Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Martin Bondkall Gjerde
Ebba Louise Toreld Fingarsen

Assessing Ranking Models' Behavior for Semantic Entity Retrieval

Master's thesis in Informatics
Supervisor: Trond Aalberg
June 2021

NTNU
Kunnskap for en bedre verden

Martin Bondkall Gjerde
Ebba Louise Toreld Fingarsen

Assessing Ranking Models' Behavior for Semantic Entity Retrieval

Master's thesis in Informatics
Supervisor: Trond Aalberg
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for en bedre verden

Abstract

Today, an increasing amount of information is stored in a structured or semi-structured manner. Semantic data is a conceptual model for structuring data that characteristically contains a small amount of text sparsely distributed over many properties. These data structures can for instance be used to represent entities. Traditional Information Retrieval (IR) methods purely rely on unstructured text documents, hence do not take semantic structures into account. Modern approaches to search, such as the state-of-the-art ranking model BM25 and its fielded counterpart, BM25F, have become increasingly common. Previous research regarding the use of these ranking models when searching in semantic data shows equivocal results. This makes it difficult to know how each ranking model behaves and which one should be used in different environments.

In this thesis, the behavior of Lucene Fulltext Search (Vector Space Model), BM25, and BM25F are compared in an entity retrieval setting. Each model is evaluated on two semantic datasets gathered from Wikidata, with a total of ten different queries per dataset. One disease dataset containing entities with several properties with discriminatory terms, and one movie dataset containing fewer properties with less discriminatory keywords. This was done by gathering the perceived relevancy of the ranked search results for each model through a platform for user evaluation specifically developed for this thesis. The user study gathered relevancy assessments from 26 respondents totaling 8130 evaluated entities. The evaluation metrics used to evaluate each model were DCG, NDCG, and the Kappa coefficient.

The tested ranking models showed promising results for users searching in semantic data. BM25F performed the best on the disease dataset with a mean average NDCG score of 0.858, while Lucene Fulltext performed the best on the movie dataset with a mean average NDCG score of 0.836. The results show that BM25F is able to capture the underlying structure to its advantage when ranking, but struggles when properties do not contain unique and discriminatory information. This is to a large degree due to its modified saturation function favoring several terms matched in a single property instead of a few matched across multiple properties.

Sammendrag

I dag lagres en økende mengde informasjon på en strukturert eller halvstrukturert måte. Semantiske data er en konseptuell modell for strukturering av data som karakteristisk inneholder en liten mengde tekst sparsomt fordelt over mange felt. Disse datastrukturene kan for eksempel brukes til å representere entiteter. Tradisjonelle metoder for informasjonsgjenfinning (IR) tar kun hensyn til ustrukurerte tekst-dokumenter og tar derfor ikke hensyn til den semantiske strukturen. Moderne tilnærminger til søk som rangeringsmodellen BM25 og dens feltbaserte motpart, BM25F, har blitt stadig vanligere. Tidligere forskning angående bruken av disse rangeringsmodellene ved søk i semantiske data viser tvetydige resultater. Dette gjør det vanskelig å vite hvordan ulike rangeringsmodeller oppfører seg og hvilken modell som skal brukes i ulike miljøer.

I denne oppgaven sammenlignes oppførselen til Lucene Fulltext Search (Vektorrom modellen), BM25 og BM25F som modeller for entitetsgjenfinning. Hver modell blir evaluert på to semantiske datasett hentet fra Wikidata, med totalt ti forskjellige spørringer per datasett. Ett sykdomsdatasett som inneholder entiteter med flere felt med diskriminerende nøkkelord, og ett filmdatasett som inneholder færre felt med mindre diskriminerende nøkkelord. Dette ble gjort ved å samle den opplevde relevansen av de rangerte søkeresultatene for hver modell gjennom en plattform for brukerevaluering spesielt utviklet for denne oppgaven. Studien samlet relevansvurderinger fra 26 respondenter som ga til sammen 8130 evaluerte entiteter. Evalueringsmetodene som ble brukt for å evaluere modellene var DCG, NDCG og Kappa-koeffisienten.

De testede rangeringsmodellene viste lovende resultater for brukere som søker i semantiske data. BM25F presterte best på sykdomsdatasettet med en gjennomsnittlig NDCG-verdi på 0,858, mens Lucene Fulltext presterte best på filmdatasettet med en gjennomsnittlig NDCG-verdi på 0,836. Resultatene viser at BM25F er i stand til å utnytte den underliggende strukturen til sin fordel når den rangerer, men sliter når entitetsfelt ikke inneholder unik og diskriminerende informasjon. Dette skyldes i stor grad den modifiserte metningsfunksjonen som favoriserer entiterer hvor flere nøkkelord matcher i et enkelt felt i stedet for noen få som matcher over flere felt.

Preface

This thesis was written in autumn 2020 to spring 2021 for the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). The project aims to compare information retrieval ranking models' behavior in domain-specific entity retrieval environments. We would like to thank the participants of the research for taking their time to partake in the survey.

Trond Aalberg served as the supervisor for this thesis. We would like to express our gratitude for his knowledge, feedback, and motivation throughout this project. His guidance was imperative for the improvement of this thesis.

Lastly, we would like to thank our close family and friends for their continuous support and motivation along the way.

Martin Bondkall Gjerde
Ebba Louise Toreld Fingarsen
Trondheim, May 25, 2021

Contents

Figures

Tables

Acronyms

BM Boolean model.

BoW bag-of-words.

DCG Discounted cumulative gain.

IDCG Ideal discounted cumulative gain.

IDF Inverse document frequency.

INEX Evaluation of XML retrieval.

IR Information retrieval.

ISP Information search process.

KB Knowledge base.

KG Knowledge graph.

KR Knowledge repository.

MAP Mean average precision.

NDCG Normalized discounted cumulative gain.

RDF Resource description framework.

TF Term frequency.

TF-IDF Term frequency · inverse document frequency.

TREC Text REtrieval Conference.

URI Uniform resource identifiers.

VSM Vector space model.

W3C World Wide Web Consortium.

Chapter 1

Introduction

1.1 Background and Motivation

Traditionally, search is about "[...] finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)." [1, p. 1]. These documents are easily accessible through popular search engines like Google, Bing, and DuckDuckGo utilizing well-researched retrieval algorithms and ranking models. However, as the semantic web continues to grow, the demand for efficient approaches to search in structured or semi-structured material increases. Search in semantic data is an increasingly important task as it has been estimated that more than 40% of web search queries target specific objects or things [2–4]. These are commonly called entities and are represented as structured documents. This creates new challenges and requirements. Characteristically, semantic data contain less text compared to unstructured text documents. Additionally, the semantic data is commonly represented in a graph-like manner with documents as nodes and relations as edges between them. Traditional *information retrieval* (IR) models are solely based on unstructured text documents, hence does not take semantic structures into account.

To address these challenges, researchers have developed and evaluated new models [5–14] and indexing techniques [6, 15]. Each approach modifies models or techniques used for unstructured document retrieval and ranking to take advantage of the underlying structure. Most researchers focus on the model developed in their research compared to some baseline instead of comparing to other competitive models. The research that does compare several modified models to their unstructured counterparts, however, shows equivocal results [6, 9, 11]. Thus, there is a need for a standardized evaluation approach and more research to further explore the ranking models' behaviors and performances in different environments.

This thesis presents a platform developed to evaluate ranking models in an entity retrieval setting using relevancy assessments collected from end-users. The

platform is additionally used to conduct a comparative evaluation study on selected ranking models to compare their behavior and explore potential strengths and weaknesses when searching in semantic data.

1.2 Research Objectives

This thesis aims to answer the following research questions:

RQ1: How do IR ranking models perform for users searching in semantic data?

> **RQ1.1:** What are the strengths and weaknesses of different IR ranking models?
> **RQ1.2:** How do the ranking models behave in an entity retrieval setting?
> **RQ1.3:** How do fielded ranking models' performances compare?

RQ2: What methods are suitable for evaluating entity retrieval?

1.3 Method and Approach

The first step towards answering the research objectives was a literature study to get an overview of relevant theory and already published research. The literature review focused on search in semantic data, graph structures, entity retrieval, and traditional information retrieval algorithms.

The literature study was followed by an applied research phase where a platform for evaluating entity retrieval was proposed. The platform includes indexing techniques and ranking models for search in semantic data. Due to the lack of standardized test sets for evaluating retrieval algorithms and ranking models in semantic data, the proposed platform allows end-users to give relevancy assessments to query results. In this research, the ranking models are evaluated on two different domain-specific datasets imported from the Wikidata knowledge base. The results from the user testing were observed and analyzed to discover the strengths and weaknesses of each ranking model in different environments. Furthermore, the results are compared to similar research to see how the findings correspond to related experiments.

1.4 Contribution

The main contribution of this thesis is the investigation into how existing IR ranking models behave on a domain-specific semantic knowledge base in an entity retrieval setting. A well-structured overview of the benefits and challenges with each model is discussed, and measurements of users' perceived relevance of the top-k results ranked by each model is presented.

Additionally, a platform to evaluate ranking models on semantic data was developed. This includes a framework to import data, a new approach to general-purpose indexing for entity retrieval, a selection of ranking models, accommodation for inclusion of new models, and a web application to gather relevancy assessments from end-users. This is publicly available and can aid in replicating experiments, executing testing on other datasets, or working as a foundation for further development and testing of new models.

