provided by a major web search engine (and possibly even without using a query log).

Adopting a general two-step pipeline approach, consisting of suggestion generation and suggestion ranking steps, in previous sections we have addressed query suggestion as an end-to-end task and studied a probabilistic generative framework. Now, we focus exclusively on the first component. Our aim is to generate sufficiently many high-quality query suggestion candidates. The subsequent ranking step will then produce the final ordering of suggestions by reranking these candidates (and ensuring their diversity with respect to the possible subtasks).

The first research question we address in this section is: (RQ5d) *Can existing query suggestion methods generate high-quality query suggestions for task-based search?* Below, we present three methods from the literature. The first two methods are specialized only in producing query completions, while the third one is able to handle both query completions and refinements. Specifically, we employ the popular suffix [138], neural language model [147], and sequence-to-sequence [175] approaches to generate candidate suggestions.

As before, for a given suggestion q, $P(q|q_0)$ denotes the probability of that suggestion.

6.5.1 Popular Suffix Model

The *popular suffix model* [138] generates suggestions using frequently observed suffixes mined from a query log. The method generates a suggestion q by extending the input query q_0 with a popular suffix s, i.e., $q = q_0 \oplus s$, where $\oplus$ denotes the concatenation operator. The query likelihood is based on the popularity of suffix s: $P(q|q_0) = pop(s)$, where $pop(s)$ denotes the relative frequency of s occurring in the query log.

6.5.2 Neural Language Model

Neural language models (NLMs) can be used for generating query suggestions [147]. Suggestion q is created by extending the input query q_0 character by character: $q = q_0 \oplus \mathbf{s} = (c_1, \ldots, c_n, c_{n+1}, \ldots, c_m)$, where $c_1, \ldots, c_n$ are the characters of q_0, and $\mathbf{s} = (c_{n+1}, \ldots, c_m)$ are characters generated by the neural model. Given a sequence of previous characters $\mathbf{c} = (c_1, \ldots, c_i)$, the model generates the next character ($i \geq n$) according to: $P(c_{i+1}|\mathbf{c}) = softmax(h_i)$, where the hidden state vector at time i is computed using $h_i = f(x_i, h_{i-1})$. Here, f denotes a special unit, e.g., a long short-term memory (LSTM) [84]; x_i is the vector representation of the ith character of suggestion

q and is taken to be $x_i = \sigma(c_i)$, where σ denotes a mapping function from a character to its vector representation. Finally, the query likelihood is estimated according to:

$$P(q|q_0) = \prod_{j=n}^{m-1} P(c_{j+1}|c_1, \ldots, c_j) \ .$$

Our implementation uses a network of 512 hidden units. We initialize the word-embedded vector with the pre-trained vector from Bing queries.[2] Beam search width is set to 30.

6.5.3 Sequence to Sequence Model

The sequence-to-sequence model (Seq2Seq) [119] aims to directly model the conditional probability $P(w'_1, \ldots, w'_m|w_1, \ldots, w_n)$ of translating a source sequence $(w_1, \ldots, w_n)$ to a target sequence $(w'_1, \ldots, w'_m)$. Thus, it lends itself naturally to implement our query suggestion task using Seq2Seq, by letting the initial query be the source sequence $q_0 = (w_1, \ldots, w_n)$ and the suggestion be the target sequence $q = (w'_1, \ldots, w'_m)$. Typically, a Seq2Seq model consists of two main components: an encoder and a decoder.

The *encoder* is a recurrent neural network (RNN) to compute a context vector representation c for the original query q_0. The hidden state vector of the encoder RNN at time $i \in [1..n]$ is given by: $h_i = f(w_i, h_{i-1})$, where w_i is the ith word of the input query q_0, and f is a special unit (LSTM). The context vector representation is updated by $c = \phi(h_1, h_2, \ldots, h_n)$, where ϕ is an operation choosing the last state h_n.

The *decoder* is another RNN to decompress the context vector c and output the suggestion, $q = (w'_1, \ldots, w'_m)$, through a conditional language model. The hidden state vector of the decoder RNN at time $i \in [1..m]$ is given by $h'_i = f'(h'_{i-1}, w'_{i-1}, c)$, where w'_i is the ith word of the suggestion q, and f' is a special unit (LSTM). The language model is given by: $P(w'_i|w'_1, \ldots, w'_{i-1}, q_0) = g(h'_i, w'_{i-1}, c)$, where g is a softmax classifier. Finally, the Seq2Seq model estimates the suggestion likelihood according to:

$$P(q|q_0) = \prod_{j=1}^{m-1} P(w'_{j+1}|w'_1, \ldots, w'_j, q_0) \ .$$

We use a bidirectional GRU unit with size 100 for encoder RNNs, and a GRU unit with size 200 for decoder RNNs. We employ an Adam optimizer with an initial learning rate of 10^{-4} and a dropout rate of 0.5. Beam search width is set to 100.

[2] https://www.microsoft.com/en-us/download/details.aspx?id=52597

6.5.4 Data Sources

The second research question we ask in this section is: (RQ5e) *What are useful information sources for each method?* We consider three independent information sources. We are particularly interested in finding out how a task-oriented knowledge base (Know-How [146]) and a community question answering site (WikiAnswers [52]) fare against using a query log (AOL [148]).

For the PopSuffix and NLM methods, we need a collection of short texts, $\mathcal{C}$. For Seq2Seq, we need pairs of question-suggestion pairs, $\langle \mathcal{Q}, \mathcal{S} \rangle$.

- *AOL query log* [148]: a large query log that includes queries along with anonymous user identity and timestamp. We extract all queries from the log as $\mathcal{C}$. We detect sessions (each session including multiple queries) using the same criterion as in the work by Sordoni et al. [175]. Then, we pair queries in the same session to obtain $\langle \mathcal{Q}, \mathcal{S} \rangle$, where $\mathcal{Q}$ denotes a set of queries and $\mathcal{S}$ are suggestions paired against $\mathcal{Q}$. In order to obtain more pairs, we extract all proper prefixes from the query, and pair them together. For example, given a query "make a pancake", we can construct two pairs ⟨ "make", "make a pancake" ⟩ and ⟨ "make a", "make a pancake" ⟩. This way, we end up with a total of around 112k prefix-query pairs.

- *KnowHow* [146]: a knowledge base that consists of two and half million entries. Each triple $\langle s, p, o \rangle$ represents a fact about a human task, where the subject s denotes a task (e.g., "make a pancake") and the object o is a subtask (e.g., "prepare the mix"). We collect all subjects and objects as $\mathcal{C}$, and take all (around 142k) subject-object pairs to form $\langle \mathcal{Q}, \mathcal{S} \rangle$. Additionally, prefixes from tasks (i.e., subjects and objects) are extracted to get more pairs, the same way as it is done for the AOL query log.

- *WikiAnswers* [52]: a collection of questions scraped from WikiAnswers.com.[3] We detect task-related questions using a simple heuristic, namely, that a task-related question often starts with question constructions "how do you" or "how to." These question constructions are removed from the questions to obtain $\mathcal{C}$ (e.g, "how to change gmail password" → "change gmail password"). This source can only be used for the PopSuffix and NLM methods, as it does not provide pairs for Seq2Seq.

[3] http://www.answers.com/Q/

6.6 Experiments and Results

We design and conduct an experiment to answers our research questions. First, we obtain a pool of candidate suggestions, using the top-N pooling technique [191], by applying the methods on different sources. We then collect annotations for each of these suggestions via crowdsourcing. Finally, we report and analyze the experimental results.

6.6.1 Pool Construction

All queries (100 in total) from the TREC 2015 and 2016 Tasks tracks [200, 188] are considered during the pool construction. We suggest candidates with each combination of the proposed methods (Section 6.5) with the various information sources (Section 6.5.4). With *s-m* we denote a particular configuration that uses method m with source s. In addition, we also include the suggestions generated by (i) the keyphrase-based query suggestion system [63], and (ii) the Google Query Suggestion Service (referred to as *Google API* for short). Two pools are constructed, one for query completions (QC) and one for query refinements (QR). The pool depth is 20, that is, we consider (up to) the top-20 ranked suggestions for each configuration.

6.6.2 Crowdsourcing

Due to the fact that many of our candidate suggestions lack assessments in the TREC ground truth (and thus are considered irrelevant), we obtain relevance assessments for all suggestions via crowdsourcing. Specifically, a total of 12,790 QC and 9,608 QR suggestions were annotated as follows. We used the Crowdflower (now: Figure Eight) platform, where a dynamic number of annotators (3 judges at least; 5 at most, if necessary to reach a majority agreement) were asked to label each suggestion as relevant or non-relevant. A suggestion is relevant if it targets for some more specific aspect of the original query, i.e., the suggestion must be related to the original query intent. It does not have to be perfectly correct grammatically, as long as the intent behind is clearly understandable. The final label is taken to be the majority vote among the assessors. Each annotation batch costed ¢5 and comprised 12 annotation instances. A minimum accuracy of 80%, a minimum time of 25 seconds per batch, and a minimum confidence threshold of 0.7, were required to ensure quality. A Fleiss' Kappa coefficient of $\kappa = 0.6071$ indicates that the inter-annotator agreement for this assessment is moderate [55].

Method	QC				QR	
	P@10	P@20	R	CR	P@10	P@20
AOL-PopSuffix	0.257	0.245	0.168	0.168	-	-
KnowHow-PopSuffix	0.195	0.170	0.102	0.256	-	-
WikiAnswers-PopSuffix	0.181	0.167	0.101	0.333	-	-
AOL-NLM	0.256	0.241	0.170	0.474	-	-
KnowHow-NLM	0.166	0.147	0.108	0.575	-	-
WikiAnswers-NLM	0.163	0.121	0.088	0.650	-	-
AOL-Seq2Seq	0.283	0.181	0.156	0.765	0.043	0.031
KnowHow-Seq2Seq	0.158	0.111	0.079	0.813	0.206	0.148
Keyphrase-based [63]	0.321	0.239	0.130	-	0.575	0.504
Google API	0.267	0.134	0.078	-	0.289	0.145

Table 6.4 Precision for candidate suggestions generated by different configurations. For QC methods, we also report on recall (R) and cumulative recall (CR).

6.6.3 Results and Analysis

Table 6.4 presents a comparison of methods in terms of precision at cut-off points 10 and 20 (P@10 and P@20). Overall, we find that our methods can generate high-quality query suggestions for task-based search. Our best numbers are comparable to that of the Google API. It should, however, be noted that the Google API can only generate a limited number of suggestions for each query. The keyphrase-based method [63] is the highest performing of all; it is expected, as it combines multiple information sources, including suggestions from Google.

We find that the AOL query log is the best source for generating QC suggestions, while KnowHow works well for QR. AOL-Seq2Seq performs poorly on QR; this is because only 2% of all training instances in $\langle \mathcal{Q}, \mathcal{S} \rangle$ are QR pairs. For QC suggestions, the performance of AOL-PopSuffix and AOL-NLM are close to that of the Google API, while AOL-Seq2Seq even outperforms it. For QR suggestions, the performance of AOL-Seq2Seq is close to that of the Google API in terms of P@20, but is lower on P@10. In general, our system is able to produce more suggestions than what the (public) Google API provides.

Additionally, we also evaluate the recall of each QC method, using all relevant suggestions found as our recall base. We further report on cumulative recall, i.e., for line i of Table 6.4 it is the recall of methods from lines 1 to i combined together. We observe that each configuration brings a considerable improvement to cumulative recall. This shows that they generate unique query suggestions. For example, given the

query "choose bathroom," our system generates unique query suggestions from different methods and sources, e.g., "choose bathroom marks" (WikiAnswers-NLM), "choose bathroom supply" (AOL-NLM), "choose bathroom for your children" (KnowHow-NLM), "choose bathroom grout" and "choose bathroom appliances" (KnowHow-Seq2Seq), which are beyond what the Google API and the keyphrase-based system [63] provide. Therefore it is beneficial to combine suggestions from multiple configurations for further reranking.

6.6.4 Summary of Findings

We answer RQ5d by observing that existing query suggestion methods are able to generate high-quality query suggestions for task-based search. We find that the sequence-to-sequence approach performs best among the tested three methods. Although its performance is lower than compared methods for some QC measurements, overall this approach can generate suggestions in both query completion and query refinement flavors. Our best numbers are similar to that of the Google API, but with this API the number of generated suggestions per query is limited compared to ours.

Regarding the data sources for answering RQ5e, we observe that, as expected, the query log is the one leading to the highest performance. Yet this information source does not perform well on QR, due to the small proportion of QR pairs among training instances. Nevertheless, the other two also provide valuable suggestion candidates that cannot be generated from the query log.

Overall, we find that different method-source configurations contribute unique suggestions, and thus it is beneficial to combine them.

6.7 Summary

In this chapter, we have addressed the task of generating query suggestions that can assist users in completing their tasks. We have performed our study adopting a pipeline architecture widely used in general query suggestion systems, consisting of suggestion generation and suggestion ranking steps.