1.5 Thesis Structure

Chapter 2: Introduces the fundamental theory, including IR ranking models for both unstructured and structured data, as well as prior research done, and their limiting factors.

Chapter 3: Concisely describes concepts and methods used to approach the exploration of the research questions.

Chapter 4: Presents the architecture and implementation of the proposed prototype developed in this research to aid in testing. Here, the implementation of the indexing and ranking techniques are thoroughly explained.

Chapter 5: Presents the process of data gathering for ranking model evaluation, in addition to explaining in detail how the testing was conducted.

Chapter 6: Presents and evaluates the findings from the research, as well as discusses their implications and validity.

Chapter 7: Concludes how each research question has been answered, which research limitations are present in this thesis, and proposes further research.

Chapter 2

Theory

The objective of this chapter is to place the research within a theoretical context. This is done by presenting the underlying theory as well as prior research within the field and its challenges.

2.1 Searching for Information

Kuhlthau [16] explains that the *information search process* (ISP) consists of six stages depicted in figure 2.1.

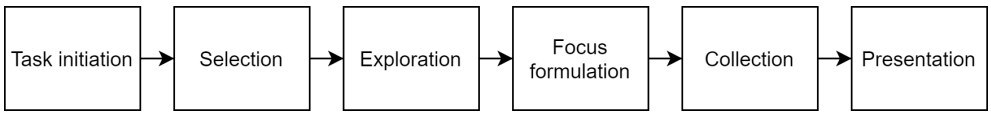


Figure 2.1: The six stages of ISP

When searching for information, users tend to begin with a broader topic before gradually becoming more focused on one topic or point towards the end. This is because the user develops confidence in their information needs and therefore narrows their perspective and ultimately searches.

Initiation is when a lack of knowledge is first observed, *selection* is when the general area of interest is recognized, and *exploration* involves investigating the area of information to further the understanding about the topic. When the user has reached this point, a more focused search is constructed during the *formulation* stage. Here the focus is narrowed and the user selects ideas and concepts to further examine. The process continues by gathering more focused information about the chosen topic in *collection*, before redundant information is removed in *presentation*, leaving a new understanding.

2.1.1 Queries

When searching for information, users can express their needs in several ways. Balog [17, p.13] categorizes the different search paradigms as the following:

- *Keyword queries* are considered free-text queries where the user can freely express their information need. This can however be challenging as the formulations may be vague and differ from one user to another.
- *keyword++* queries improve the precision of the query by complementing the query with some additional information or context. This can be done by asking the user to select a category or filter along with their query.
- *Structured queries* are formulated in a specific language such as SQL. These queries are precise, but require some expertise from the users about both the language and the structure of the data to be queried to be effectively used.
- *Natural language queries* are usually expressed similar to a daily conversation formulated as a question. This is more common with voice-activated search engines like Amazon's Alexa or Apple's Siri.
- *Zero-queries* aim to anticipate and access the users' information needs without a query. Here user context is utilized to proactively answer information needs.

2.1.2 Results

Queries need to be handled in different ways and can therefore solicit multiple types of results. To answer the users' queries, an understanding of the underlying goals and intent behind the query is needed. This can be done by analyzing the query structure, recognizing the need for specific services, referenced items, or concrete ideas that can indicate the information need.

The process of searching for and presenting answers is also substantially affected by the structure of the data. When browsing the web, *Unstructured data* is commonly encountered, appearing as free-text in documents. Some examples are news articles, emails, and tweets. *Structured data* is organized following strict rules and is typically stored in tables in a relational database. Finally, there is *semi-structured data* which falls somewhere in between. It is characterized by the lack of a formal and strict schema, making it more flexible than its structured counterpart. Balog [18, p. 76] describes it as "[...] the schema is contained within the data and is evolving together with the content, thereby making it 'self-describing'." Examples of semi-structured data are JSON, XML, and RDF.

The main search engines predominantly present their results as a title, a link, and a section of the most relevant text. When the information need is more complex, the user has to further investigate each result in order to find an answer. Some search engines also provide the results in the form of a list, such as the

highest mountains in the world, or as a card to summarize simple information that is commonly searched for.

2.2 Representing and Storing Semantic Data

This thesis mainly focuses on keyword search in semantic data. Semantic data involves retaining the semantic information surrounding the data intact by providing context through different types of structures such as relations between data. The role of semantic information is to describe the meaning of a concept. Using semantics, two concepts can be described as equivalent even though they might be stored in different knowledge bases, in different forms, and with different names [19]. Some of the structures used to represent semantic data will be presented in this section.

2.2.1 Knowledge Base and Knowledge Graph

One way to store data and descriptive information about the data is through a *knowledge repository* (KR). This can be done in a semi-structured or structured way. The data is represented by semantic categories and descriptions or properties that further define them. A structured version of the KR with a set of assertions about the data it contains is defined as a *knowledge base* (KB) [17, p.5].

KBs aims to reflect the real world which includes a lot of ever-changing details. Because of this dynamic nature, the KB will always be incomplete. Today, this imperfect depiction is embraced and instead of relying on fully developed ontologies, the focus lies on lightweight relations between data. A KB with this focus can be referred to as a *knowledge graph* (KG). A KG allows the data to be represented as nodes and the relationships between them as edges. One way to represent a KG would be the *Resource description framework* (RDF) triple structure discussed in the following section.

2.2.2 Resource Description Framework

RDF is a standardized data model part of the World Wide Web Consortium (W3C) specifications developed to represent and interchange interconnected data[1]. The RDF data model suitable mechanisms to represent semantic information by facilitating semantic interoperability [20, p. 65]. The model concerns specific instances of data and their relations and can be seen as an extension to the linking structure of the web using *uniform resource identifiers* (URI) syntax [17, p. 38]. URI is a syntax restricted sequence of characters used as identifiers for resources [21]. Resources can be anything with identity and will in this context correlate to the specific data instance. Using RDF, URI can, in addition to link two ends, represent

[1]RDF specifications for N-triples from W3C: `https://www.w3.org/TR/2014/REC-n-triples-20140225/`

the relationships between resources. These relationships are often referred to as *triples* and is expressed as *subject-predicate-object* [5].

Here, the *predicate* denotes traits or aspects that expresses a relationship between the resource, *subject*, and the *object*. Objects can be other resources or values such as numbers and strings of text. Figure 2.2 shows an example of such a triple, namely a relation between the hero Spider-Man and the villain Green-Goblin.

```
1       <http://example.org/#spiderman>
2       <http://www.perceive.net/schemas/relationship/enemyOf>
3       <http://example.org/#green-goblin>
4
```

Figure 2.2: RDF triple example

There are many advantages to the RDF model. The linking structure formed by RDF triples creates a directed graph commonly used for easy-to-understand visualization of the data. As a result, it has all the advantages of structuring information using graphs. These advantages include intuitive, agile, and scalable representation of data while having highly optimized and high-performance search compared to relational databases [22]. Figure 2.3 shows the graph visualization of the RDF triple in figure 2.2.

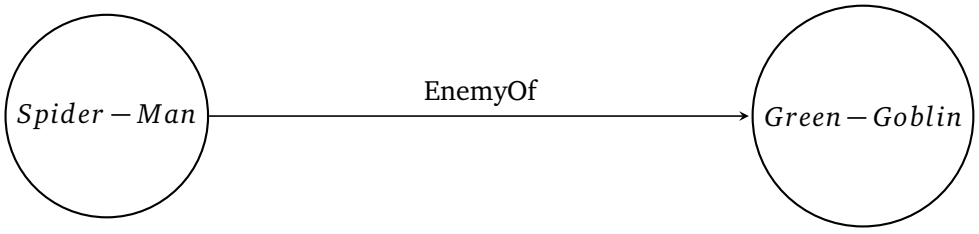


Figure 2.3: Graph visualization of RDF triple

Additionally, resources within the RDF model are flexible as modification consists of adding and removing RDF triples. If Spider-Man and Green-Goblin suddenly became friends, you would remove the *EnemyOf* relation between them and add a new *FriendsWith* relation. This is possible without affecting the rest of the graph.

SPARQL

SPARQL is the query language used to retrieve and manipulate data stored in the RDF format [23]. It can retrieve the data both in table format and RDF triples. SPARQL is a powerful query language, but structured queries require expertise in both the language and the data to be queried. This makes it unfit for the average end-user.

2.3 Entities

A uniquely identified instance of data correlating to a concept or an object is referred to as an *entity*. These entities are characterized by their types, attributes, and relationships to other entities [17]. Types can be seen as categories or instances. An example would be that Usain Bolt is an instance of 'sprinter' which is a subtype of 'athlete'. Attributes are referred to as sets of features or characteristics that the entity possesses, and relationships describe how two entities are associated with one another. A practical example of an entity and its properties is present when searching for "Matt Damon" in the search engine DuckDuckGo. As illustrated in 2.4, the query will result in an *entity card* on the right-hand side showing information about him and his career.

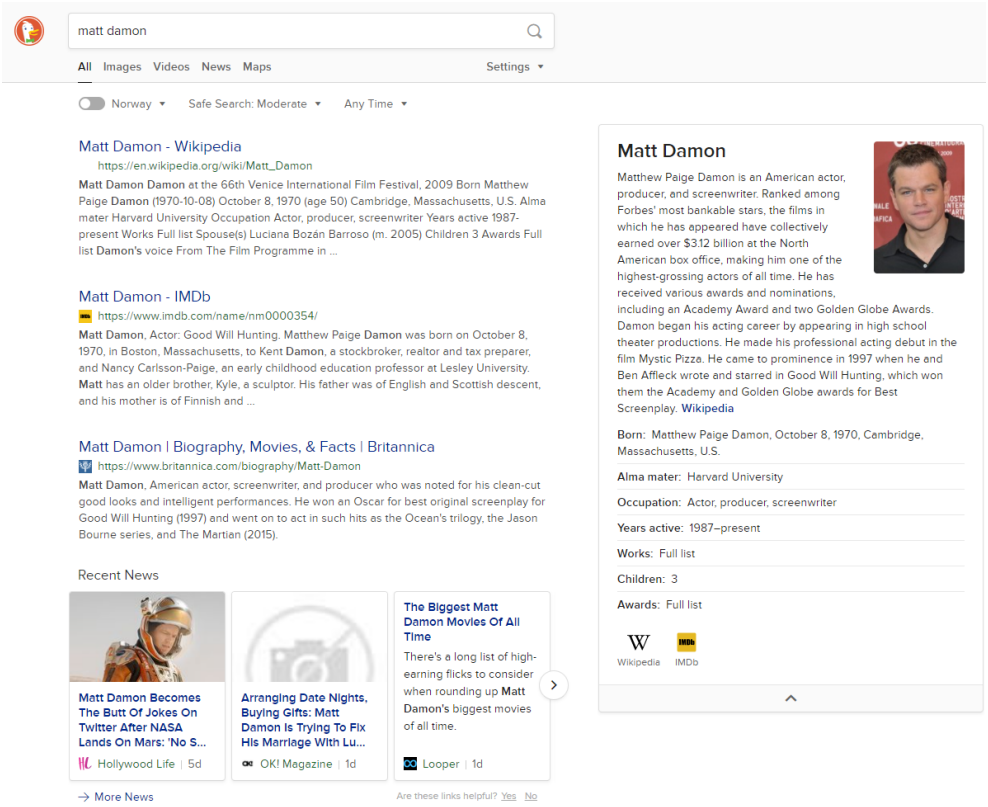


Figure 2.4: Example of a keyword search on DuckDuckGo that is enriched with an entity card

2.3.1 Entity Representation

Having a good entity representation is essential for effective retrieval and ranking of entities. An entity representation is a textual depiction of the entity, used to aid in the retrieval process [17, p. 19]. These can be developed in several ways

depending on the available data. Some entities have ready-to-use representations based on structured data, while others have semi-structured information like an entity's Wikipedia page. Sometimes, however, they are made from purely unstructured documents with no readily made representations. These entities are commonly represented using semantic information stored as RDF triples. Here, semantic properties are represented as RDF predicates. In the context of entities, these are called fields and represent the attributes and relationships of the entity.

Balog [17, p.61-71] presents several approaches to creating entity representations. One way is having each document annotated by the entities they reference, then use the documents to connect terms and entities. Candidate entities like these can be found either by human interaction or through fully automated (entity linking) procedures. On a document level, a representation can be as simple as concentrating the contents of all the documents that mention the given entity. Additionally, you could consider the context in which the entity appears by looking at a set number of adjacent entities. This can reveal a more accurate entity representation because the entities within certain proximity may be more related than separately placed entities in the same document.

Another way to create entity representations, when dealing with predefined entities with fields and relations, is by utilizing predicate folding [17, p. 70-71]. Predicate folding is when several predicates are grouped into predefined categories. This way one entity representation can contain several fields. Depending on the sparsity in fields and data, the fields used in the predicate folding can be changed. Some examples are predicates like names, attributes, types, relations, catch-all fields, or some form of top-predicates(which entails the most used fields). It is common to remove URIs as they are not suited for text-based search.

2.3.2 Entity Retrieval

Entity retrieval is an important field because entities bridge the gap between structured and unstructured data [17, p. 15]. Formally, entity retrieval is the task of retrieving and ranking entities mentioned in a document collection following the relevancy of a query [24]. This is an increasingly important task as it has been estimated that more than 40% of web search queries target entities [2–4]. Retrieval of entities can give direct and specific answers to the user, as well as enrich the search result and contextualize the information. Two important points set entity retrieval apart from traditional document retrieval [25]. Firstly, entities are not always explicitly represented as retrievable units like documents are. Entities consist of mentions and occurrences in the document collection. This creates a need for new "profile" documents for each entity that includes additional information about occurrences in the document collection. Secondly, compared to unstructured documents, in entity retrieval, the underlying structure of the entities can be utilized. This includes entity types, attributes, and relationships to

other entities.

Ad-hoc Entity Retrieval

Semantic search is a broad term concerning a wide range of concepts and challenges. In this thesis, the focus falls on a sub-group of search within semantic data called *ad-hoc entity retrieval*. The goal of ad-hoc entity retrieval is to return a ranked list of entities with respect to each entity's relevance to a given query [17]. These queries are mainly keyword- and natural language queries to make the information accessible to users with an ad-hoc information need and no prior knowledge of the data.

A popular entity representation for ad-hoc entity retrieval is to represent each entity as a semi-structured document of fields with *bag-of-words* (BoW) values [24]. These fields include entity attributes and relationships to other entities. This makes it possible to use and adapt existing approaches from unstructured document retrieval in an entity retrieval setting. For example, retrieving entities with explicit representation was formally proposed by Pound *et al.* [2]. By mapping an object ranking problem in an entity-based context to the already established problem of ad-hoc document retrieval, it is possible to reuse and adapt established theory and existing works. For instance, the baseline for their ranking approach is to consider TF-IDF over entity properties in an RDF graph.

2.4 Preprocessing and Indexing

While traditional IR aims to maximize effectiveness, it encounters some challenges when dealing with a plethora of documents, which can make the search process time-consuming. To combat this, indexing deals with the efficiency of the search and aims to process user queries using minimal resources. This is done by creating a data structure to store and quickly locate data without having to search through every entry in the database. Generally, this leads to a trade-off between faster retrieval and processing of data at the cost of additional writes and storage space. This is done to create and maintain the index when changes or updates are necessary.

2.4.1 Text Operations

Before indexing and later searching through the documents, the contents should be preprocessed to reduce overhead. Not every character or word in a natural language appears as frequently. The distribution of use can be skewed with stopwords such as 'at, the, this, it' being used more frequently than jargon words such as 'holistic, Q.E.D, cache'. The most frequent words supply very little discriminatory power and can therefore be disregarded. By picking the most fitting words for indexing a lot of the overhead can be reduced and the efficiency of the search

increased. This can, however, affect the effectiveness as the meaning of phrases and specific words may no longer be as searchable.

Lexical Analysis

Lexical analysis converts looking at the text as a stream of characters to recognizing it as a stream of words. This is done not only by separating words by spaces, but by handling cases with special characters like digits, hyphens, punctuation, upper and lower case. Numbers by themselves are usually not relevant unless set in a context such as "513B.C" or "COVID-19", or presented in a special format such as a date. Hyphens might be inconsistently used. In this case, splitting by the hyphen might be beneficial so that 'up-to-date' and 'up to date' are treated equally. Other words might benefit from staying hyphenated. The same goes for punctuation and case lettering as they are usually disregarded by being removed or converted, but can be beneficial to keep in some circumstances.

Removal of Stopwords

Most of the frequently used words are stopwords that do not help with the effectiveness of retrieval. If a term occurs in almost every document, it will not significantly reduce the number of potential documents and is therefore not very discriminatory. A word that appears in 80% of the collection is practically useless for retrieval purposes [26, p.226]. Phrases consisting almost solely of stopwords, such as Shakespeare's 'to be or not to be', will however be nearly impossible to find after stopword removal.

Stemming

A document collection can include the same word in different forms such as plural, present, and past tense. Typically, each is a slight variation with a common denominator, namely the *stem*. An example would be the word *wait* seen in table 2.1 where the stem has many suffix variants.

Word variation	**Form**
wait	plain form
wait*s*	third-person singular
wait*ed*	past tense
wait*ed*	past participle
wait*ing*	present participle

Table 2.1: Different versions of the verb 'wait'.

Stemming is the process of removing these suffixes ending up with the word's stem [27]. In this case, all variations of the verb 'wait' will be processed as the word's stem, namely, wait. As a result, every time a variation appears, it will be

counted together with other appearances which will significantly reduce the number of different words to keep track of. Frakes [28, p. 158-165] performed a literature review comparing eight distinct studies on the potential benefit on search performance when using stemming. He concludes that stemming may affect retrieval performance, but that the studies have equivocal results. Because of this, stemming might not be beneficial from a ranking perspective, but can significantly reduce the size of the index.

In the English language, many words share common suffixes such as 's, es, ed, ing'. However, every language is different and comes with a varying set of rules and exceptions that makes stemming more challenging.

2.4.2 Inverted Index

Inverted index is one of the most commonly used data structures in document retrieval systems [29, Chapter 6.5]. It is a word-oriented database index for storing and mapping content to its exact location in the database. When working with graph data, it can be useful to map terms to the Nodes where they occur. If document 1 and 3 contain the term "hello" in position (3, 5, 6, 200) and (9, 10) respectively, a typical inverted index structure would be the following:

```
1 "hello" => [1:<3,5,6,200> , 3:<9,10>]
```

As a result, documents containing query terms can be efficiently retrieved. This drastically reduces the number of processed nodes, which will improve the retrieval speed. This is because we only need to perform the search on a subset of the dataset known to be relevant.