We have first addressed query suggestion as an end-to-end task. Specifically, we have proposed a probabilistic generative framework with four components: source importance, document importance, keyphrase relevance, and query suggestion generation. We have proposed and experimentally compared various alternatives for these components, evaluating with respect to the datasets from the TREC Tasks track. We have found,

in particular, a large contribution of query suggestions provided by major web search engines. Our observations have been largely consistent across the 2015 and 2016 test collections.

In the last part of this chapter, we have focused on the first component of a two-step pipeline, dedicated to creating suggestion candidates. We were interested in studying additional challenges, regarding whether a unified method can produce both query completions and query refinements, and also doing so without relying on resources such as a major search engine suggestions or query logs). For this, we have considered several methods (popular suffix model, neural language model, and sequence-to-sequence model), as well as multiple information sources (a query log, a task-oriented knowledge base, and a collection of questions). We have based our evaluation on the TREC Tasks track, and collected a large number of annotations via crowdsourcing. Our results have shown that we are able to generate high-quality suggestion candidates. We have further observed that the different methods and information sources lead to distinct candidates.

We have approached supporting task-based search by providing query suggestions. The next chapter will continue with our study on strategies aiming to support completing a particular subset of search tasks. We will introduce the problem of task recommendation, this is, recommending specific tasks from a task repository to users, based on their search queries.

Chapter 7

Recommending Tasks based on Search Queries and Missions

People often issue queries with an underlying goal that is concerned with the completion of some task, such as planning a holidays trip or organizing a birthday party. The support for more *complex* search tasks, including exploratory, multi-search, multi-session and multi-step tasks, has been identified as an area where search engines are ought to provide better assistance to their users [99, 82]. The ultimate challenge is to build useful systems "to achieve work task completion" [180], to assist the users in "outlining the necessary steps to accomplish a complex task" [82]. Web search is an experience that naturally lends itself to recommendations, including query suggestions [25, 177, 44, 144] and related entities [20, 19, 53]. In this chapter, we propose to recommend specific tasks to users, based on their search queries. An illustrative scenario is depicted in Fig. 7.1, where an envisaged interface answers the user query "birthday party" by providing a list of recommended tasks.

We introduce the problem of query-based task recommendation, which is cast as a top-N recommendation problem: returning a list of recommended tasks based on a given search query. One peculiarity of this problem that hinders the applicability of collaborative filtering and representation learning approaches is the lack of training data. While a small number of query-task pairs may be obtained as training instances by manual labeling (which is indeed what we also do in this study), this is inherently limited in scale. Given the practically unconstrained space of possible tasks and queries, for any given query, this essentially remains a cold start problem. Therefore, we approach task recommendation as a content-based recommendation problem, which boils down to the challenge of scoring tasks in response to input queries. For that reason, we start with well-established term-based ranking approaches, and combine

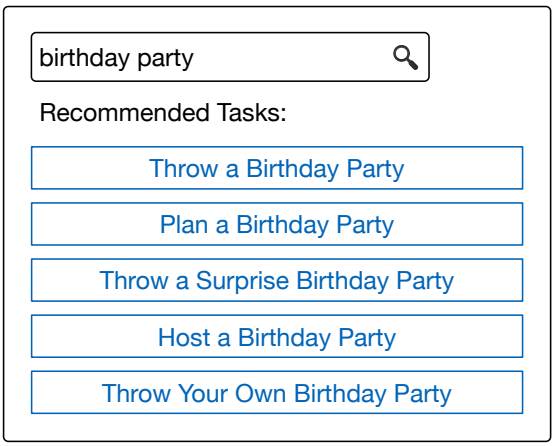


Fig. 7.1 Query-based task recommendations.

them with semantic matching signals developed for the task at hand. Beyond a single input query, we are also interested in the scenario of a set of queries that all share the same underlying task, referred to as search mission. Hence, we introduce the problem of mission-based task recommendation. In this chapter, we aim to answer the following research question: (RQ6) *How can we recommend tasks based on search queries and missions?*

One of the main contributions of this study addresses the lack of a standard test collection to evaluate our novel problem. Building on a dataset of detected search missions developed by Hagen et al. [74], we create the first test collection to assess task recommendation. Another central contribution is establishing baseline methods for the query-based and mission-based task recommendation problems. We propose a query-based task recommendation approach, by combining well-established term-based matching methods with semantic matching signals based on continuous representations. Further, we extend our approach to deal with an input search mission. Task recommendation results in a search component that can be applied on top of any method proposed for detecting queries that belong to the same underlying task [92, 74, 117, 118, 116].

Next in this chapter is a revision of related literature (Section 7.1); the rest is organized as follows. We introduce two research problems, query-based and mission-based task recommendation, in Section 7.2. Section 7.3 describes the approaches that we propose to address these problems. We build a test collection, consisting of a task repository, a set of search missions, representative "best" queries for each mission, and corresponding relevance assessments (Section 7.4). Then, in Section 7.5 we report the experimental results, as well as provide a detailed feature and query analysis.

7.1 Related Work

In this section, we firstly discuss a body of related work on search sessions and missions. Then, we contrast our work with research literature that studies procedural knowledge.

7.1.1 Search Queries, Sessions, and Missions

The seminal high-level categorization of search behavior by Broder [28], classifying the queries in navigational, informational or transactional, has been subsequently extended as more online search activities are identified [165, 89, 122]. Of particular interest for our work is multi-step search task [81], which encompasses the behavior where a complex task, involving multiple particularities, is aimed to be fulfilled.

A large body of work targets the detection of search sessions in query logs, with the ultimate aim of identifying search patterns in user behavior and of providing additional support upon them. In the field of query log analysis, a *search session* is usually defined as a sequence of queries where the elapsed time between two consecutive queries is not larger than a delta value, which ranges from 30 minutes [83, 154] to 90 minutes [74]. Beyond session detection, focus was also placed on the challenge of understanding the task-based user interaction with search engines. Here, "task" refers to the overall goal ("work task") of the user that motivated her to search in the first place. A notion of *search mission* is established by Jones and Klinkner [92]. They introduce the concept of hierarchical search, instituting the idea of a search mission as a set of topically-related queries, not necessarily consecutive, composed by simpler search goals. Their work also discusses a method for detecting whether a given query pair belongs to the same goal. We adhere to their concept of a search mission, to emphasize the coherent nature of search queries motivated by a single underlying goal, as opposed to other lines of work that focus on multi-tasking behavior across sessions. Several works have extended the problem of detecting search missions, and proposed approaches accordingly [50, 117, 74, 157, 193, 108, 189, 190, 109, 116, 125]. Rather than the usual flat representation of tasks, methods to obtain a hierarchy of tasks and their corresponding subtasks have been studied [128, 126, 129].

We assume that we have access to a mission detection oracle which identifies a sequence of queries all corresponding to the same (unknown) task. The assumption of such a mission oracle in place avoids to be hindered by the errors of the mission detection method, on top of which our task recommendation approach can be performed. Moreover, and for the sake of simplicity, we relax the definition of mission, by discarding

the notion of sequence and only considering search mission as an unordered set of queries all with the same underlying task.

Recommendations based on search queries have been studied for specific domains, such as movies or celebrities [19, 203]. Such model-based methods typically rely on manually designed domain-dependent features, related to specific properties of entities, and usually deal with a query that consists of one of the names of an entity. Instead, we address search queries that express the need for solving an underlying task. We recommend elements from a repository of task descriptions, using different attributes in the document representation of a task, such as title or explanation.

7.1.2 Procedural Knowledge

Task recommendations do not benefit necessarily from descriptive or declarative KBs. Indeed, these KBs lack sufficient coverage of *procedural* knowledge. Procedural knowledge [70] is the knowledge involved in accomplishing a task. A few *procedural knowledge bases* are specialized in semi-structured knowledge [145, 37], that has been extracted from corpora of how-to guides collaboratively created in projects like WikiHow or eHow.[1] Jung et al. [94] automatically create a situation ontology from how-to instructions through a model of daily situational knowledge with goals, action sequences, and contextual items. They also envisage a schema mapping actions to actual service providers. As mentioned above, for pragmatic reasons we choose to work with a collection of procedural articles from WikiHow as our repository to recommend tasks from.

Pareti et al. [145] identify properties that define community-centric tasks. They then propose a framework to represent the semantic structure of procedural knowledge in web communities. The proposed representation operates distributed across different KBs, and comprises a process ontology to express tasks, sub-tasks, requirements, and execution status.

Chu et al. [37] propose to build a procedural KB by normalizing the how-to tasks from WikiHow into a hierarchical taxonomy that also represents the temporal order of sub-tasks, as well as the attributes of involved items. Their method relies on hierarchical agglomerative clustering of task frames obtained by open-domain semantic role labeling. They show the usefulness of this KB in expanding individual, task-oriented queries for retrieving relevant YouTube videos. Although we address a different problem, namely

[1] http://www.ehow.com

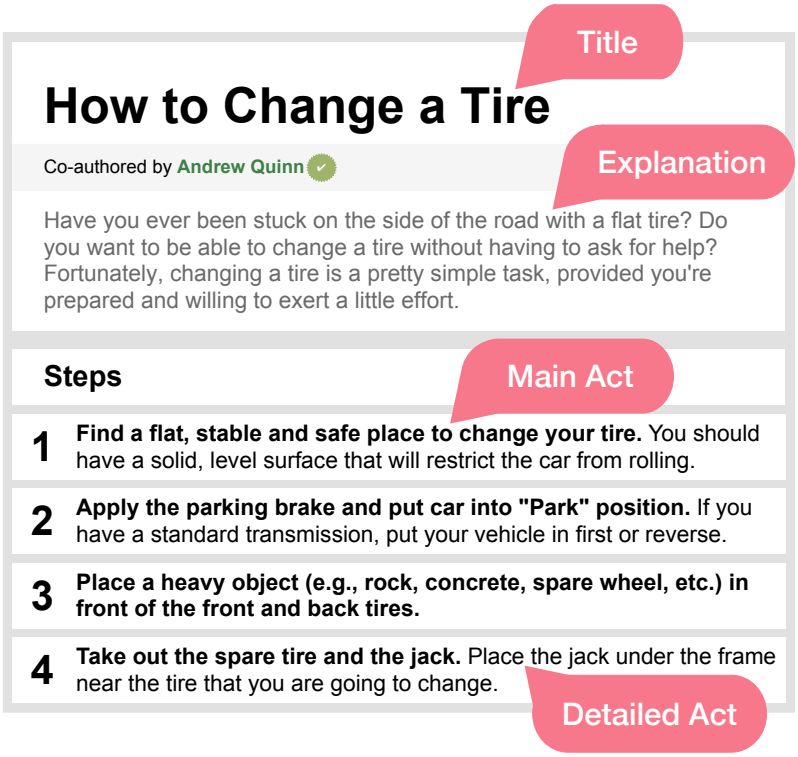


Fig. 7.2 WikiHow article corresponding to the task "How to change a tire." The task has a *title* and a *brief explanation*. Each step has a *main act* corresponding to a subtask (here, the first sentence of each step), and is possibly complemented with a *detailed act*.

explicit task recommendation rather than leveraging tasks for recommending other kind of items, we use their dataset of WikiHow articles as our task repository.

7.2 Problem Statement

In this section, we formally introduce two specific task recommendation problems.

7.2.1 Query-based Task Recommendation

The first problem is concerned with the recommendation of tasks in response to a web search query. A key step in addressing this problem has to do with the operationalization of the abstract notion of *task*. We need to establish the universe of possible tasks as well as a computational representation of tasks. For that, we introduce the concept of a *task repository*.

Definition 2. *A task repository $\mathcal{T}$ is a catalog of task descriptions. A* task description *$t \in \mathcal{T}$ is a semi-structured document that explains the steps involved in the completion of a given task.*

A task description can be, for example, a how-to article illustrating how to change a tire (see Fig. 7.2). It typically contains parts such as a title and a brief explanation of the task, then enumerates the steps needed to accomplish the task. We say that the task description is semi-structured to emphasize that some of these document elements may be missing. Each element may be viewed as a *task attribute*, similar to attributes of entities in a knowledge base. Throughout the chapter, when we say *task*, we simply refer to this structured representation of a task description.

Now, we can formally define the problem of *query-based task recommendation*.

Definition 3. *Given a search query q, the problem of* query-based task recommendation *is to return a ranked list of tasks from a task repository $\mathcal{T}$ that correspond to the task underlying q.*

An illustration of this problem is given by Fig. 7.1, where an example query "birthday party" leads to the recommendation of five tasks.

7.2.2 Mission-based Task Recommendation

We introduce now the *mission-based task recommendation* problem. It generalizes the query-based task recommendation to recommend tasks for a set of queries, all of which are motivated by the same (unknown) underlying task. We refer to such query sets as *search missions*. Specifically, a search mission, or simply *mission*, is a set of queries $m = \{q_1, \ldots, q_n\}$, possibly spanning multiple search sessions, that all share the same underlying task.

We do not address the problem of search mission detection, and instead assume that there is a process in place that identifies and groups queries that share the same underlying task [92, 74, 118].

Definition 4. *Given a search mission m as input, the problem of* mission-based task recommendation *is to return a ranked list of recommended tasks, corresponding to the queries in m.*

Similar to the previous example, given a single query "cake recipe," a highly relevant task to be recommended is t_1 = "Make a cake." Now, assuming that the search mission also contains a second query "cake decorating ideas," the task t_2 = "Make a birthday

cake" would also be relevant. Further, if the mission had a third query "birthday invitation cards," then the recommended tasks could also include t_3 = "Organize a birthday party."

7.3 Approach

This section presents the methods we develop for the problems stated in Section 7.2. It is important to point out that for these novel problems, we do not have large volumes of query-task pairs to be exploited as training data. This essentially leaves us with a cold start problem and influences the choice of methods to consider. We approach task recommendation as a content-based recommendation problem, which boils down to the challenge of scoring tasks in the repository in response to input queries or missions. The user is then presented with the top-N highest scoring tasks as recommendations.

7.3.1 Query-based Task Recommendation

Our approach for query-based task recommendation is based on document ranking techniques. By considering individual task attributes as documents, we can apply traditional term frequency-based techniques to rank tasks. Further, we propose to take semantic matching signals into account as well, to extend the capabilities of these already strong traditional techniques.

Term-based Matching

We consider every task $t \in \mathcal{T}$ as a document and build an inverted index for each task attribute t_a (Title, Explanation, MainAct, and DetailedAct) for efficient computation. We then score each task attribute t_a against the input query q using the well-established BM25 retrieval method [163], obtaining $\text{score}_{BM25}(t_a, q)$.

Semantic Matching

Consider the query "how to loan shark" (taken from our test corpus, cf. Section 7.4.1). When matching potential task descriptions, a traditional retrieval method based on term frequencies would fail to capture that the term "shark" does not refer to the fish but to the financial practice of predatory lending with extremely high interest rates. Another example is the query "how do i put photos in my ipod." A standard document-based scoring method would not be able to capture that "pictures" is a synonym of "photos" in the task title "How to add pictures to my ipod." This kind of

Group	Word embedding	Word functions	#
BM25	-	-	4
W2V, selected	word2vec	SELECTED	28
W2V, complement	word2vec	COMPLEMENT	28
DESM, selected	DESM IN-OUT	SELECTED	16
DESM, complement	DESM IN-OUT	COMPLEMENT	16
Total number of features			92

Table 7.1 Features used for learning to recommend tasks.

vocabulary mismatch is the motivation behind our choice for additionally leveraging semantic representations of terms provided by continuous word embeddings.

We use the cosine similarity between a query q and a given task attribute t_a as the semantic score for q and t_a:

$$\text{score}_{sem}(t_a, q) = cos(\boldsymbol{t}_a, \boldsymbol{q}) \ . \tag{7.1}$$

The query and task attribute are each represented by a vector, $\boldsymbol{q}$ and $\boldsymbol{t}_a$, respectively, which are taken to be the centroids of the word vectors corresponding to the individual terms making up q and t_a. The set of words may be further restricted to a word function, i.e., filtered according to the part-of-speech (POS) tag of a term. Formally:

$$\boldsymbol{q} = \frac{1}{|\{w \in q \wedge F(w)\}|} \sum_{w \in q \wedge F(w)} \boldsymbol{w} \ , \tag{7.2}$$

where a term vector $\boldsymbol{w}$ is given by some word embedding, and a query term w is only considered if it satisfies a given word function F. The task attribute vector $\boldsymbol{t}_a$ is calculated analogously.

We propose to combine these semantic matching signals together with the term-based matching score in a learning-to-recommend (LTR) approach. LTR has been widely used for various recommendation problems in the past [174, 173, 171, 172, 155]. The novelty of our work lies in developing features that are appropriate for the task at hand. Specifically, we instantiate the semantic matching score in Eq. (7.1) for each task attribute using multiple pre-trained word embeddings and combinations of word functions.

Features

Next, we detail the set of features we consider; these are summarized in Table 7.1. The first group of features are based on term-based matching using the BM25 method (cf. Section 7.3.1), computed for each of the four task attributes. The next 4 groups of features (lines 2–5 in Table 7.1) are based on semantic matching, using all 4-way combinations of word embeddings (2) and sets of word functions (2).

Embeddings We consider two pre-trained word embeddings.

- **W2V**: We employ the widely used embeddings provided by the *word2vec*[2] toolkit [135]. These vectors are trained on part of the Google News dataset (about 100 billion words) using a CBOW model with negative sampling. The embedding contains 3 million 300-dimensional vectors. These embeddings correspond to the input matrix in the CBOW model [135], i.e., the matrix between the input and hidden layers.
- **DESM**: The Dual Embedding Space Model [139] provides two embedding spaces, IN and OUT, each with 200-dimensional vectors for around 2,74 million terms. This model is the same as in word2vec but (i) is trained on a large unlabelled query corpus, and (ii) retains both projections IN and OUT, corresponding to the input and output matrices in CBOW model, respectively. By using IN embedding for the query terms and OUT embedding for the task terms, the semantic similarity is higher between words that often co-occur in the same query or task (topical similarity) [139], i.e., overcoming part of the vocabulary mismatch problem. We distinguish between these IN and OUT embeddings when taking vectors for words, but refer to this setting altogether as vectors from the DESM model.

Word functions We propose to leverage signal based on part-of-speech tags of words. Consider, for example, the query "put a photo in my ipod." Intuitively, the verb "put" should be a strong indicator of the underlying task, with respect to other similar tasks like "take a photo with my ipod." Similarly, the noun "photo" should distinguish the corresponding task from others like "put a video in my ipod." We consider the following word POS-tags: verbs (V), nouns (N), and adjectives (A). Each of these three POS-tags is taken as a word function, e.g., we say that a term satisfies the word function V if the term is a verb. With these building blocks, we define combinations, such as $V + N$, meaning that a word is a verb or a noun, and consider these combinations as word functions as well. We use ALL to refer to any word

[2]https://code.google.com/archive/p/word2vec/

function. The possible combinations of word functions are divided into two disjunct sets:

- **SELECTED** = $\{V, N, A, V+N, V+A, N+A, V+N+A\}$.
- **COMPLEMENT** = $\{ALL-V, ALL-N, ALL-A, ALL\}$.

In total, our feature set includes 92 features. As we show in Section 7.5.2, it is possible to get a competitive level of performance with just 2 features, while maximal effectiveness is reached when using 27 features.

7.3.2 Mission-based Task Recommendation

We turn now to the problem of mission-based task recommendation. Given a search mission $m = \{q_1, ..., q_n\}$, let R_i denote the ranked list of recommended tasks for q_i, obtained by a query-based task recommender method (cf. Section 7.3.1). Further, we let T be the set of all tasks in all rankings, $T = \bigcup_{i=1}^{n} \{t : t \in R_i\}$.

We introduce a general approach for mission-aware task recommendation that aggregates the individual query-based rankings for each query $q_i \in m$ into a mission-level task ranking. Specifically, let s be a scoring function, such that $s(q_i, t)$ is the score corresponding to task t in the ranking R_i. Then, for each task $t \in T$, the mission-based recommendation score of t is obtained by aggregating the query-based recommendation scores with a given aggregator function $aggr$:

$$\text{score}_{aggr}(m, t) = aggr_{q_i \in m}\{s(q_i, t)\} \ . \tag{7.3}$$

We propose two ways to estimate $s(q_i, t)$, each leading to a particular family of mission-based task recommendation methods.

- **Score-based method**: we take $s(q_i, t)$ to be the raw score of the query-based task recommender, score(q_i, t) (with $s(q_i, t)$ equal to 0 if $t \notin R_i$).
- **Position-based method**: we define $s(q_i, t)$ to be the reciprocal rank position of t in the ranking for q_i. I.e., $s(q_i, t) = 1/r_i(t)$, where $r_i(t)$ is the rank position of recommended task t in the ranking R_i (with indexing starting from 1 for the top task, and setting $r_i(t) = |R_i| + 1$ if $t \notin R_i$, where $|R_i|$ is the length of the ranked list R_i).

The aggregator function is one of $\{sum, max, avg\}$. Accordingly:

- if $aggr = sum$, we have $\text{score}_{sum}(m, t) = \sum_{q_i \in m} s(q_i, t)$;

- $aggr = max$ leads to $\text{score}_{max}(m,t) = \max_{q_i \in m}\{s(q_i,t)\}$;
- and when $aggr = avg$, $\text{score}_{avg}(m,t) = \frac{1}{n}\sum_{q_i \in m} s(q_i,t)$.

7.4 Dataset Construction

In Section 7.2, we have introduced two task recommendation problems. Given their novelty, there is no standard test collection available to evaluate methods addressing these problems. As one of the main contributions of this study, we create the first test collection for task recommendation. A key challenge here is that of establishing a task repository. Finding one that would cover any arbitrary task appears to be very difficult to accomplish. For this reason, we focus our efforts on a particular subset of tasks, referred to as *procedural tasks*, which are of general interest and are also well covered in a public resource, namely WikiHow.[3] A procedural task is a search task that can be accomplished by following a sequence of specific actions or subtasks. Correspondingly, a *procedural mission* is a search mission whose underlying task is procedural.

To illustrate the concept of a procedural task, consider two user goals: (1) reading reviews about a movie and (2) finding out how to make a birthday cake. Of these two, only the latter is a procedural task, since it can naturally be explained as a sequence of steps.

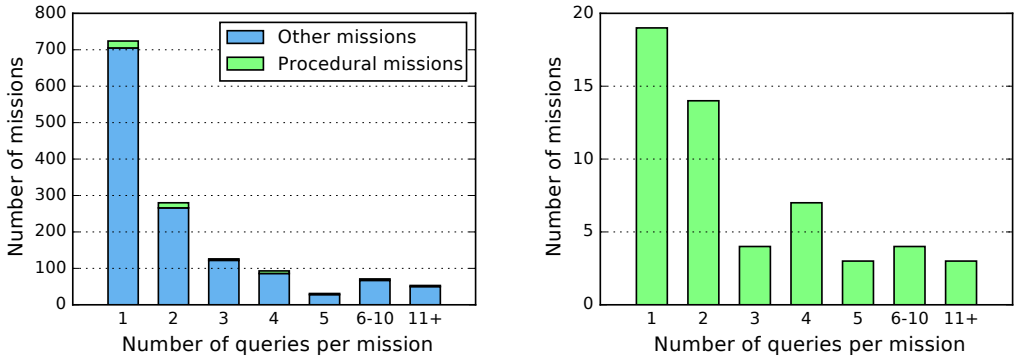
In this section, we describe our corpus of search queries and missions (Section 7.4.1), the identification of procedural missions (Section 7.4.2), and the usage of WikiHow as our task repository $\mathcal{T}$ (Section 7.4.3).

7.4.1 Corpus of Search Queries and Missions

We base our experiments on the Webis Search Mission Corpus 2012 (Webis-SMC-12)[4] developed by Hagen et al. [74]. It contains 8,840 queries issued by 127 users, corresponding to a three-months sample from the 2006 AOL query log. These queries have been manually assigned to a total of 1,378 search missions by two human assessors. By using this dataset, we eliminate potential errors associated with mission detection and thus ensure that our evaluation is not affected by noisy input from an upstream processing step.

In the original dataset, consecutive queries that belong to different missions are delimited by a special "logical break" marker. Since we are not interested in the

[3] http://www.wikihow.com
[4] https://webis.de/data/webis-smc-12.html



(a) All missions (b) Procedural missions

Fig. 7.3 Distribution of (a) missions and (b) procedural missions according to the number of queries they contain.

multi-tasking aspect of search behavior (i.e., switching between missions in a same session), it is only the set of all queries within a given mission that matters to us. Hence, we post-process the original corpus to merge all queries that belong to the same mission into a single set, by ignoring the logical breaks. We further remove duplicate queries when they correspond to consecutive queries with the same timestamp (effectively, we ignore multiple consecutive clicks for the same query). This results in a corpus of 1,378 missions, as originally, but now with a total of 3,873 queries.

7.4.2 Corpus of Procedural Search Missions

In the next step, three human experts assessed each of the search missions to decide whether a mission is procedural or not. In the case of a mission being procedural, each assessor was also requested to choose a single query from the mission that is the most representative for the task. The Fleiss' Kappa inter-annotator agreement for this assessment was 0.55, which is considered moderate [55]. For each mission, we take the majority vote to be the final label. Further, we take the union of all queries that were chosen as most representative by at least one assessor, and refer to this set as the *best queries* of the mission. As a result, 54 out of the 1,378 missions were labeled as procedural. This amounts to approximately 4% of all missions in our sample, which is a considerable proportion. Among the 54 procedural missions, the vast majority has each a single best query (6 missions have 2 best queries and 2 missions have 3 best queries).

Figure 7.3 shows the distribution of missions in our corpus according to the number of queries they contain. 51% of the missions consist of a single query, and 19% of the

Mission ID: 707848_8	Mission ID: 1045635_22
how to put music on your ipod	**find adress by phone number**
add songs to ipod	acton adress by phone number
put music on your ipod	public records adress by phone
	free adress lookup by phone
	free phone book phone and adress
	acton california adress by phone

Table 7.2 Examples of queries for two procedural missions in our corpus. Best queries in boldface.

missions contain each exactly 2 queries (Fig. 7.3(a)). As for the procedural missions, 19 out of the 54 consist of a single query (Fig. 7.3(b)). This leaves us with 35 procedural missions with 2+ queries, which is a suitable sample to be used for evaluating our second problem (mission-based task recommendation). Two example procedural missions are shown in Table 7.2.