2.4.3 Indexing for RDF Data

The two main approaches for indexing RDF data are a *horizontal index* and a *vertical index* [6, 15]. For a horizontal index, the RDF resources are represented using three fields following the triple structure (subject-predicate-object). This means that all the subjects, predicates, and objects for a resource are stored horizontally in parallel with a set number of fields. Vertical index, on the other hand, creates a new field for each predicate occurring for the resource. Corresponding objects are then matched vertically. Thus, the horizontal position does not play a big role but can be useful to represent properties with multiple values which is a small advantage over the horizontal index. Table 2.2 and 2.3 are examples of a horizontal and vertical index, respectively, for the sample RDF data shown in listing 2.5. This example is taken from [15].

```
@prefix foo: <http://example.org/ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
foo:peter foaf:name "peter mika" .
foo:peter foaf:age "32" .
foo:peter vcard:location "barcelona" .

```

Figure 2.5: Sample RDF data in Turtle format

Field	pos1	pos2	pos3	pos4	pos5
object	peter	mika	32	barcelona	
predicate	foaf:name	foaf:name	foaf:age	vcard:location	
subject	http	example	org	ns	peter

Table 2.2: Horizontal index of the data in Table 2.5. Example taken from [15]

Field	pos1	pos2	pos3	pos4	pos5
foaf:name	peter	mika			
foaf:age	32				
vcard:location	barcelona				

Table 2.3: Vertical index of the data in Table 2.5. Example is taken from [15]

Mika [15] has shown that these indexes can be efficiently applied in a distributed fashion, supporting big data use-cases. Both the horizontal and the vertical indexes are similar in size. The vertical index does, however, not include all the information needed for ranking like subject information which gives access to term frequencies and document sizes. Therefore a horizontal index can either be applied alone or in combination with a vertical index for faster matching.

2.5 Ranking Models

Baeza-Yates and Ribeiro-Neto [26] describe IR models as functions that assign scores to documents with regard to a given query. They explain that it consists of two main tasks, namely a logical framework for the representation of documents and queries, and a ranking function that calculates a rank for each of the documents in consideration to a specific query [26, p. 57]. In this section, the focus will be on the last-mentioned, specifically the computation of a rank.

2.5.1 TF-IDF

One of the simplest approaches for scoring relevancy of potential results is a method called TF-IDF which stands for *Term Frequency - Inverse Document Fre-*

quency. This is done by statistically evaluating how relevant a term is to a document in a collection of documents.

Term Frequency

If a document frequently mentions a query term, it must be strongly related to that term and should receive a higher score. This is called the *term frequency* (TF) and is denoted $\mathrm{tf}_{t,d}$, where t refers to a term and d refers to a document. When calculating the TF, each document is regarded as a *bag-of-words*. This entails that the order of the terms are ignored and the only information of interest is the number of occurrences [1, p.117]. This means that 'The cat is chasing the dog' is seen as equivalent to 'The dog is chasing the cat' even if the context is a little different, the content is fairly similar.

Inverse Document Frequency

When considering TF, one challenge appears; all the matching terms are seen as equally important when calculating a document's relevance to a query. For instance, if you look at a domain-specific collection of documents about movies, a lot of reoccurring words, like 'director' and 'actor', will appear in close to every document. The deciding factor will then solely depend on the frequency of the term in one document. This means that a movie with a greater cast of actors will receive a higher relevance score than other movies. To combat this, the relevancy of terms frequently mentioned in the entire collection of documents has to be scaled down. If the term appears in every document, it is not very discriminatory. To scale the weights of terms, *inverse document frequency* (IDF) is used. It is defined as:

$$\mathrm{idf}_t = log \frac{N}{\mathrm{df}_t}. \tag{2.1}$$

where N is the total number of documents in a collection and the TF in a document is denoted df_t. This makes the score of a rare term high, while a common term lower.

Combining TF and IDF

If the two concepts are merged, a combined weight for each term in each document can be produced by using:

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t. \tag{2.2}$$

This means that the score will be highest when a term, t, occurs a great number of times in few documents leading to a high discrimination factor. It will be lower when t occurs fewer times in a document or is prevalent in several documents, and the lowest when it is present in practically every document. To sum up all the TF-IDF weights for each term in document d, this formula can be used:

$$Score(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}. \tag{2.3}$$

Variants of TF-IDF

There are several variants of TF-IDF and its components. The ones discussed by Salton and Buckley [30] and mentioned by Written, Moffat, and Bell [31, p. 183-185], which are presented in *Modern Information Retrieval* [26, p. 73-74] will be in focus.

Weighting scheme	**TF weight**
Binary	{0,1}
Raw frequency	$f_{i,j}$
Log normalization	$1 + log f_{i,j}$
Double normalization 0.5	$0.5 + 0.5 \frac{f_{i,j}}{max_i f_{i,j}}$
Double normalization K	$K + (1-K) \frac{f_{i,j}}{max_i f_{i,j}}$

Table 2.4: TF variants

There are mainly five different versions of TF as depicted in table 2.4. The binary version assigns 0 or 1 score to query terms in documents based on occurrence, where 1 is present and 0 is not present. Raw frequency is the most common alternative and is used to describe TF in section 2.5.1. Log normalization uses logarithm to decrease the score of frequent terms. Double normalization 0.5 normalizes weight by maximum frequency (in document or query) and normalizes the weight to end up between 0.5 and 1. Lastly, double normalization K is a generalization where changing K can reduce or increase the frequency's influence on TF.

Weighting scheme	**IDF weight**
Unary	1
Inverse frequency	$log \frac{N}{n_i}$
Inverse frequency smooth	$log(1 + \frac{N}{n_i})$
Inverse frequency max	$log(1 + \frac{max_i n_i}{n_i})$
Probabilistic inverse frequency	$log \frac{N - n_i}{n_i}$

Table 2.5: IDF variants

For IDF, there are five variants presented in table 2.5. Unary assigns 1 to all terms so that IDF has no impact on the score. Inverse frequency is the standard version explained in section 2.5.1. Inverse frequency smooth avoids extreme or odd weights by summing 1 to the fraction. Inverse frequency max computes the weight while taking the largest document frequency into consideration. This is done in place of the number of documents in the collection. Lastly, probabilistic inverse

frequency is a variant to the classic inverse frequency that instead of only considering the number of documents in the collection it subtracts the frequency of the i-th document. Combining versions of TF and IDF will yield different versions of TF-IDF as shown in table 2.6.

Weighting scheme	Document term weight	Query term weight
1	$f_i, j \times log\frac{N}{n_i}$	$(0.5 + 0.5\frac{f_i,q}{max_i f_i,q}) \times log\frac{N}{n_i}$
2	$1 + logf_i, j$	$log(1 + \frac{N}{n_i})$
3	$(1 + logf_i, j) \times log(\frac{N}{n_i})$	$(1 + logf_i, q) \times log\frac{N}{n_i}$

Table 2.6: TF-IDF variants

2.5.2 Boolean Model

The Boolean model is a simplistic model based on set theory and Boolean algebra. Similarly to the binary version of TF-IDF, it only considers the occurrences of terms. If a term is present in a document it is represented by a 1, and when absent it is represented by a 0. Each document in a collection has a binary-term document *incidence matrix*, where terms are indexed units(often words). If there for instance is a document with information about characters in movies and which movies these characters appeared in, the matrix would look something like this:

	The Fellowship of the Ring	The Two Towers	The Return of the King
Legolas	1	1	1
Theoden	0	1	1
Lurtz	1	0	0

Table 2.7: Incidence matrix of Lord of the Rings trilogy

In the Boolean retrieval model, all queries are formulated as Boolean expressions with operators such as *AND, OR*, and *NOT*. An example would be if a document contains **Legolas AND Theoden NOT Lurtz**. Which in the case of table 2.7 would result in 111 AND 011 AND 011 = 011. The answer is then that Legolas and Theoden but not Lurtz appear in 'The Two Towers' and 'The Return of the King'.

This model is straightforward and easy to understand, and with the right discriminatory terms, it is simple to significantly reduce the number of relevant documents. The downsides are that the formulation of Boolean queries might be difficult for end-users, and the retrieved documents are not ranked. This results in every document containing the term(s) being deemed just as relevant, making it difficult to distinguish them further.