7.4.3 Tasks Repository

We use the HowToKB dataset,[5] developed by Chu et al. [37], as our tasks repository $\mathcal{T}$ (cf. Section 7.2). This dataset contains 168,697 articles crawled from WikiHow. Figure 7.2 shows an excerpt from a WikiHow article. Each article describes a procedural task and contains the following elements: title, explanation, and sequence of steps (each made of a main act, and possibly also of a detailed act) corresponding to its subtasks. Thus, each task $t \in \mathcal{T}$ is uniquely identified by its WikiHow ID, and represented by a set of four task attributes: Title, Explanation, MainAct, and DetailedAct.

7.4.4 Relevance Assessments

In order to build a test collection for task recommendation, we collect annotations for query-task pairs via crowdsourcing. For each annotation instance (query-task pair), crowd workers were presented with a search query, and a candidate WikiHow article consisting of a title and an explanation. We annotate every best query for all the procedural missions. We asked crowd workers to decide whether a task is highly relevant, relevant, or non-relevant for the query. A document is highly relevant if it is the exact task that the user wants to accomplish when issuing the query. A document is relevant if it is somehow related to the task behind the query, but not

[5] http://people.mpi-inf.mpg.de/~cxchu/projects/howtokb/readme-howtokb.html

Query	Recommended task	Relevance
"opening a small business"	Open a Small Business	Highly relevant
	Brainstorm Small Business Ideas	Relevant
	Compete for Talent As a Small Business	Non-relevant
"how do i put photos in my ipod"	Put Pictures on an iPod	Highly relevant
	Transfer Photos from iPod to PC	Relevant
	Take Screenshots With iPod Touch	Non-relevant

Table 7.3 Examples of queries, recommended tasks, and corresponding relevance assessments in our test collection.

exactly the searched task. Otherwise, it is non-relevant. Table 7.3 lists a number of specific examples from our test collection, illustrating the relevance of tasks for queries. Our annotation process uses the well-established top-N pooling technique [191]; further details are provided below.

Pooling. For each query, we rank tasks using three document ranking baselines: BM25, RM3, and SDM. (cf. Section 7.5.1 for details.) Each method was used to generate rankings based on two task attributes, Title and Explanation, resulting in a total of 6 rankings. We take the top 10 tasks from each of these six rankings to form the pool of tasks that will be annotated. (All tasks outside this pool are assumed to be non-relevant for a given query.)

Collecting judgments. Using the Figure Eight platform,[6] a total of 933 unique instances (query-task pairs) were annotated, each by at least 3 judges (5 at most, if necessary to reach a majority agreement, using dynamic judgments). We paid ¢8 per batch, comprising 5 annotation instances. We ensured quality by requiring a minimum accuracy of 80%, a minimum time of 35 seconds per batch, and a minimum confidence threshold of 0.8. The final label is taken to be the majority vote among the assessors. The Fleiss' Kappa inter-annotator agreement for this assessment was 0.4116, which is considered moderate [55].

In summary, the resulting test collection comprises 59 queries and corresponding relevance assessments for query-based task recommendation (based on the best queries in the missions and removing those for which no relevant task was found in the task catalog). For mission-based task recommendation, we have 54 missions (again, after

[6] https://www.figure-eight.com/

the removal of those without any relevant tasks in the catalog). Relevance labels for missions are obtained by taking the union of the corresponding assessments for the best queries in the mission. In case of conflicting labels for a given task, we take the maximal relevance value assigned to the task.

7.5 Experimental Results

In this section, we describe the experimental setup designed to assess our proposed approaches for the problems of query-based and mission-based task recommendation, and report and analyze the results we obtained.

7.5.1 Experimental Setup

We report on the following rank-based metrics: Normalized Discounted Cumulative Gain (NDCG) and precision (P), each at a cut-off of 10 (indicated as @10), and Mean Average Precision (MAP). These metrics provide high robustness and discriminative power for assessing top-N recommendations [184]. The relevance level ("gain") of a task to calculate NDCG is set to 2 if it is highly relevant, 1 if relevant, and 0 otherwise. For the LTR results, we used 5-fold cross-validation.

We obtain the baseline rankings using Anserini [199]. This software toolkit, built around the open-source Lucene search engine, provides state-of-the-art bag-of-words ranking models [111]. We build an inverted index per task attribute (cf. Section 7.3.1) and consider three baselines: (i) BM25, where parameters are set as usual to be $k_1 = 1.2$ and $b = 0.75$ [163], (ii) SDM (Sequential Dependence Model) [132], and (iii) RM3 variant [87] of relevance models [106]. SDM and RM3 are used with the default parameters in Anserini.

Semantic features, for each attribute, are computed for the top-N results retrieved using the corresponding BM25 methods, with k set to 200. We employ the Random Forest algorithm for regression as our supervised ranking method. Following standard parametrization, we set the number of trees (iterations) to 1000, and the maximum number of features in each tree, m, to (the ceiling of) the square root of the size of the feature set.

7.5.2 Query-based Task Recommendation

We begin our experimental investigation by assessing the extent to which content-based recommendation is able to recommend tasks for search queries, and how much we are

Method	NDCG@10			
	Title	Explanation	MainAct	DetailedAct
BM25	0.3634	0.2926	0.1723	0.1723
RM3	0.3500	0.2736	0.1881	0.1761
SDM	0.3676	0.3019	0.1686	0.1884
LTR	**0.4298**			

Method	P@10			
	Title	Explanation	MainAct	DetailedAct
BM25	0.2237	0.1881	0.1220	0.1339
RM3	0.2305	0.1831	0.1390	0.1508
SDM	0.2237	0.1915	0.1220	0.1458
LTR	**0.2678**			

Method	MAP			
	Title	Explanation	MainAct	DetailedAct
BM25	0.2731	0.2327	0.1420	0.1332
RM3	0.2551	0.2267	0.1476	0.1429
SDM	0.2752	0.2408	0.1364	0.1356
LTR	**0.4150**‡			

Table 7.4 Performance of query-based task recommendation, measured in terms of NDCG@10, P@10, and MAP. Best scores are in boldface. The symbol ‡ denotes statistical significance of the LTR method against BM25-Title, tested using a two-tailed paired t-test with Bonferroni correction [137] at $p < 0.000\widehat{3}$.

able to improve by incorporating distributional semantic features. Hence, we start by identifying a ranking baseline. Our first research question is (RQ6a) *Which one is a solid text-based baseline for task recommendation?*

The top block of Table 7.4 presents the document ranking methods we experiment with. We apply the three ranking models on each of the indexed task attributes. The main observation we make is that the performances of the three models are very close to each other, for all metrics and task attributes. Indeed, using a two-tailed paired t-test with Bonferroni correction [137], we find that no statistical significance exists either between SDM and BM25 or between RM3 and BM25 (for any metric or task attribute). While SDM and RM3 are known to be effective techniques across a broad range of ranking problems [42, 73], for task recommendation they fail to bring any

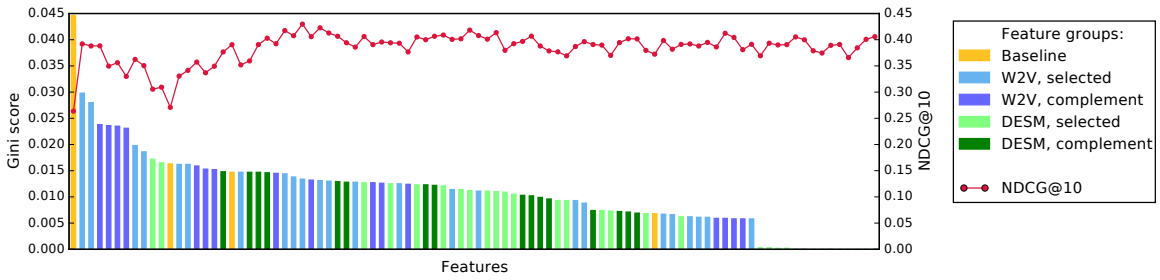


Fig. 7.4 Performance of our LTR approach, in terms of NDCG@10, when incrementally adding features according to their individual information gain, measured by Gini score. The feature grouping is described in Table 7.1.

improvements over BM25. If we focus on contrasting by attributes, we observe that Title is consistently the best performing attribute by a clear margin in all metrics and all models. This is reasonable, since the best queries in our procedural missions are rather well-formed, and WikiHow article titles are syntactically correct phrases, mostly adhering to the pattern "How to ⟨verb⟩ ⟨noun phrase⟩ ⟨complement⟩," which often matches very well with the query. The Title attribute is followed by Explanation, in terms of importance, while the other two attributes do not have a clear order. Based on the above observations, we answer RQ6a by identifying BM25-Title as a solid term-based matching method, and will consider that henceforth as our *baseline* for query-based task recommendation.

Next, we wish to assess the effectiveness of our semantic matching features that are combined in a learn-to-recommend (LTR) approach. This gives rise to the following research question: (RQ6b) *How effective are our semantic matching features and learning-to-recommend approach for query-based task recommendation?*

Evaluation results for our LTR approach are shown in the bottom line of Table 7.4. Clearly, the improvements w.r.t. the baseline are substantial (18% in terms of NDCG@10, 20% in terms of P@10, and 52% in terms of MAP). Moreover, we verify that these improvements are statistically significant for MAP. Hence, the answer to RQ6b is that leveraging semantic matching signals in a learning-to-recommend approach constitutes a significantly more effective method than any of the existing term-based baselines. These results attest that we were able to help substantially reduce the vocabulary mismatch problem between queries and tasks.

Feature Analysis

The semantic features proposed in our approach (cf. Section 7.3.1) encompass several different signals, taken from all task attributes, two word embeddings, and two different

combinations of possible word functions. To understand how each of these features contribute to overall performance, we formulate the following question to guide our analysis: (RQ6c) *What are the most important features when learning to recommend tasks?*

We analyze the discriminative power of our features by sorting them according to their individual information gain, measured in terms of Gini importance; these are shown as the vertical bars in Fig. 7.4. Then, we incrementally add features, one by one, according to their importance and report on performance in terms of NDCG@10; this is shown as the line plot in Fig. 7.4. (Since the number of features change, in each iteration, we set the m parameter of the Random Forests algorithm to the square root of the size of the feature set.)

The performance and importance of the 27 most important features is detailed in Table 7.5. Our findings are as follows. The top feature is BM25-Title; this is not surprising, given its performance. The next eight most important features are all semantic features obtained from the Title attribute and using the W2V embedding. This confirms the importance of Title as the most informative attribute. Also, it shows that, contrarily to what we expected as explained in Section 7.3.1, the *typical* similarities that dominate the way this embedding is usually employed contributes more to the semantic matching than the *topical* similarity in the IN-OUT strategy with DESM. These semantic features use mainly the following word functions, in order: $V + N$ (which shows, as expected, the importance of selected words as verbs and nouns), $V + N + A$ (practically same contributions as $V + N$ since almost no adjective occurs in the queries or task titles), followed by COMPLEMENT word functions.

Looking at how performance changes when adding features one-by-one, we can actually outperform the baseline by adding a single semantic feature (NDCG@10 of 0.3920). After adding seven more features, performance slowly deteriorates, yet still remains above 0.3505. The next two semantic features (again, by $V+N$ and $V+N+A$ word functions, but now from DESM on Explanation), and then BM25-DetailedAct, make notably decrease the recommendation performance. In particular, DetailedAct, the attribute that represents the subtasks for the candidate task, fails to exhibit any contribution on its own when considered only for term-based matching. However, when including additional features that follow in line, which are semantic matching features obtained for DetailedAct, we observe substantial performance gains. We find that performance peaks at an NDCG@10 of 0.4298, which is achieved with 27 features. Adding more features slightly hurts performance, but not significantly so. In answer to RQ6c, we remark the effectiveness of (i) textual similarity, enriched with distributional

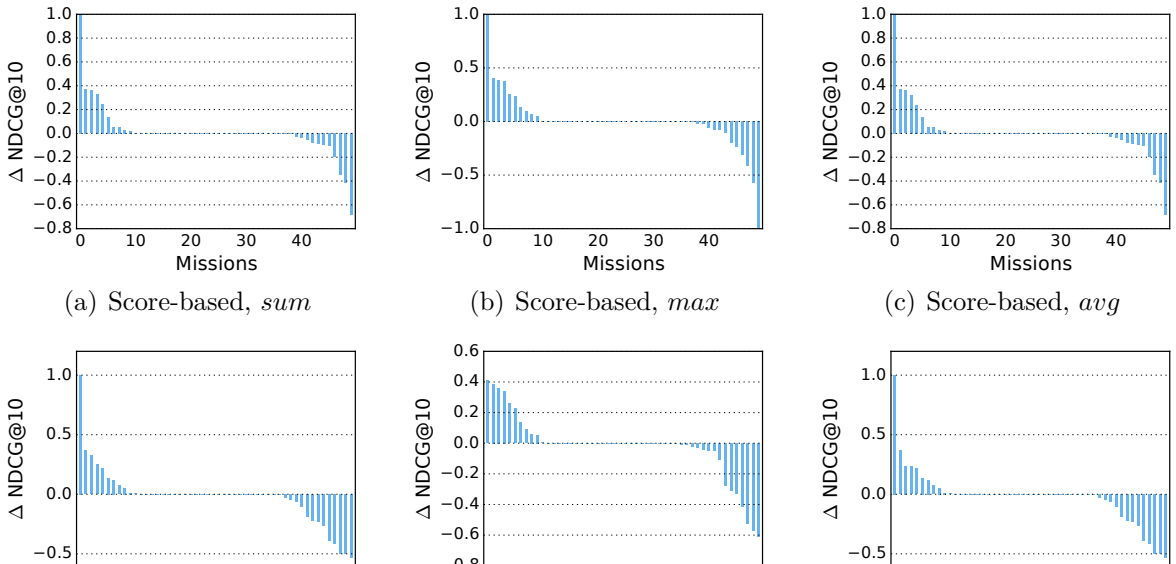
Added feature	NDCG@10	Gain
BM25-Title	0.2637	0.0446
$\oplus$ Title : $V + N$: W2V	0.3920	0.0298
$\oplus$ Title : $V + N + A$: W2V	0.3881	0.0280
$\oplus$ Title : $ALL - N$: W2V	0.3883	0.0238
$\oplus$ Title : $ALL - A$: W2V	0.3496	0.0236
$\oplus$ Title : ALL : W2V	0.3561	0.0235
$\oplus$ Title : $ALL - V$: W2V	0.3300	0.0231
$\oplus$ Title : $N + A$: W2V	0.3625	0.0198
$\oplus$ Title : N : W2V	0.3505	0.0186
$\oplus$ Explanation : $V + N$: DESM	0.3056	0.0172
$\oplus$ Explanation : $V + N + A$: DESM	0.3094	0.0165
$\oplus$ BM25-DetailedAct	0.2709	0.0163
$\oplus$ DetailedAct : $N + A$: W2V	0.3306	0.0162
$\oplus$ DetailedAct : N : W2V	0.3415	0.0162
$\oplus$ DetailedAct : ALL : W2V	0.3573	0.0159
$\oplus$ DetailedAct : $ALL - A$: W2V	0.3369	0.0153
$\oplus$ DetailedAct : $ALL - V$: W2V	0.3494	0.0152
$\oplus$ Title : ALL : DESM	0.3767	0.0148
$\oplus$ Title : $ALL - N$: DESM	0.3906	0.0147
$\oplus$ DetailedAct : $V + N$: W2V	0.3519	0.0147
$\oplus$ Title : $ALL - V$: DESM	0.3595	0.0147
$\oplus$ BM25-Explanation	0.3905	0.0147
$\oplus$ Title : $ALL - A$: DESM	0.4029	0.0146
$\oplus$ DetailedAct : $ALL - N$: W2V	0.3923	0.0145
$\oplus$ DetailedAct : $V + N + A$: W2V	0.4175	0.0144
$\oplus$ Explanation : V : W2V	0.4076	0.0138
$\oplus$ Explanation : $V + N$: W2V	0.4298	0.0134

Table 7.5 Performance of query-based LTR approach, measured by NDCG@10, after incrementally incorporating the first 27 most important features proportional to their individual information gain, measured by Gini score, into the features set. With $t_a : F : E$, we denote a semantic feature using task attribute t_a, word function F and word embedding E.

semantic representations leveraged by typical similarity, (ii) selected word functions, and with (iii) highest contributions from the Title task attribute.

Method	Aggregator	Metric		
		NDCG@10	P@10	MAP
Query-based	-	0.4264	0.2660	**0.4161**
Mission-based by score	Sum	**0.4356**	0.2940	0.4074
	Max	0.4247	**0.2980**	0.3897
	Avg	0.4351	0.2940	0.4055
Mission-based by position	Sum	0.4078	0.2820	0.3744
	Max	0.4056	0.2860	0.3619
	Avg	0.4057	0.2800	0.3729

Table 7.6 Performance of mission-based task recommendation, compared against a query-based method, in terms of NDCG@10, P@10, and MAP. Best scores for each metric are in boldface; improvements over the query-based method are underlined.



(a) Score-based, *sum* (b) Score-based, *max* (c) Score-based, *avg*

(d) Position-based, *sum* (e) Position-based, *max* (f) Position-based, *avg*

Fig. 7.5 Differences in NDCG@10 per query between a given mission-based method and the query-based method.

7.5.3 Mission-based Task Recommendation

In this second part of the analysis, our experiments aim to answer the research question (RQ6d) *Can task recommendation performance benefit from having access to all search queries in a mission, in addition to the best query?*

We report on all combinations of scoring methods (score-based vs. position-based) and aggregator functions (*sum*, *max* or *avg*) in Table 7.6. For query-based task recommendation, we use the best performing approach from the previous section (i.e., LTR with the top 27 features). In order to make a fair comparison, we randomly select a single best query per mission and compare against the query-based LTR restricted to only these 54 best queries. Overall, the results are mixed. In terms of MAP, none of the mission-based methods improve over the query-based approach. On the other hand, when using P@10, all cases in the mission-based approach improve considerably. In terms of NDCG@10, improvements only occur in the score-based method. However, none of these differences is statistically significant for any metric.

To shed some light on the reasons behind this, we perform the following analysis. In Figure 7.5, we plot the differences in NDCG@10, for each individual mission, between a given mission-based configuration and the query-based method. Each positive bar corresponds to a mission where the mission-based method improved over the query-based one, and each negative bar represents a mission where the opposite is true. A key observation we can make here is that there are indeed substantial differences on some missions, but there are about as much wins as losses. There are several possible reasons for this. Since the query-based method uses the best query in a mission, which is identified by an oracle, it may be that, on average, it simply does not help to consider other queries in the mission as those often only bring in noise. Weighing queries in a mission may help to rectify it to some extent; we leave this exploration to future work. Another reason may lie in how the test collection was created. Recall that relevance labeling was done based only on the best queries, and not on other queries in the mission. Those other queries could possibly yield more relevant tasks, which have not been labeled as such.

We now report some of the particular examples when analyzing results of individual missions. Table 7.7 presents best and worst performing missions per configuration, as well as examples of relevant tasks in the test collection, and their rank position in each ranking. A mission with an NDCG@10 difference of 1 on each of the three score-based methods has “install spyware” as its best query. For this mission, all score-based methods rank the only relevant task, “How to Remove Spyware Manually (Windows),” as the top result, meanwhile LTR ranks it beyond the rank 10 cut-off. This mission comprises 10 queries, most of them with an occurrence of the term “spyware.” Only the best query has a verb, “install,” antonym of the verb “remove” in the relevant task, hence the vocabulary drift in the LTR ranking for this query, that the mission-based method is able to address. The score-based method also exhibits a mission with a

Δ	Mission ID (and best query)	Relevant task	Rank position MB	Rank position LTR
Score-based, with any aggregator				
+1.0	1045635_18 ("install spyware")	How to Remove Spyware Manually (Windows)	1	16
-0.68	10733023_1 ("michigan teacher certification")	How to Become Certified to Teach Hunter Safety in Michigan	8, 12	1
Position-based, *sum* or *avg*				
+1.0	1045635_18 ("install spyware")	How to Remove Spyware Manually (Windows)	1	16
-0.5	2161465_3 ("using the normal curve to find probability")	How to Calculate Probability	3	1
-0.53	2385817_13 ("red velvet cake recipes")	Make Red Velvet Cake	3	5
		Make Red Velvet Cupcakes with Pancake Mix	13	14
		Make a Red Velvet Microwavable Mug Cake	14	9
Position-based, *max*				
+0.41	1045635_22 ("find adress by phone number")	Find a Place from a Phone Number	13	3
		Trace the Owner of a Phone Number	1	20
-0.57	2161465_3 ("using the normal curve to find probability")	How to Calculate Probability	4	1
-0.68	10733023_1 ("michigan teacher certification")	How to Become Certified to Teach Hunter Safety in Michigan	5	1

Table 7.7 Queries with the largest ΔNDCG@10 differences, using a mission-based configuration, in comparison with the query-based method. Examples of relevant tasks in the test collection, as well as their rank position in the respective mission-based (MB) and query-based LTR ranking, are also presented.

negative difference of -0.68 for *sum* (Fig. 7.5(a)) and *avg* (Fig. 7.5(c)) aggregators as its worst performing mission. Represented by "michigan teacher certification," the mission has a single relevant task, "How to Become Certified to Teach Hunter Safety in Michigan," ranked as top result by query-based LTR, but only as low ranked as the eighth result by a score-based method. This mission has 5 queries, most of them mentioning terms such as "teaching license exam" and "gid promotions."

For position-based methods (Figs. 7.5(d) - 7.5(e)), the phenomena regarding best and worst differences are very similar, excepting that for the respective worst query, the difference here is smaller (-0.53 to -0.61). A mission with one of the worst differences in all position-based methods is represented by "using the normal curve to find probability" and has a total of 17 queries drifting through several topics related to statistics, such as "linear regression" and "sample size." The single relevant task for this mission, "How to Calculate Probability," is the top recommended task by query-based LTR but only ranked as the fourth result at highest in these position-based methods. The best performing mission for position-based method with *max*, with best query "find adress by phone number," comprises 6 queries, including a couple of more specific ones like "acton california adress by phone." The LTR and mission-based methods behave quite differently for the the relevant tasks of the mission in the test collection.

In answer to RQ6d, we find that our extensions of the query-based approach are promising, but more work is needed to be able to benefit from additional queries in a mission and to reach significant improvements.

7.6 Summary

In this chapter, we have introduced the problem of recommending tasks to users, based on their search queries. Specifically, we have defined the query-based task recommendation problem, and approached it as a content-based recommendation problem, which boils down to scoring tasks from a task repository in response to input queries. We have proposed a method for this problem that combines a strong text-based ranking technique with continuous semantic representations in a learning-to-recommend framework. Further, we have extended our approach to deal with a search mission, i.e., a set made of an arbitrary number of queries that all share the same underlying task. Specifically, we have introduced the problem of mission-based task recommendation, and proposed methods that aggregate the individual query-based recommendations for each query in the mission into mission-level recommended tasks. In this way, task recommendation can be coupled as a support component on top of any existing method for detecting the queries that belong to the same underlying task.

To overcome the lack of a standard test collection to evaluate our novel problems, we have created the first test collection to assess task recommendation. The collection comprises 59 queries, annotated via crowdsourcing with relevance assessments for tasks in a repository of WikiHow articles. Our experimental results have demonstrated the effectiveness of our approach, and in this way, the suitability of the proposed method

as baseline for the query-based task recommendation problem. Finally, feature and query analyses have brought insights concerning the importance and effectiveness of various semantic signals leveraged from distributional representations.

Chapter 8

Conclusions

In this thesis we have investigated approaches to enhance search engines with functionalities for recognizing the underlying tasks and providing support for task completion. In particular, we have studied the utilization of type information in entity retrieval, the construction of a knowledge base of entity-oriented search intents, and the support of task-based search by providing query suggestions and task recommendations.

This chapter presents a summary of our methods and main findings, as well as the conclusions we draw by answering our research questions.

8.1 Results

In this section, we recapitulate our work, by providing answers to the research questions raised in Chapter 1.

Entity Type Information for Entity Retrieval

In the first part of this thesis (Chapters 3 and 4), we focused on understanding the usefulness of type information for ad-hoc entity retrieval. We asked our first research question:

RQ1 How can entity type information be utilized in ad-hoc entity retrieval?

To answer this question, we proposed to study three main dimensions in the problem of *type-aware entity retrieval*: type taxonomy, retrieval model, and representation of hierarchical type information. Assuming "perfect" type-based information provided by a target entity types oracle, we conducted a systematic evaluation of all the combinations

of these dimensions with respect to a strong text-based baseline. We found that type information from large, fine-grained taxonomies provides the best performance. Also, we verified that, even though hierarchical relationships from ancestor types result in improved retrieval effectiveness, using the most specific type assignments is the most effective way of representing entity type information.

In a realistic setting, target entity types are not provided, hence a follow-up challenge we addressed is the problem of target entity type identification. We asked:

RQ2 How can we automatically identify target entity types?

We revisited the *hierarchical target type identification* task, that is, the problem of finding the single most specific type within a reference ontology that is general enough to cover all relevant entities for the query. We also proposed a supervised learning approach for automatically identifying target entity types. Concerning evaluation, we built a test collection of target type assignments for queries. The experimental results showed that our approach is able to significantly outperform methods previously established. In particular, we found that textual similarity features, enriched with distributional semantic representations, measured between the query and the type label, result to be the most effective signals. Furthermore, the results confirmed our hypothesis that using automatic target entity types not only improves entity retrieval performance, but also brings the same, or even higher, level of performance achievable by human target type annotations.

Entity-oriented Search Intents

The second part of this thesis (Chapter 5) aimed firstly to better understand the intents behind entity-oriented queries. We started by asking:

RQ3 What do entity-oriented queries ask for, and how can they be fulfilled?