2.5.3 Vector Space Model

Another way to look at documents and queries is as vectors in a common vector space. The vector of a document is denoted by $\vec{V}(d)$ where each term is represented by one component in the vector. How are then two vectors compared? If you simply look at the vector difference, documents with similar content may be significantly different because of the difference in length. Manning says that "the relative distributions of terms may be identical in the two documents, but the absolute term frequencies of one may be far larger" [1, p.121]. One way to compensate for document length is to utilize *cosine similarity* for two document vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ as:

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \tag{2.4}$$

where the $\vec{V}(d_1) \cdot \vec{V}(d_2)$ denotes a *dot product* and $|\vec{V}(d_1)||\vec{V}(d_2)|$ is the product of the vectors *Euclidean length*. The denominator will here be responsible for the length-normalization mentioned earlier. Here $\vec{V}(d_1)$ and $\vec{V}(d_2)$ are set to unit vectors $\vec{v}(d_1) = \frac{\vec{V}(d_1)}{|\vec{V}(d_1)|}$ and $\vec{v}(d_2) = \frac{\vec{V}(d_2)}{|\vec{V}(d_2)|}$. This means equation (2.4) can be reformatted as

$$sim(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \tag{2.5}$$

with this similarity in mind, one can use a document d_i in a collection to find the documents that are the most similar. But if the aim is to compare the documents to a query, how is that done? Similarly to a document, the query can be considered as a unit vector making it possible to assign each document d a score with dot product:

$$\vec{v}(q) \cdot \vec{v}(d). \tag{2.6}$$

By considering a query as a BoW it can be treated as a small document. Thus (2.4) can be changed to compare a query q and a document d instead of two documents:

$$sim(q, d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}. \tag{2.7}$$

Lucenes approach to TF-IDF

Apache Lucene is a free and open-source information retrieval library[2]. It provides powerful and high-performance indexing and searching features written in Java. Apache Lucene's approach to TF-IDF is to combine the boolean model (BM) and the vector space model (VSM). First, the documents are 'approved' by the BM and then scored by the VSM[3]. This will, in this thesis, be referred to as *Lucene Fulltext*

[2]Apache Lucene's website: `https://lucene.apache.org/`

[3]Apache Lucene's documentation on Class TFIDFSimilarity: `https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`

Search.

The VSM score (equation 2.7) is redefined as the following:

$$\begin{aligned}\text{score(q,d)=coord-factor(q,d)} \cdot \text{query-boost(q)} \cdot \\ \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)|} \cdot \text{doc-len-norm(d)} \cdot \text{doc-boost(d)}\end{aligned} \tag{2.8}$$

Doc-len-norm(d) is an adaption to the normalization of document length using unit vectors and normalizes a vector to equal or larger than the unit vector. *Doc-boost(d)* and *query-boost(q)* are present so specific documents and query terms can be weighted accordingly. This is done by multiplying each document or query score with the boost value.

Since documents don't necessarily contain all query terms, a multi-term query *Coord-factor(q,d)* is used to further reward documents that match several query terms. The larger the coordination factor, the higher amount of query terms are matched.

2.5.4 BM25

The *BM25 weighting scheme* is a probabilistic ranking model sensitive to term frequency and document lengths [1, p. 232]. The calculated score estimates the relevance of documents for a given query.

Equation 2.9 defines the score of a document D given a query Q that contains the terms $t_1, t_2, ..., t_n$.

$$\text{score}(D,Q) = \sum_{i=1}^{n} idf(t_i) \cdot \frac{tf(t_i,D) \cdot (k_1+1)}{tf(t_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \tag{2.9}$$

Here, $idf(t_i)$ is the inverse term frequency of the given term calculated as previously seen in Equation 2.1, $tf(t_i,D)$ is the TF of the given term in document D, $|D|$ is the document length, avgdl is the average document length in the collection, and k_1 and b are free parameters.

$$\frac{tf(t_i,D)}{tf(t_i,D) + k_1} \tag{2.10}$$

Equation 2.10 shows the saturation function [8]. This function describes the non-linear increase of the probability that a document is relevant as the TF increases. It is important because, as previously mentioned, the probability of relevance increases along with the TF. This is, however, not a linear increase and k_1 allows control over the non-linear growing TF function. A large k_1 value corresponds to raw TF, and the smaller the value is, the less each term occurrence counts until the value is 0 which corresponds to a binary model. b lets us control the scaling

of term weight by document length. Reasonable parameter values deduced by experiments have shown $k_1 \in [1.2, 2.0]$ and $b = 0.75$ [1, p. 233]. It should be noted that even if these are common default values, they can yield sub-optimal retrieval performance [17, p.76].

2.5.5 BM25F

BM25F is an extension of the BM25 ranking model that incorporates field weights and per-field length normalization to better suit the retrieval of structured documents. To calculate the BM25F score, the steps described by Craswell in [8] can be followed, starting by calculating a normalized TF for each field c.

$$tf_c(t, D) = \frac{occurrence_c(t, D_c)}{1 + b_c(\frac{|D_c|}{avgdl_c} - 1)} \tag{2.11}$$

$occurrence_c(t, d)$ is the occurrences of term t in the given document field D_c, $|D_c|$ is the field length, $avgdl_c$ is the average field length for the given field, and finally b_c is a field-dependent variant to the b parameter in BM25. With the field-dependent normalized term frequencies, the BM25 saturation function 2.10 can be used to calculate BM25F. Equation 2.12 defines the BM25F score of a structured document D containing the fields $c_1, c_2, ..., c_n$ given a query Q.

$$BM25F(D, Q) = \sum_{i=1}^{n} idf(t_i) \cdot \frac{tf(t_i, D)}{tf(t_i, D) + k_1} \tag{2.12}$$

$$tf(t_i, D) = \sum_{c=1}^{n} \omega_c \cdot tf_c(t_i, D_c)$$

Here, $tf_c(t_i, D_c)$ is the field term frequency function described in equation 2.11, and ω_c is the boost factor describing the importance of each field.

2.5.6 Comparing Saturation Functions

Most ranking models have a method to deal with the non-linear increase of relevance as term frequencies increase. Some models use either a simple logarithmic or a square-root function, while other models, like BM25, use a tunable parameter approach. This is commonly called a saturation function. When combining scores from different fields in structured data, the saturation function used by the given model has to be taken into account. Converting models designed for unstructured documents to be used on structured documents may lead to some undesired characteristics [32]. The Lucene library is a good example of this. As described in chapter 2.5.3, Lucene utilizes a combination of the vector space model and the Boolean model. When using Lucene to retrieve structured documents, the score is computed by linearly combining the scores of each field. Note that these formulas are not Lucene's exact implementation, but a simplification to illustrate important points.

$$\text{score}(Q, D) = \sum_{c=1}^{n} \text{score}(Q, D_c) \tag{2.13}$$

$$\text{score}(Q, D_c) = \sum_{i=1}^{n} tf_c(t_i, D_c) \cdot idf(t_i) \cdot \omega_c$$

$$tf_c(t, D_c) = \sqrt{freq(t)}$$

Here the saturation function $\sqrt{freq(t)}$ is applied before the boost factor ω_c. As described by Pérez-Agüera *et al.* [11], this is problematic since the linear combination of the field scores, $\text{score}(Q, D_c)$, breaks the non-linear saturation effect when the boost factor is applied after the saturation function. In other words, there is no non-linear saturation for the terms across document fields when combining the field scores. As a result, documents matching a single query term over several fields score much higher than documents scoring several query terms in a single document field. When a single query term appears across several fields, it is not necessarily more relevant because of redundancy. For instance, a term in a name field can be expected to appear several times in an alias field, and maybe even in a description. Hence, there is more interest in matching more query terms despite not getting hits across document fields. BM25F applies the boost factor after the saturation function, and thus, combining the field scores does not cancel the saturation effect. As a result, we can expect BM25F to perform better when dealing with some structured data.

2.6 Evaluation of Ranking Models

To evaluate an IR system you have to assess how well it fulfills the information need of users. This can be difficult as identical result sets may be interpreted differently by users. A common way to approach evaluation is by comparing results produced by the IR system with results chosen to be relevant by humans. This thesis will mainly utilize evaluation measure definitions described by Baeza-Yates [26].

2.6.1 Precision and Recall

Precision and recall are some of the most widely used methods to evaluate the retrieval qualities of an IR system. Consider a query requesting some information that yields an ideal set of relevant documents R, and an answer-set of documents given by the retrieval algorithm A. If $|R|$ denotes the number of relevant documents and $|A|$ the number of retrieved documents, then $|R \cap A|$ is the intersection of these documents. *Precision* can then be defined as the fraction of the retrieved documents (A) which are relevant:

$$Precision = \frac{|R \cap A|}{|A|} \tag{2.14}$$

Recall can be defined as the fraction of relevant documents (R) that was retrieved:

$$Recall = \frac{|R \cap A|}{|R|} \tag{2.15}$$

2.6.2 F-Measure

Both precision and recall are important and can be interesting to look at individually. However, it is useful to get a single value score that combines both of these measures. To get a score that balances the concerns of both precision and recall, F-Measure can be calculated as:

$$F = \frac{2 \cdot Precision \cdot Recall}{(Precision + Recall)} \tag{2.16}$$

2.6.3 Mean Average Precision

To get a calculation of the user impression of the performance, it is normal to calculate the precision when a certain amount of documents has been retrieved (normally p@5, p@10, p@20). To get a single value score of all the precision values the *average precision* can be calculated for the set of relevant documents to the query (R):

$$AvgP = \frac{\sum_{k=1}^{|R|} P(R[k])}{|R|} \tag{2.17}$$

Here, $R[k]$ refers to the k-th document in R, and $P(R[k])$ is the precision when the document occurs in the ranking. If the document never occurs in the ranking, the precision for that document is zero.

If the goal is to get a single value score for a set of queries, Q, the mean average precision (MAP) can be used:

$$MAP = \frac{\sum_{q=1}^{|Q|} AvgP(q)}{|Q|} \tag{2.18}$$

2.6.4 Discounted Cumulative Gain

Precision and recall only grant binary relevance assessment. This can lead to high scores even though highly relevant documents are ranked low. Discounted cumulative gain (DCG) solves this by penalizing highly relevant documents appearing lower in a search result [33]. The use of DCG in this thesis is based on the works of Järvelin and Kekäläinen [33, 34]. The DCG accumulated at the n-th position of the ranking is defined as:

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{log_2(i+1)} \tag{2.19}$$

where rel_i is the relevance score for the document at the i-th position.