To address this question, we investigated patterns of queries that typically consist of an entity mention and possibly additional words that refine the intent behind a query. We developed a classification scheme to categorize entity-oriented search intents. Then, we selected a representative sample of type-level refiners, and annotated it using this categorization scheme. We observed that, on average, more than a half of all unique type-level refiners correspond to interacting with external services, while over a quarter of the refiners look for information that may be looked up in a knowledge base.

We then moved on to representing entity-oriented search intents via structured knowledge. Our next research question was:

RQ4 How can we build a knowledge base of entity-oriented search intents?

We answered this question by defining a relational knowledge representation model. We then devised a method for populating this knowledge base, consisting of a four-step pipeline. This approach includes a stage for automatically assigning intent categories to type-level query refiners, and a clustering stage of refiners which express the same underlying intent. In a component-wise evaluation, we found that semantic features perform much better than lexical features for refiner categorization, and that our automatic clustering performs close to using oracle categories. Additionally, we represented each intent uniquely in a knowledge repository, and mapped to it the facts we acquire about its categorization and refiners, via RDF triples. Evaluating in an end-to-end task, we verified that the higher the associated confidence score of a triple, the more likely it is that the triple is correct. Overall, the results show that our approach is able to generate high-quality data.

Task-based Search

In the third part of this thesis (Chapters 6 and 7), we focused on task-based support in search engines. First, we were interested in generating query suggestions to support task-based search. We asked this research question:

RQ5 How can we generate query suggestions for supporting task-based search?

To answer the above question, we adopted a general two-step pipeline approach, comprising the suggestion generation and suggestion ranking components. Addressing query suggestion firstly as an end-to-end task, we introduced a probabilistic generative framework with four components: source importance, document importance, keyphrase relevance, and query suggestion generation. We also proposed various alternatives for estimating the components, as well as experimentally compared these alternatives using the datasets from the TREC Tasks track. Observations consistent across the 2015 and 2016 test collections showed, in particular, a large contribution of query suggestions provided by major web search engines. We then focused on the suggestion generation component of the two-step pipeline, to address additional challenges, such as whether a unified method can produce both query completions and query refinements,

and whether we can avoid relying on resources such as suggestions from major search engines, or query logs. We considered several methods, as well as multiple information sources, and developed a large collection of relevance assessments for task-based query suggestions. The experimental results confirmed that we are able to generate high-quality suggestion candidates. We observed, in particular, that different method-source configurations contribute unique suggestions, and thus it could be beneficial to combine them.

Finally, as web search lends itself to recommendations like the query suggestions previously studied, we proposed to investigate task recommendations as a suitable mechanism to help the user complete her task. We asked:

RQ6 How can we recommend tasks based on search queries and missions?

To answer this question, we introduced a couple of problems for recommending tasks to users, based on their search queries. First, we defined the problem of *query-based task recommendation*, and proposed supervised learning methods that combine well-established text-based matching techniques with continuous semantic representations. We also extended our approach to deal with a search mission, that is, a set made of an arbitrary number of queries that all share the same underlying task. Specifically, we introduced the problem of *mission-based task recommendation*, and proposed methods that aggregate the individual query-based recommendations for each query in the mission into mission-level recommended tasks. Challenged by the lack of a standard test collection, we built the first test collection to evaluate task recommendation. The results showed that our approach can serve as a solid baseline for the query-based task recommendation problem. In this way, task recommendation can be coupled as a support component on top of any existing method for detecting the queries that belong to the same underlying task.

References

[1] Agrawal, R., Gollapudi, S., Halverson, A., and Ieong, S. (2009). Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 5–14.

[2] Alsulmi, M., Sabhnani, K., and Carterette, B. (2016). University of Delaware at TREC 2016 Tasks track. In *Proceedings of The Twenty-Fifth Text REtrieval Conference*, TREC 2016.

[3] Amigó, E., Corujo, A., Gonzalo, J., Meij, E., and de Rijke, M. (2012). Overview of RepLab 2012: Evaluating online reputation management systems. *CEUR Workshop Proceedings*, 1178.

[4] Anagnostopoulos, A., Broder, A. Z., and Carmel, D. (2005). Sampling search-engine results. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 245–256.

[5] Andolina, S., Klouche, K., Peltonen, J., Hoque, M. E., Ruotsalo, T., Cabral, D., Klami, A., Glowacka, D., Floréen, P., and Jacucci, G. (2015). IntentStreams: Smart parallel search streams for branching exploratory search. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, IUI '15, pages 300–305.

[6] Balog, K. (2015). Task-completion engines: A vision with a plan. In *Proceedings of the First International Workshop on Supporting Complex Search Tasks co-located with the 37th European Conference on Information Retrieval*, SCST@ECIR '15.

[7] Balog, K. (2018). *Entity-Oriented Search*, volume 39 of *The Information Retrieval Series*. Springer.

[8] Balog, K., Azzopardi, L., and de Rijke, M. (2006). Formal models for expert finding in enterprise corpora. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 43–50.

[9] Balog, K., Bron, M., and de Rijke, M. (2011). Query modeling for entity search based on terms, categories, and examples. *ACM Trans. Inf. Syst.*, 29(4):22:1–22:31.

[10] Balog, K., de Vries, A. P., Serdyukov, P., Thomas, P., and Westerveld, T. (2010). Overview of the TREC 2009 Entity track. In *Proceedings of The Twentieth Text REtrieval Conference*, TREC '10.

[11] Balog, K. and Neumayer, R. (2012). Hierarchical target type identification for entity-oriented queries. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2391–2394.

[12] Balog, K. and Neumayer, R. (2013). A test collection for entity search in DBpedia. In *Proceedings of the 36th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '13, pages 737–740.

[13] Balog, K., Serdyukov, P., and de Vries, A. P. (2012). Overview of the TREC 2011 Entity track. In *Proceedings of The Twentieth Text REtrieval Conference*, TREC '11.

[14] Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2670–2676.

[15] Benetka, J. R., Balog, K., and Nørvåg, K. (2017). Anticipating information needs based on check-in activity. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 41–50.

[16] Bennett, P. N. and White, R. W. (2015). Mining tasks from the web anchor text graph: MSR notebook paper for the TREC 2015 Tasks track. In *Proceedings of The Twenty-Fourth Text REtrieval Conference*, TREC 2015.

[17] Bergsma, S. and Wang, Q. I. (2007). Learning noun phrase query segmentation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '07, pages 819–826.

[18] Bhatia, S., Majumdar, D., and Mitra, P. (2011). Query suggestions in the absence of query logs. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '11, pages 795–804.

[19] Bi, B., Ma, H., Hsu, B.-J. P., Chu, W., Wang, K., and Cho, J. (2015). Learning to recommend related entities to search users. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 139–148.

[20] Blanco, R., Cambazoglu, B. B., Mika, P., and Torzec, N. (2013). Entity recommendations in web search. In *Proceedings of the 12th International Semantic Web Conference*, ISWC '13, pages 33–48.

[21] Blanco, R., Joho, H., Jatowt, A., and Yu, H. (2017a). Test collection for evaluating actionable knowledge graphs. In *Proceedings of the workshop on Knowledge Graphs for Information Retrieval co-located with the 40th international ACM SIGIR conference on Research and development in information retrieval*, KG4IR@SIGIR '17, pages 32–37.

[22] Blanco, R., Joho, H., Jatowt, A., Yu, H., and Yamamoto, S. (2017b). Overview of actionable knowledge graph generation (AKG) task. In *Proceedings of NTCIR-13*, pages 340–345.

[23] Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., and Vigna, S. (2008). The query-flow graph: Model and applications. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management*, CIKM '08, pages 609–618.

[24] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250.

[25] Bonchi, F., Perego, R., Silvestri, F., Vahabi, H., and Venturini, R. (2012). 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*, SIGIR '12, pages 345–354.

[26] Bortnikov, E., Donmez, P., Kagian, A., and Lempel, R. (2012). Modeling transactional queries via templates. In *Advances in Information Retrieval, Proceedings of the 34th European Conference on IR Research*, ECIR '12, pages 13–24.

[27] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

[28] Broder, A. (2002). A taxonomy of web search. *SIGIR Forum*, 36(2):3–10.

[29] Bron, M., Balog, K., and de Rijke, M. (2010). Ranking related entities: Components and analyses. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management*, CIKM '10, pages 1079–1088.

[30] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 89–96.

[31] Byström, K. (2002). Information and information sources in tasks of varying complexity. *Journal of the American Society for Information Science and Technology*, 53(7):581–591.

[32] Cai, F. and de Rijke, M. (2016). *A Survey of Query Auto Completion in Information Retrieval*. Now Publishers Inc.

[33] Cao, H., Hu, D. H., Shen, D., Jiang, D., Sun, J.-T., Chen, E., and Yang, Q. (2009). Context-aware query classification. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 3–10.

[34] Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. Technical Report MSR-TR-2007-40.

[35] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, Jr., E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI '10, pages 1306–1313.

[36] Chapelle, O., Metzler, D., Zhang, Y., and Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 621–630.

[37] Chu, C. X., Tandon, N., and Weikum, G. (2017). Distilling task knowledge from how-to communities. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 805–814.

[38] Clarke, C. L., Kolla, M., Cormack, G. V., Vechtomova, O., Ashkan, A., Büttcher, S., and MacKinnon, I. (2008). Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 659–666.

[39] Craswell, N. and Szummer, M. (2007). Random walks on the click graph. In *Proceedings of the 30th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 239–246.

[40] Croft, W. B., Bendersky, M., Li, H., and Xu, G. (2011). Query representation and understanding workshop. *SIGIR Forum*, 44(2):48–53.

[41] Croft, W. B., Metzler, D., and Strohman, T. (2009). *Search Engines - Information Retrieval in Practice*. Pearson Education.

[42] Dalton, J., Dietz, L., and Allan, J. (2014). Entity query feature expansion using knowledge base links. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '14, pages 365–374.

[43] de Vries, A. P., Vercoustre, A.-M., Thom, J. A., Craswell, N., and Lalmas, M. (2008). Overview of the INEX 2007 Entity Ranking track. In *Focused Access to XML Documents*, INEX '07, pages 245–251.

[44] Dehghani, M., Rothe, S., Alfonseca, E., and Fleury, P. (2017). Learning to attend, copy, and generate for session-based query suggestion. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1747–1756.

[45] Demartini, G., de Vries, A. P., Iofciu, T., and Zhu, J. (2009). Overview of the INEX 2008 Entity Ranking track. In *Advances in Focused Retrieval*, INEX '08, pages 243–252.

[46] Demartini, G., Firan, C. S., and Iofciu, T. (2008). L3S at INEX 2007. In *Focused Access to XML Documents, 7th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX '08, pages 252–263.

[47] Demartini, G., Firan, C. S., Iofciu, T., Krestel, R., and Nejdl, W. (2010a). Why finding entities in Wikipedia is difficult, sometimes. *Information Retrieval*, 13(5):534–567.

[48] Demartini, G., Iofciu, T., and de Vries, A. P. (2010b). Overview of the INEX 2009 Entity Ranking track. In *Focused Retrieval and Evaluation, 8th International Workshop of the Initiative for the Evaluation of XML Retrieval, Revised and Selected Papers*, INEX '09, pages 254–264.

[49] Ding, H., Zhang, S., Garigliotti, D., and Balog, K. (2018). Generating high-quality query suggestion candidates for task-based search. In *Advances in Information Retrieval, Proceedings of the 40th European Conference on IR Research*, ECIR '18, pages 625–631.

[50] Donato, D., Bonchi, F., Chi, T., and Maarek, Y. (2010). Do you want to take notes?: Identifying research missions in Yahoo! search pad. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 321–330.

[51] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge Vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610.

[52] Fader, A., Zettlemoyer, L., and Etzioni, O. (2014). Open question answering over curated and extracted knowledge bases. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14.

[53] Fernández-Tobías, I. and Blanco, R. (2016). Memory-based recommendations of entities for web search users. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 35–44.

[54] Fleischman, M. and Hovy, E. (2002). Fine grained classification of named entities. In *Proceedings of the 15th International Conference on Computational Linguistics*, COLING '02, pages 1–7.

[55] Fleiss, J. et al. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382.

[56] Fossati, M., Kontokostas, D., and Lehmann, J. (2015). Unsupervised learning of an extensive and usable taxonomy for DBpedia. In *Proceedings of the 11th International Conference on Semantic Systems*, SEMANTICS '15, pages 177–184.

[57] Fourney, A., Mann, R., and Terry, M. (2011). Characterizing the usability of interactive applications through query log analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1817–1826.

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

[59] Fuhr, N. (2017). Some common mistakes in IR evaluation, and how they can be avoided. *SIGIR Forum*, 51(3):32–41.

[60] Gangemi, A., Nuzzolese, A. G., Presutti, V., Draicchio, F., Musetti, A., and Ciancarini, P. (2012). Automatic typing of DBpedia entities. In *Proceedings of the Semantic Web - 11th International Semantic Web Conference*, ISWC '12, pages 65–81.

[61] Garigliotti, D. (2018). A semantic search approach to task-completion engines. In *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '18, page 1457.

[62] Garigliotti, D. and Balog, K. (2016). The University of Stavanger at the TREC 2016 Tasks track. In *Proceedings of The Twenty-Fifth Text REtrieval Conference*, TREC 2016.