The normalized discounted cumulative gain (NDCG) is the fraction of the DCG for a search result compared to the ideal discounted cumulative gain (IDCG) for the same result.

$$NDCG_n = \frac{DCG_n}{IDCG_n} \tag{2.20}$$

The IDCG is the highest DCG score possible meaning a perfectly ranked search result set gives an NDCG score of one.

2.6.5 Cohen's Kappa Coefficient

When performing user evaluation testing, it is important to consider the degree of agreement between the testers. This is commonly done using a simple percentage agreement calculation. However, this does not take into account the agreement happening by chance. The kappa coefficient, first introduced by J. Cohen [35], solves this by addressing the hypothetical probability of agreement. The agreement of I testers who each evaluated N items can then be calculated as:

$$\kappa = 1 - \frac{1 - p_o}{1 - p_e} \tag{2.21}$$

where p_o is the observed proportionate agreement

$$p_o = \frac{Number\ in\ Agreement}{Total}$$

and p_e is the hypothetical probability of agreement

$$p_e = \frac{1}{N^2} \sum_k \prod_{i=1}^{I} n_{ki}$$

Here, k are the categories and n_{ki} is the amount of times rater i predicted category k.

Fleiss Kappa Coefficient

While Cohen's Kappa Coefficient only concerns two parties at a time, Fleiss [36] takes into consideration when there is a constant number of raters, larger than two. This is done by first calculating the proportion of assignments to a given category, then calculating to which extent rater pairs agree on a certain subject. Lastly, the mean of the former calculations is computed.

Evaluation of the Kappa Coefficient

How do you assess the scores given by the Kappa Coefficient? Viera and Garrett [37] propose that a scale of evaluation such as shown in table 2.8 is used. This is supposed to help visualize the Kappa values beyond just numbers.

Kappa	Agreement
<0	Less than chance agreement
0.01-0.20	Slight agreement
0.21-0.40	Fair agreement
0.41-0.60	Moderate agreement
0.61-0.80	Substantial agreement
0.81-0.99	Almost perfect agreement

Table 2.8: Visualization of Kappa Coefficient

2.7 Previously Explored Approaches to Keyword Search in Graphs

There are several ways of approaching keyword search on graph data. Each paper has a different approach and ultimately focuses on retrieval and results to be fetched.

The authors of [5], Elbassuoni, and Blanco, conducted research on challenges closely related to this thesis, where they proposed a retrieval model for keyword queries over RDF graphs. Their model regards each RDF triple as a separate document, and associates each document with a set of keywords. By adopting a backtracking algorithm and a statistical language model to their retrieval model, they retrieve and rank top-k sub-graphs matching the query keywords. A language model is a probability distribution over terms based on how likely specific terms are to be observed in a given language [38]. They specifically aim to retrieve subgraphs as they claim it "[...] can be particularly beneficial for both result retrieval and result representation" [5, p. 237] as they believe that "[...] the graph representation provides more concise answers to the users' information need than a set of entities" [5, p. 238]. However, in this thesis, the result representation will not be regarded on the same level of importance as relevance assessments. As a result, looking at keyword queries over RDF graphs as an entity-oriented search problem provides a more practical foundation for comparison between the different ranking models. Thus, the work on the backtracking algorithm to retrieve sub-graphs would not be applicable to the research in this thesis but would be an optional supplement to any of the ranking models compared in chapter 6 if retrieving sub-graphs is regarded as more desirable.

Tran *et al.* [14] took a vastly different approach to keyword search over RDF data. They propose an algorithm that computes top-k structured queries from a keyword query, instead of the traditional approach of directly computing answers. This adds an additional intermediate step where the user has to choose the most appropriate structured query for their information need. In other words, it is assumed that a keyword query is an implicit representation of a structured query. This solves the

challenge of needing prior knowledge of the data and its structure when working with structured queries. It would also ease the challenge of writing queries and require less technical expertise from the users as they argue "Structured queries can serve as descriptions of the answers and can also be refined more precisely than using keywords" [14, p. 416]. Some expertise is still required, and the two-step process would not be beneficial for a large-scale, user-friendly application. Since their approach pivots towards the use of *keyword++* queries, and this thesis focuses on the comparison of ranking models behavior, this would not be a part of the implementation.

Tonon *et al.* [9] chose to build upon Ad-hoc entity retrieval by proposing a hybrid approach combining structured search techniques and traditional IR used for *ad-hoc* document retrieval. Their architecture uses an inverted index to retrieve intermediate top-k results in addition to graph traversals and neighborhood queries to finalize the graph-enriched results. Their results show that "[...] the use of structured search on top of standard IR approaches can lead to significantly better results (up to 25% improvement over the BM25 baseline in terms of Mean Average Precision)." [9, p. 134]. Additionally, they found that approaches based on structured inverted index with BM25F significantly outperformed approaches based on the same index with BM25 in terms of MAP, NDCG, and early precision (P@10). It would be intriguing to see if these results still hold for smaller, domain-specific knowledge graphs. Additionally, it would be interesting to compare these results with other structured ranking models like Lucene.

The authors of [12] explore the use of both structure and content when ranking results. Here, the ranking models for web objects (in this case RDF representation of entities) are developed based on a traditional language model. Throughout the paper, they assume the entire process deals with entities, including the results given to the user. They also assume a document is connected to each entity, something that is not present in the research of this thesis. They developed and compared three models for web object retrieval: unstructured object retrieval model, structured object retrieval model, and a hybrid model including features from both unstructured and structured. Their research shows that in the context of searching for academic papers, the hybrid model consistently performed better than the other models.

Castells, Fernández and Vallet [13] adapted the vector space model for ontology-based representation. They argued that "documents hold a value of their own and are not equivalent to the sum of their pieces no matter how well formalized and interlinked" and therefore wanted to expand on already existing models. They worked with domain-specific KBs similarly to the research in this thesis but assumed some consistent data structure with the presence of three root ontology classes. Instead of just relying on keyword search, the group decided to supplement the search with the use of ontologies. The inclusion of keyword-based search

results provided more robustness where pure ontology-based search results were lacking. On the other hand, the cases where pure ontology-based search significantly outperformed keyword-based search deteriorated. The inclusion of ontologies could therefore be seen as a trade-off to make the ranking model more consistent while missing out on some previously outstanding results.

Due to the Covid-19 pandemic, the TREC-COVID challenge [39] was initialized in 2020. A vast number of scientific papers regarding the virus were published in a short amount of time, making it challenging to keep up to date. As a result, the challenge of finding relevant information and keeping evidence from getting buried became the focus of this initiative. A. Esteva *et al.* [10] handled this challenge by generating a bipartite graph of document paragraphs and citations, making it a semantic search problem. The presented semantic search engine, CO-Search, is split into two main parts: A retriever and a ranker. The retriever approximates the nearest neighbors and returns a set of paragraphs along with each paragraph's cosine similarity. The paragraphs are ranked using a combination of TF-IDF and BM25 to retrieve documents. The ranker then combines this score of the retrieved documents with a *Question Answering Model* (QA model) and an *Abstractive Summarization* which results in a ranked collection of documents given a query. This approach resulted in the system achieving #1 taking all evaluation metrics into consideration. The method was, however, very specifically tailored to the challenge, making it difficult to generalize. This shows that good performance often requires customization towards the given problem. A general-purpose model is more conveniently applicable to various use cases. This thesis will focus on the latter.

In this system, TF-IDF and BM25 operate on the document level meaning these models do not take advantage of the underlying graph structure when ranking. However, it uses nearest-neighbor approximation and the QA model. It would be interesting to see how well this system performs for data collections containing semantic data, and compare these results when swapping TF-IDF and BM25 with fielded variants.

2.7.1 Explored Evaluation Methods

Evaluating IR systems is a challenging field, but over the years, two main approaches have been developed; a user-based evaluation approach, and an automatic evaluation approach. In the case of the study [5], Elbassuoni and Blanco evaluated their retrieval model by taking a user-based approach. This was done by creating their own query benchmark and relevance assessments due to the lack of an appropriate query benchmark for keyword search over RDF data. This means they asked people to judge the IR system's performance based on a predefined scale of relevance. This thesis employs a similar evaluation approach for its research which is explained in further detail in chapter 5.

Pérez-Agüera *et al.* [11] takes the automatic evaluation approach, using the Text REtrieval Conference (TREC) evaluation software[4] looking at performance measures like *MAP*, *GMAP*, and *R-Precision*. In this study, they compared both standard and fielded variants of Lucene search with BM25 on semantic data. They proposed a semantic search evaluation framework based on the evaluation of XML retrieval (INEX[5]) (INEX have since come to an end). For a collection of RDF documents collected from DBpedia, their research shows that both BM25 and BM25F perform better than Lucene and LuceneF. However, they conclude that BM25F does not make any significant improvement compared to BM25. This may be due to the nature of the structure in the dataset, where some fields contained a great amount of text, while others contained less and had fewer relevant keywords. As a result, it is interesting to see if these results hold for data sets where all fields contain a small amount of text and all fields can contain relevant keywords. This will be explored further in chapter 6.