[63] Garigliotti, D. and Balog, K. (2017a). Generating query suggestions to support task-based search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1153–1156.

[64] Garigliotti, D. and Balog, K. (2017b). Learning to rank target types for entity-bearing queries. In *Proceedings of the 1st International Workshop on LEARning Next gEneration Rankers co-located with the 3rd ACM International Conference on the Theory of Information Retrieval*, LEARNER@ICTIR '17.

[65] Garigliotti, D. and Balog, K. (2017c). On type-aware entity retrieval. In *Proceedings of the 2017 ACM International Conference on Theory of Information Retrieval*, ICTIR '17, pages 27–34.

[66] Garigliotti, D. and Balog, K. (2018a). IntentsKB: A knowledge base of entity-oriented search intents. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pages 1659–1662.

[67] Garigliotti, D. and Balog, K. (2018b). Towards an understanding of entity-oriented search intents. In *Advances in Information Retrieval, Proceedings of the 40th European Conference on IR Research, ECIR 2018*, pages 644–650.

[68] Garigliotti, D., Hasibi, F., and Balog, K. (2017). Target type identification for entity-bearing queries. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 845–848.

[69] Garigliotti, D., Hasibi, F., and Balog, K. (2019). Identifying and exploiting target entity type information for ad hoc entity retrieval. *Information Retrieval Journal*, 22(3):285–323.

[70] Georgeff, M. P. and Lansky, A. L. (1986). Procedural knowledge. *Proceedings of the IEEE*, 74:1383–1398.

[71] Giuliano, C. (2009). Fine-grained classification of named entities exploiting latent semantic kernels. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 201–209.

[72] Guo, J., Cheng, X., Xu, G., and Zhu, X. (2011). Intent-aware query similarity. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 259–268.

[73] Guo, J., Fan, Y., Ai, Q., and Croft, W. B. (2016). Semantic matching by non-linear word transportation for information retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 701–710.

[74] Hagen, M., Gomoll, J., Beyer, A., and Stein, B. (2013). From search session detection to search mission detection. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval*, OAIR '13, pages 85–92.

[75] Hagen, M., Göring, S., Keil, M., Anifowose, O., Othman, A., and Stein, B. (2015). Webis at TREC 2015: Tasks and Total Recall tracks. In *Proceedings of The Twenty-Fourth Text REtrieval Conference*, TREC 2015.

[76] Hagen, M., Kiesel, J., Adineh, P., Alahyari, M., Fatehifar, E., Bahrami, A., Fichtl, P., and Stein, B. (2016). Webis at TREC 2016: Tasks, Total Recall, and Open Search tracks. In *Proceedings of The Twenty-Fifth Text REtrieval Conference*, TREC 2016.

[77] Hagen, M., Potthast, M., Stein, B., and Bräutigam, C. (2011). Query segmentation revisited. In *Proceedings of the 20th International World Wide Web Conference*, WWW '11, pages 97–106.

[78] Hasibi, F., Balog, K., and Bratsberg, S. E. (2015). Entity linking in queries: Tasks and evaluation. In *Proceedings of the 2015 ACM International Conference on Theory of Information Retrieval*, ICTIR '15, pages 171–180.

[79] Hasibi, F., Balog, K., and Bratsberg, S. E. (2016). Exploiting entity linking in queries for entity retrieval. In *Proceedings of the ACM International Conference on Theory of Information Retrieval*, ICTIR '16, pages 209–218.

[80] Hasibi, F., Nikolaev, F., Xiong, C., Balog, K., Bratsberg, S. E., Kotov, A., and Callan, J. (2017). DBpedia-Entity v2: A test collection for entity search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1265–1268.

[81] Hassan Awadallah, A., White, R. W., Dumais, S., and Wang, Y.-M. (2014a). Struggling or exploring? Disambiguating search sessions. In *Proceedings of the 7th Annual International ACM Conference on Web Search and Data Mining*, WSDM '14, pages 53–62.

[82] Hassan Awadallah, A., White, R. W., Pantel, P., Dumais, S. T., and Wang, Y.-M. (2014b). Supporting complex search tasks. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 829–838.

[83] He, D., Göker, A., and Harper, D. J. (2002). Combining evidence for automatic web session identification. *Information Processing and Management*, 38(5):727 – 742.

[84] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9:1735–1780.

[85] Hu, J., Wang, G., Lochovsky, F., Sun, J.-T., and Chen, Z. (2009). Understanding user's query intent with Wikipedia. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 471–480.

[86] Huang, J., Gao, J., Miao, J., Li, X., Wang, K., Behr, F., and Giles, C. L. (2010). Exploring web scale language models for search query processing. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 451–460.

[87] Jaleel, N. A., Allan, J., Croft, W. B., Diaz, F., Larkey, L. S., Li, X., Smucker, M. D., and Wade, C. (2004). UMass at TREC 2004: Novelty and HARD. In *Proceedings of the Thirteenth Text REtrieval Conference*, TREC '04.

[88] Jämsen, J., Näppilä, T., and Arvola, P. (2008). Entity ranking based on category expansion. In *Focused Access to XML Documents, 7th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX '08, pages 264–278.

[89] Jansen, B. J., Booth, D. L., and Spink, A. (2008). Determining the informational, navigational, and transactional intent of web queries. *Inf. Process. Manage.*, 44(3):1251–1266.

[90] Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446.

[91] Jiang, J., He, D., Han, S., Yue, Z., and Ni, C. (2012). Contextual evaluation of query reformulations in a search session by user simulation. In *Proceedings of the 21st ACM International Conference on Conference on Information and Knowledge Management*, CIKM '12, pages 2635–2638.

[92] Jones, R. and Klinkner, K. L. (2008). Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 699–708.

[93] Jones, R., Rey, B., Madani, O., and Greiner, W. (2006). Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 387–396.

[94] Jung, Y., Ryu, J., Kim, K.-m., and Myaeng, S.-H. (2010). Automatic construction of a large-scale situation ontology by mining how-to instructions from the web. *Web Semant.*, 8(2–3):110–124.

[95] Kanoulas, E., Yilmaz, E., Mehrotra, R., Carterette, B., Craswell, N., and Bailey, P. (2017). TREC 2017 Tasks track overview. In *Proceedings of The Twenty-Sixth Text REtrieval Conference*, TREC '17.

[96] Kaptein, R. and Kamps, J. (2009). Finding entities in Wikipedia using links and categories. In *Advances in Focused Retrieval, 7th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX '09, pages 273–279.

[97] Kaptein, R. and Kamps, J. (2013). Exploiting the category structure of Wikipedia for entity ranking. *Artificial Intelligence*, 194:111–129.

[98] Kaptein, R., Serdyukov, P., de Vries, A. P., and Kamps, J. (2010). Entity ranking using Wikipedia as a pivot. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management*, CIKM '10, pages 69–78.

[99] Kelly, D., Arguello, J., and Capra, R. (2013). NSF workshop on task-based information search systems. *SIGIR Forum*, 47(2):116–127.

[100] Kelly, D., Gyllstrom, K., and Bailey, E. W. (2009). A comparison of query and term suggestion features for interactive searching. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 371–378.

[101] Kim, J. and Croft, W. B. (2012). A field relevance model for structured document retrieval. In *Advances in Information Retrieval, Proceedings of the 34th European Conference on Advances in Information Retrieval*, ECIR'12, pages 97–108.

[102] Kim, J., Xue, X., and Croft, W. B. (2009). A probabilistic retrieval model for semistructured data. In *Advances in Information Retrieval, Proceedings of the 31th European Conference on IR Research*, ECIR 2009, pages 228–239.

[103] Kliegr, T. and Zamazal, O. (2016). LHD 2.0: A text mining approach to typing entities in knowledge graphs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 39:47–61.

[104] Kruschwitz, U., Lungley, D., Albakour, M.-D., and Song, D. (2013). Deriving query suggestions for site search. *JASIST*, 64(10):1975–1994.

[105] Lavrenko, V. (2009). *A Generative Theory of Relevance*, volume 26 of *The Information Retrieval Series*. Springer.

[106] Lavrenko, V. and Croft, W. B. (2001). Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 120–127.

[107] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195.

[108] Li, L., Deng, H., Dong, A., Chang, Y., and Zha, H. (2014). Identifying and labeling search tasks via query-based Hawkes processes. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 731–740.

[109] Li, L., Deng, H., He, Y., Dong, A., Chang, Y., and Zha, H. (2016). Behavior driven topic transition for search task identification. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 555–565.

[110] Li, X., Wang, Y.-Y., and Acero, A. (2008). Learning query intent from regularized click graphs. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 339–346.

[111] Lin, J., Crane, M., Trotman, A., Callan, J., Chattopadhyaya, I., Foley, J., Ingersoll, G., Macdonald, C., and Vigna, S. (2016). Toward reproducible baselines: The open-source IR reproducibility challenge. In *Advances in Information Retrieval, Proceedings of the 38th European Conference on IR Research*, ECIR '16, pages 408–420.

[112] Lin, T., Mausam, M., and Etzioni, O. (2012a). No noun phrase left behind: Detecting and typing unlinkable entities. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 893–903.

[113] Lin, T., Pantel, P., Gamon, M., Kannan, A., and Fuxman, A. (2012b). Active objects: Actions for entity-centric search. In *Proceedings of the 21st International World Wide Web Conference*, WWW '12, pages 589–598.

[114] Ling, X. and Weld, D. S. (2012). Fine-grained entity recognition. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI '12, pages 94–100.

[115] Lopez, V., Unger, C., Cimiano, P., and Motta, E. (2013). Evaluating question answering over linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 21:3–13.

[116] Lucchese, C., Nardini, F. M., Orlando, S., and Tolomei, G. (2016). Learning to rank user queries to detect search tasks. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 157–166.

[117] Lucchese, C., Orlando, S., Perego, R., Silvestri, F., and Tolomei, G. (2011). Identifying task-based sessions in search engine query logs. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 277–286.

[118] Lucchese, C., Orlando, S., Perego, R., Silvestri, F., and Tolomei, G. (2013). Discovering tasks from search engine query logs. *ACM Trans. Inf. Syst.*, 31(3):14:1–14:43.

[119] Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, EMNLP '15, pages 1412–1421.

[120] Macdonald, C., Santos, R. L. T., and Ounis, I. (2013). The whens and hows of learning to rank for web search. *Inf. Retr.*, 16(5):584–628.

[121] MacKay, B. and Watters, C. (2009). Building support for multi-session tasks. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 4273–4278.

[122] Marchionini, G. (2006). Exploratory search: From finding to understanding. *Commun. ACM*, 49(4):41–46.

[123] Mausam, M., Schmitz, M., Bart, R., Soderland, S., and Etzioni, O. (2012). Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 523–534.

[124] Medelyan, A. (2015). Modified RAKE algorithm. https://github.com/zelandiya/RAKE-tutorial. Accessed: 2017-01-23.

[125] Mehrotra, R., Bhattacharya, P., and Yilmaz, E. (2016a). Characterizing users' multi-tasking behavior in web search. In *Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval*, CHIIR '16, pages 297–300.

[126] Mehrotra, R., Bhattacharya, P., and Yilmaz, E. (2016b). Deconstructing complex search tasks: A Bayesian nonparametric approach for extracting sub-tasks. In *Proceedings of NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 599–605. The Association for Computational Linguistics.

[127] Mehrotra, R., Hassan Awadallah, A., Shokouhi, M., Yilmaz, E., Zitouni, I., El Kholy, A., and Khabsa, M. (2017). Deep sequential models for task satisfaction prediction. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 737–746.

[128] Mehrotra, R. and Yilmaz, E. (2015). Towards hierarchies of search tasks and subtasks. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 73–74.

[129] Mehrotra, R. and Yilmaz, E. (2017a). Extracting hierarchies of search tasks and subtasks via a Bayesian nonparametric approach. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 285–294.

[130] Mehrotra, R. and Yilmaz, E. (2017b). Task embeddings: Learning query embeddings using task context. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 2199–2202.

[131] Mei, Q., Klinkner, K. L., Kumar, R., and Tomkins, A. (2009). An analysis framework for search sequences. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management*, CIKM '09, pages 1991–1994.

[132] Metzler, D. and Croft, W. B. (2005). A Markov Random Field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 472–479.

[133] Metzler, D. and Croft, W. B. (2007). Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274.

[134] Mika, P. (2013). Entity Search on the Web. In *Proceedings of the 22nd International World Wide Web Conference*, WWW '13, pages 1231–1232.

[135] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems*, NIPS '13, pages 3111–3119.

[136] Mikolov, T., Yih, W., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proceedings of Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '13, pages 746–751.

[137] Miller, R. G. J. (1981). *Simultaneous Statistical Inference*. Springer.

[138] Mitra, B. and Craswell, N. (2015). Query auto-completion for rare prefixes. In *Proceedings of the 24th ACM International Conference on Conference on Information and Knowledge Management*, CIKM '15, pages 1755–1758.

[139] Mitra, B., Nalisnick, E. T., Craswell, N., and Caruana, R. (2016). A dual embedding space model for document ranking. *CoRR*, abs/1602.01137.

[140] Nakashole, N., Tylenda, T., and Weikum, G. (2013). Fine-grained semantic typing of emerging entities. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, ACL '13, pages 1488–1497.