The TREC-COVID challenge uses several different metrics for evaluation in the competition. These include NDCG, Precision with N documents(P@N), MAP, and Binary preference.

2.7.2 Comparing Results

Blanco and Vigna conducted a study [6] on the efficiency of different index techniques on a large RDF data set in addition to the effectiveness of BM25F ranking vs BM25. Just like Tonon *et al.* [9], but in contrast to Pérez-Agüera *et al.* [11], they found that BM25F significantly outperformed BM25. They performed their testing on the Billion Triples Challenge[6] from 2009. Due to these contradicting results, more research is needed to conclude when BM25F could be more beneficial than BM25.

During The Semantic Search Challenge in 2010, which concerned keyword queries over RDF data, [7] was the best performing approach. They ranked the entities using BM25F with tailored indexing, manual classifications (important, unimportant, and neutral), and property and site class weights. This could indicate that the BM25F model thrives on customization in accordance with the data to be queried.

[4] `https://trec.nist.gov/trec_eval/`
[5] `https://inex.mmci.uni-saarland.de/`
[6] `https://zenodo.org/record/2634588`

Chapter 3

Concepts and Methods

The objective of this chapter is to give an overview of the research questions and the methods used to answer them. The concepts and ideas that lead to the prototype implemented for the purpose of this research will be the main focus.

As mentioned in the introduction, our thesis focuses on 4 research questions;
RQ1: How do IR ranking models perform for users searching in semantic data?

> **RQ.1:** What are the strengths and weaknesses of different IR ranking models?
> **RQ1.2:** How do the ranking models behave in an entity retrieval setting?
> **RQ1.3:** How do fielded ranking models' performances compare?

RQ2: What methods are suitable for evaluating entity retrieval?

To answer these questions we need at least three aspects:

- At least one semantic knowledge base containing entities
- A platform to index and query the knowledge base using different IR ranking models
- An evaluation measure to rate the performance of ranking models and compare them to each other

3.1 Domain-Specific Semantic Knowledge Base

A domain-specific semantic knowledge base is a knowledge base containing semantic data within a specific field or theme. Examples of this are library knowledge bases containing books and authors, or knowledge bases with hospital medical records. This is in contrast to a semantic knowledge base with no specific domain, containing all kinds of data. The biggest difference between the two is the consistency in the data structure. In a domain-specific knowledge graph, the nodes will to a large degree share the same properties and relations to each other. This is desirable during indexing and ranking because it is easier to take advant-

age of the underlying structure of the graph to a greater extent.

In regards to RQ1.2, the focus falls on entity retrieval. This is because entities bridge the gap between structured and unstructured data, making the IR models traditionally used for unstructured document retrieval more relevant and appropriate to use on semantic data. Additionally, as mentioned in section 2.3, more than 40%, with some measures up to 70%, of web queries target entities. As the aim is to assess how IR ranking models behave with semantic data, it is interesting to explore how well they take advantage of the underlying structure and how they can be modified to increase performance. With this in mind, a domain-specific KB consisting of entities was a natural choice to evaluate the ranking models' behavior.

Representation of Entities

A central part of entity-oriented search is the creation of entity representations with techniques such as entity detection and entity linking [17, 25]. This includes creating relations with other entities as well as important fields. In this research, two knowledge bases were built by fetching RDF data from Wikidata's SPARQL endpoint. This was done by querying the endpoint constructing the knowledge base as desired. Correspondingly, a disease domain knowledge graph and a movie domain knowledge graph containing entities were constructed. The entities are explicitly represented as separate nodes with fields in a graph database.

As mentioned in section 2.3.1 there are several ways to create entity descriptions. Since this project works with already structured RDF data fetched from Wikidata, property folding was used. Even with some sparsity in the datasets, most entities had values in each used field. On average, each field consisted of few terms, consequently making each term potentially highly impactful when using traditional IR ranking models. The disease knowledge graph contains fewer entities compared to the movie knowledge graph. However, it contains more fields, and the terms are more unique and discriminatory, thus containing more semantic information. This difference aids with answering RQ1 by analyzing and comparing the performances in different environments. Overall, text representation of two domain-specific knowledge graphs containing entities with different characteristics was a good foundation to answer the research questions.

3.2 Indexing and Ranking Models

A requisite to fully answer the research questions is a platform that indexes knowledge graphs to support IR ranking models. Preferably, this platform should be flexible in the type of data it can index, and which ranking models it supports. This is because testing was to be performed on different KGs and use a wide range of ranking models. Additionally, this allows for easy modification of ex-

isting ranking models with alternative weighting schemes or tweaking of the free parameters such as k_1 and b in BM25. It would also be beneficial to have the ability to add new ranking models to expand upon the research in this thesis. As there are no preexisting platforms that fulfill all these demands, a plug-in for an existing graph database platform included in a functional prototype was developed for the purpose of this research. The proposed prototype also consists of a working web application with a search engine to search through the datasets and a survey for collecting relevance assessments. A simplified overview of the prototype architecture is showcased in figure 3.1. Chapter 4 takes a deeper dive into the implementation and architecture.

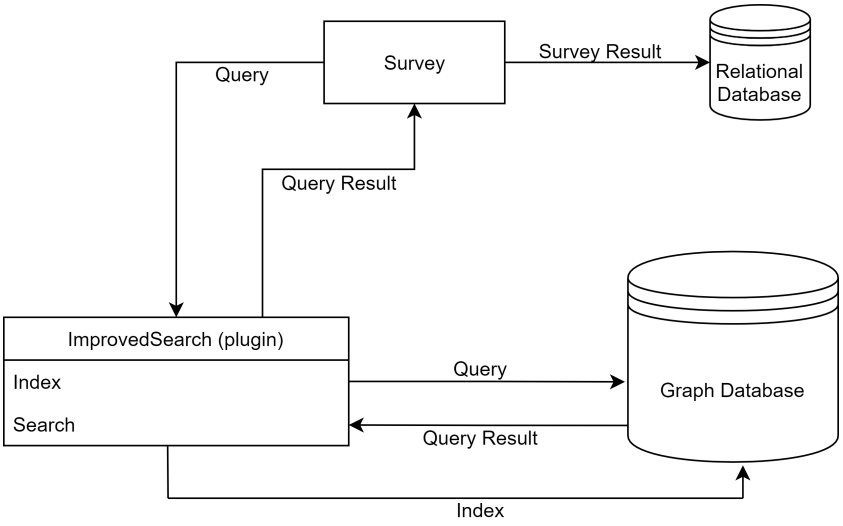


Figure 3.1: Simplified overview of the system architecture

In section 2.5 several ranking models were introduced. These are the models explored throughout this thesis. To answer RQ1 and its sub-question RQ1.1, a review of related research papers and written theory was conducted, gathering the preexisting information. This way there was grounds to compare the already recognized strengths and weaknesses of each model, in addition to the findings in this thesis related to the domain-specific user survey.

These are the following ranking models chosen to be evaluated through web survey results, and additional ranking models for theoretical investigation:

Chosen ranking models for the survey:

- Lucene Fulltext Search
- BM25
- BM25F

Additional ranking models:

- Vector space model
- BM25FF

The main focus was on the vector space model and BM25 because they are widely accepted and popular ranking models used for unstructured document retrieval. Furthermore, other organizations and previous research have explored modifications of these models for information retrieval in semantic data. An example of this is the Apache Lucene library's development and inclusion of its fulltext search which ranks both unstructured and structured documents. Additionally, research such as [6, 8–11] uses either BM25 or its fielded variant, BM25F, for information retrieval in semantic data. This lays a solid foundation to build upon previous research, and compare the strengths and weaknesses of the IR ranking models. It allows for a substantial evaluation of the difference in the behavior of the fielded variants compared to the models developed for unstructured data.

Even though the Apache Lucene library includes all the chosen models, they were all implemented from scratch except Lucene Fulltext Search. This was done because the chosen knowledge base platform supported Lucene Fulltext Search, but no other functions in the library. It was therefore a great learning opportunity to implement the other models from the ground up, getting familiar with each aspect of the ranking models. It built a thorough understanding of how the models calculate relevancy scores and their behavior, which was important in order to correctly interpret the research results. This also opened up the opportunity to develop and test new fielded ranking model variants. With RQ1.3 in mind, a new model was developed for the purpose of this research based on the already known ranking model BM25F. This model is named *BM25FF* where the last F stands for 'fielded' again. This is to indicate it does not calculate IDF-score globally for all fields but does a calculation per unique field type. This is further discussed in section 4.3.2. This model further explores the impact of taking fields into account when ranking.

When working with BM25 and its fielded variants, appropriate k_1 and b values needs to be chosen. Section 2.5.4 mentioned that reasonable parameter values deduced by experiments are $k_1 \in [1.2, 2.0]$ and $b = 0.75$ [1, p. 233], but these values can yield sub-optimal retrieval performance. He and Ounis [40, 41] describe the importance of these parameters and the steps to tune them for optimal performance. However, to stay within the scope of the project, and test the generality of the models, the common parameter values were used for all datasets. This entailed the use of $b = 0.75$ for BM25, $b_c = 0.75$ for all fields for BM25F and BM25FF, and $k_1 = 1.2$.