[141] Neumayer, R., Balog, K., and Nørvåg, K. (2012). On the modeling of entities for ad-hoc entity search in the web of data. In *Advances in Information Retrieval, Proceedings of the 34th European Conference on IR Research*, ECIR '12, pages 133–145.

[142] Nuzzolese, A. G., Gangemi, A., Presutti, V., and Ciancarini, P. (2012). Type inference through the analysis of Wikipedia links. In *Proceedings of the Linked Data on the Web workshop co-located with the 21st International World Wide Web Conference*, LDOW@WWW '12.

[143] Ogilvie, P., Callan, J., and Callan, J. (2003). Combining document representations for known-item search. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 143–150.

[144] Ozertem, U., Chapelle, O., Donmez, P., and Velipasaoglu, E. (2012). Learning to suggest: A machine learning framework for ranking query suggestions. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 25–34.

[145] Pareti, P., Klein, E., and Barker, A. (2014a). A semantic web of know-how: Linked Data for community-centric tasks. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 1011–1016.

[146] Pareti, P., Testu, B., Ichise, R., Klein, E., and Barker, A. (2014b). Integrating know-how into the Linked Data cloud. In *Knowledge Engineering and Knowledge Management*, pages 385–396.

[147] Park, D. H. and Chiba, R. (2017). A neural language model for query auto-completion. In *Proceedings of the 40th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '17, pages 1189–1192.

[148] Pass, G., Chowdhury, A., and Torgeson, C. (2006). A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, InfoScale '06.

[149] Paulheim, H. and Bizer, C. (2013). Type inference on noisy RDF data. In *Proceedings of the Semantic Web - 12th International Semantic Web Conference*, ISWC '13, pages 510–525.

[150] Paulheim, H. and Bizer, C. (2014). Improving the quality of Linked Data using statistical distributions. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):63–86.

[151] Pehcevski, J., Thom, J. A., Vercoustre, A.-M., and Naumovski, V. (2010). Entity ranking in Wikipedia: Utilising categories, links and topic difficulty prediction. *Information Retrieval*, 13(5):568–600.

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

[153] Pound, J., Mika, P., and Zaragoza, H. (2010). Ad-hoc object retrieval in the web of data. In *Proceedings of the 19th International World Wide Web Conference*, WWW '10, pages 771–780.

[154] Radlinski, F. and Joachims, T. (2005). Query chains: Learning to rank from implicit feedback. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '05, pages 239–248.

[155] Rafailidis, D. and Crestani, F. (2017). Learning to rank with trust and distrust in recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 5–13.

[156] Rahman, A. and Ng, V. (2010). Inducing fine-grained semantic classes via hierarchical and collective classification. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 931–939.

[157] Raman, K., Bennett, P. N., and Collins-Thompson, K. (2013). Toward whole-session relevance: Exploring intrinsic diversity in web search. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 463–472.

[158] Raviv, H., Carmel, D., and Kurland, O. (2012). A ranking framework for entity oriented search using Markov Random Fields. In *Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search*, JIWES '12, pages 1:1–1:6.

[159] Reinanda, R., Meij, E., and de Rijke, M. (2015). Mining, ranking and recommending entity aspects. In *Proceedings of the 38th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '15, pages 263–272.

[160] Risvik, K. M., Mikolajewski, T., and Boros, P. (2003). Query segmentation for web search. In *Proceedings of the Twelfth International World Wide Web Conference*, WWW'03.

[161] Rizzo, G., Pereira, B., Varga, A., van Erp, M., and Cano-Basave, A. E. (2017). Lessons learnt from the Named Entity rEcognition and Linking (NEEL) challenge series. *Semantic Web*, 8(5):667–700.

[162] Robertson, S. (1977). The probability ranking principle in IR. *Journal of Documentation*, 33:294–304.

[163] Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

[164] Robertson, S., Zaragoza, H., and Taylor, M. (2004). Simple BM25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 42–49.

[165] Rose, D. E. and Levinson, D. (2004). Understanding user goals in web search. In *Proceedings of the 13th International World Wide Web Conference*, WWW '04, pages 13–19.

[166] Rosenberg, A. and Hirschberg, J. (2007). V-Measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '07, pages 410–420.

[167] Ruthven, I. (2008). Interactive Information Retrieval. *Annual Review of Information Science and Technology*, 42(1):43–91.

[168] Salton, G. and McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc.

[169] Sawant, U. and Chakrabarti, S. (2013). Learning joint query interpretation and response ranking. In *Proceedings of the 22nd International World Wide Web Conference*, WWW '13, pages 1099–1109.

[170] Shen, D., Sun, J.-T., Yang, Q., and Chen, Z. (2006). Building bridges for web query classification. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 131–138.

[171] Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., and Hanjalic, A. (2013a). GAPfm: Optimal top-n recommendations for graded relevance domains. In *22nd ACM International Conference on Information and Knowledge Management*, CIKM '13, pages 2261–2266.

[172] Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., and Hanjalic, A. (2013b). xCLiMF: Optimizing expected reciprocal rank for data with multiple levels of relevance. In *Proceedings of the Seventh ACM Conference on Recommender Systems*, RecSys '13, pages 431–434.

[173] Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A., and Oliver, N. (2012a). TFMAP: Optimizing MAP for top-n context-aware recommendation. In *The 35th International ACM SIGIR conference on research and development in Information Retrieval*, SIGIR '12, pages 155–164.

[174] Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., and Hanjalic, A. (2012b). CLiMF: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 139–146.

[175] Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., and Nie, J.-Y. (2015). A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, CIKM '15, pages 553–562.

[176] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). YAGO: A core of semantic knowledge. In *Proceedings of the 16th International World Wide Web Conference*, WWW '07, pages 697–706.

[177] Szpektor, I., Gionis, A., and Maarek, Y. (2011). Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 47–56.

[178] Tan, B. and Peng, F. (2008). Unsupervised query segmentation using generative language models and Wikipedia. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 347–356.

[179] Tanon, T. P., Vrandecic, D., Schaffert, S., Steiner, T., and Pintscher, L. (2016). From Freebase to Wikidata: The great migration. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016*, pages 1419–1428.

[180] Toms, E. G., Villa, R., and Mccay-Peet, L. (2013). How is a search system used in work task completion? *J. Inf. Sci.*, 39(1):15–25.

[181] Tonon, A., Catasta, M., Demartini, G., Cudré-Mauroux, P., and Aberer, K. (2013). TRank: Ranking entity types using the web of data. In *Proceedings of the Semantic Web - 11th International Semantic Web Conference*, ISWC '13, pages 640–656.

[182] Tonon, A., Catasta, M., Prokofyev, R., Demartini, G., Aberer, K., and Cudré-Mauroux, P. (2016). Contextualized ranking of entity types based on knowledge graphs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 37–38:170–183.

[183] Tran, T. A., Schwarz, S., Niederée, C., Maus, H., and Kanhabua, N. (2016). The forgotten needle in my collections: Task-aware ranking of documents in semantic information space. In *Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval*, CHIIR '16, pages 13–22.

[184] Valcarce, D., Bellogín, A., Parapar, J., and Castells, P. (2018). On the robustness and discriminative power of information retrieval metrics for top-n recommendation. In *Proceedings of the Twelfth ACM Conference on Recommender Systems*, RecSys '18, pages 260–268.

[185] Vallet, D. and Zaragoza, H. (2008). Inferring the most important types of a query: A semantic approach. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 857–858.

[186] van Erp, M., Mendes, P. N., Paulheim, H., Ilievski, F., Plu, J., Rizzo, G., and Waitelonis, J. (2016). Evaluating entity linking: An analysis of current benchmark datasets and a roadmap for doing a better job. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, LREC '16, pages 4373–4379.

[187] Vercoustre, A.-M., Pehcevski, J., and Thom, J. A. (2008). Using Wikipedia categories and links in entity ranking. In *Focused Access to XML Documents, 6th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX '07, pages 321–335.

[188] Verma, M., Kanoulas, E., Yilmaz, E., Mehrotra, R., Carterette, B., Craswell, N., and Bailey, P. (2016). Overview of the TREC Tasks track 2016. In *Proceedings of The Twenty-Fifth Text REtrieval Conference*, TREC '16.

[189] Verma, M. and Yilmaz, E. (2014). Entity oriented task extraction from query logs. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 1975–1978.

[190] Verma, M. and Yilmaz, E. (2016). Category oriented task extraction. In *Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval*, CHIIR '16, pages 333–336.

[191] Voorhees, E. M. (2014). The effect of sampling strategy on inferred measures. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '14, pages 1119–1122.

[192] Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A free collaborative knowledge base. *Commun. ACM*, 57(10):78–85.

[193] Wang, H., Song, Y., Chang, M.-W., He, X., White, R. W., and Chu, W. (2013). Learning to extract cross-session search tasks. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 1353–1364.

[194] Weerkamp, W., Balog, K., and Meij, E. J. (2009). A generative language modeling approach for ranking entities. In *Advances in Focused Retrieval, 7th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX '09, pages 292–299.

[195] White, R. W. and Huang, J. (2010). Assessing the scenic route: Measuring the value of search trails in web logs. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 587–594.

[196] Wu, Q., Burges, C. J., Svore, K. M., and Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, 13:254–270.

[197] Yaghoobzadeh, Y., Adel, H., and Schütze, H. (2017). Noise mitigation for neural entity typing and relation extraction. In *Proceedings of the 2017 Conference of the European Chapter of the Association for Computational Linguistics*, EACL '17, pages 1183–1194.

[198] Yaghoobzadeh, Y. and Schütze, H. (2015). Corpus-level fine-grained entity typing using contextual information. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, EMNLP '15, pages 715–725.

[199] Yang, P., Fang, H., and Lin, J. (2018). Anserini: Reproducible ranking baselines using Lucene. *J. Data and Information Quality*, 10(4):16:1–16:20.

[200] Yilmaz, E., Verma, M., Mehrotra, R., Kanoulas, E., Carterette, B., and Craswell, N. (2015). Overview of the TREC 2015 Tasks track. In *Proceedings of The Twenty-Fourth Text REtrieval Conference*, TREC '15.

[201] Yin, X. and Shah, S. (2010). Building taxonomy of web search intents for name entity queries. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1001–1010.

[202] Yosef, M. A., Bauer, S., Spaniol, J. H. M., and Weikum, G. (2012). HYENA: Hierarchical tYpe classification for Entity NAmes. In *Proceedings of the 25th International Conference on Computational Linguistics*, COLING '12, pages 1361–1370.

[203] Yu, X., Ma, H., Hsu, B.-J. P., and Han, J. (2014). On building entity recommender systems using user click log and Freebase knowledge. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 263–272.

[204] Zhai, C. (2008). Statistical language models for Information Retrieval: A critical review. *Foundations and Trends in Information Retrieval*, 2(3):137–213.

[205] Zhang, S. and Balog, K. (2017). Design patterns for fusion-based object retrieval. In *Advances in Information Retrieval, Proceedings of the 39th European Conference on IR Research*, ECIR '17, pages 684–690.

[206] Zheng, Z., Zha, H., Zhang, T., Chapelle, O., Chen, K., and Sun, G. (2007). A general boosting method and its application to learning ranking functions for web search. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 1697–1704.

[207] Zhiltsov, N., Kotov, A., and Nikolaev, F. (2015). Fielded sequential dependence model for ad-hoc entity retrieval in the web of data. In *Proceedings of the 38th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '15, pages 253–262.

[208] Zhu, J., Song, D., and Rüger, S. (2008). Integrating document features for entity ranking. In *Focused Access to XML Documents, 7th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX '08, pages 336–347.

Appendix A

Resources

We have made publicly available the resources developed within the studies presented in this thesis.

Identifying and Utilizing Target Entity Type Information for Entity Retrieval

The resources developed for the experiments of Chapters 3 and 4 are available at http://bit.ly/irj-types. These include the source code for type-aware entity retrieval, the test collection (DBpedia-Entity v2 [80]), and all the run files. The repository also contains the test collection of target entity types built in Chapter 4, as well as all the run files for target entity type identification, and the training set with all the precomputed features for learning to rank target entity types.

Understanding and Modeling Entity-Oriented Search Intents

Two repositories provide the resources related to Chapter 5. First, the repository at http://bit.ly/ecir2018-intents contains our collection of categorized type-level refiners, developed in Section 5.2. Second, the knowledge base of entity-oriented search intents is made available at http://bit.ly/cikm2018-intentsKB. It includes the outputs of all the stages described in Section 5.6.

Suggesting Queries to Support Task-Based Search

The resources related to Chapter 6 are available in two repositories. First, we make available at http://bit.ly/sigir2017-tasks the run files corresponding to all the experiments of Section 6.4, as well as the datasets provided by the TREC Tasks track [200, 188]. Second, the repository at http://bit.ly/ecir2018-suggestions contains the resources developed for the experiments of Section 6.6. These include the collection of relevance assessments, and all the run files.

Recommending Tasks based on Search Queries and Missions

Finally, we release the resources related to Chapter 7 at http://bit.ly/task-rec. This repository contains the corpus of search queries and missions, the corpus of procedural search missions, the test collection that we built for task recommendation, as well as all the run files.