Fielded ranking models can choose a separate weight for each field. As with the BM25 parameters, instead of tuning each field weight to achieve the best performance for each model, generality was chosen. Thus, the weight parameter was set $\omega_c = 1$ for all fields. By setting all free parameters and field weights to general values all models had a common ground for all datasets. Even though this could give sub-optimal performances, the research primarily focuses on comparing the models, and thus giving each model a fair basis for comparison was deemed more

important.

3.3 Evaluation

With a semantic knowledge base containing entities and a platform to index and query the data established, there was a need for a method to evaluate and compare the ranking models. The intention was to have some quantitative basis to compare how well each ranking model performed with regards to a user's information need. As an exploratory method to evaluate the ranking models, a web application was developed to gather data about users' perceived relevancy of ranked entities. This was done especially with RQ2 in mind as there are no standardized test sets available for the chosen datasets. With the survey data from the web application, evaluation metrics were applied in several ways to fully answer the research questions.

The chosen methods to evaluate the ranking models and the validity of the results were:

- Average Normalized Discounted Cumulative Gain
    - Top 10 results
    - Top 5 results
- Mean Average Normalized Discounted Cumulative Gain
    - Top 10 results
    - Top 5 results
- Kappa coefficient

When working with data from the chosen knowledge graph, automated evaluation measures like MAP and F-measure were not ideal due to the lack of preexisting test collections. To combat this, relevance assessments were gathered by performing an experimental user study with two domain-specific datasets. Here, the testers were asked to score each result from a wide range of queries ranked by different ranking models.

A score of the users' perceived performance of the ranking models can be achieved by calculating the NDCG value at different top-k result sets. This method takes the placement of each relevant result into account and compares the ranking models' performance with an ideal result set. Compared to most traditional evaluations, NDCG allows for a spectrum of relevance instead of the common binary view of relevant or not relevant. More specifically, NDCG involves a discount function over the rank while many other measures uniformly weigh all positions [42, p.2].

The advantage of this model, even if it requires an ideal result set, is that it allows for convenient measurements of how well each ranking model performs compared

to each other. The NDCG was calculated for each ranking model's result to each query. Having scores for each query aids with answering RQ1 by looking at how each ranking model handles different types of queries. Additionally, the average NDCG was calculated for each ranking model in each dataset as well as both datasets combined. This gives an overall performance score for each ranking model. To assure the results are outcomes of intention and not chance, the Kappa coefficient was used. By calculating this for each dataset a degree of the agreement between the test subject can be highlighted.

More details on how the user study was conducted are further discussed in section 4.4 and more on the ranking models' evaluation can be found in chapter 5.

Chapter 4

Implementation and Architecture

The objective of this chapter is to describe the implementation and architecture of the software system developed in this research from both a theoretical perspective as well as a development perspective. From a development perspective, software consists of libraries, plugins, or subsystems packed in small chunks. This chapter will describe how these small chunks are organized, how they are mapped to files, how they are built and managed, and give reasoning for design and implementation decisions made during development.

The proposed system consists of three main components as listed below.

- Two Neo4j databases storing different subsections of Wikidata.
- Neo4j plugin responsible for indexing the data as well as retrieve and rank relevant data based on user queries.
- Web application where end-users can query data sets using different ranking models from the plugin, in addition, to serve as a survey to gather research data.

An overview of the overall system architecture is illustrated in figure 4.1. The Database servers consist of the Neo4j databases and the ImprovedSearch plugin developed in this research. Finally, the Flask server includes the web application used for querying the data and gathering test evaluations.

4.1 Wikidata

Wikidata is a free collaborative KB that over the years has become one of the most active Wikimedia projects [43]. Section 2.2.1 mentioned that KBs are incomplete due to the constant change of available information. Wikipedia also encounters this challenge as the number of articles and languages has vastly increased. An example is the population of Trondheim which might differ in the English and

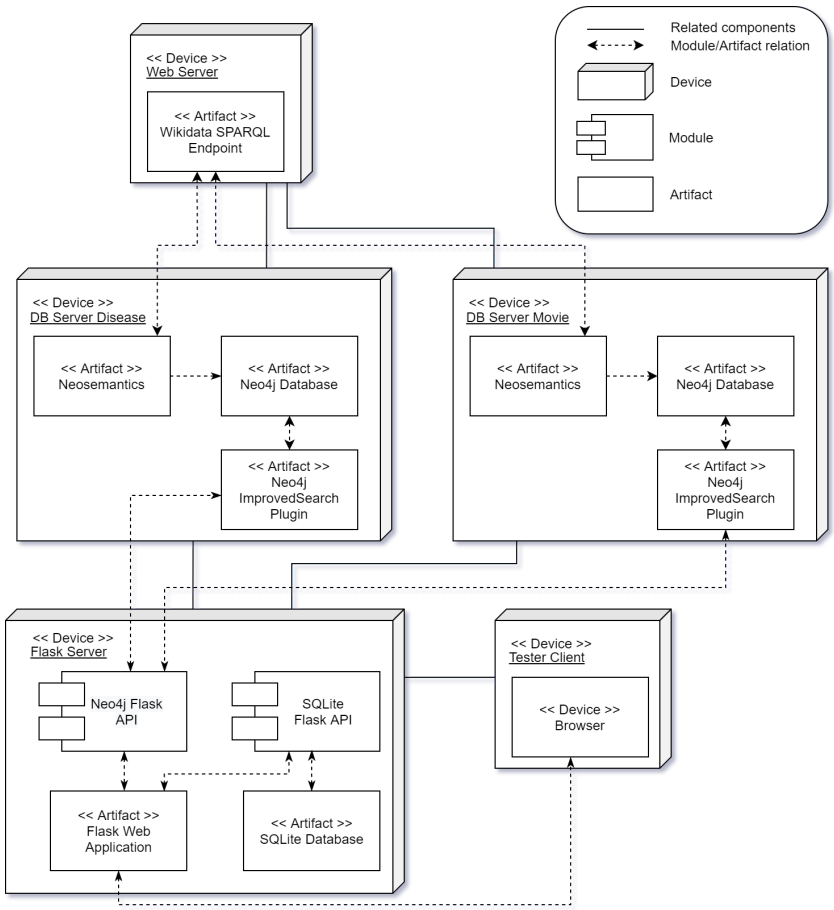


Figure 4.1: A deployment diagram giving an overview of how the different systems are connected.

the Norwegian article. One way Wikidata aims to solve this challenge is by acting as a *"[...] central storage for the structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wiktionary, Wikisource, and others."*[1]. Wikipedia articles can retrieve facts, such as Trondheim's population, from Wikidata. That way, there is only one central knowledge base to keep up to date, and Wikipedia articles about the same topic stay consistent across languages.

The data in the Wikidata repository consists of *items* with unique identifiers beginning with a Q followed by a number. Each item also has a *label*, a *description*, and any number of *altNames*. An item may also have any number of *statements* describing its characteristics. A statement is a *property-value pair*. Together with the item, the statements form a familiar data structure. The *item - property - value* triple, shown in table 4.1, is in accordance with a *subject - predicate - object* triple

[1] `https://www.wikidata.org/wiki/Wikidata:Main_Page`

in the RDF model described in section 2.2.2.

Item (Subject)	Property (Predicate)	Value (Object)
Q90	P1376	Q142
Paris	capital of	France

Table 4.1: Example of a triple from Wikidata

Section 2.3 mentioned that entities are not always represented as retrievable units. As a result, it can be necessary to generate entity documents as a representation, using entity detection and entity linking. Wikidata provides an explicit representation for each entity, making it simple to work with. Given the vast range of data and deep ontology, it is a suitable choice to control the data domain that is collected. This includes entity types and relations, in addition to statements used for entity fields. Because of this preexisting data structure, there is no need to utilize further methods to create additional entity representations.

4.1.1 Subsection of Wikidata

Wikidata was a natural choice to use as a data source based on its extensive range of data and relations being easily accessible through the SPARQL endpoint. This can be publicly queried through their web-GUI[2]. Neosemantics is a plugin for Neo4j that enables the use of RDF and associated vocabularies like OWL, RDFS, and more[3]. This made it easy to import the Wikidata datasets as n-triples from RDF data. Each Wikidata entity contains a field with alternative names. This field can be empty or contain several values at once. To handle the varying number of names, the variable "multival" in the configuration file of Neosemantics was set to "ARRAY". This enabled the storage of multiple values because properties were stored in an array.

Other data sources such as DBpedia[4] and YAGO[5]were also considered. However, these options had some shortcomings considering the needs of the research. Firstly there was a need for domain-specific subsets of data. These types of datasets are easier to get familiar with which is necessary in order to create ideal ranking results as there is a lack of available test sets. Furthermore, being familiar with the datasets can give a better understanding of the ranking models' behavior. Due to its SPARQL endpoint, Wikidata has a clear advantage over the other options. This made it easy to specify the exact scope of subsections through SPARQL queries. Secondly, a domain-specific dataset, such as the ones fetched from Wikidata, often ensures some repeated structure. All entities contain fields like *Name*, *Description*

[2] `https://query.wikidata.org/`
[3] `https://neo4j.com/labs/neosemantics/`
[4] `https://wiki.dbpedia.org/`
[5] `https://yago-knowledge.org/`

and *AltNames* which lays a good foundation for consistency. On the other hand, Wikidata entities have a wide range of different properties and relations. This provides a good balance between a predictable structure, while still supplying differently structured subsections.

One of the chosen subsections of Wikidata was the disease category. All diseases stored in Wikidata, the symptoms of these diseases, and the drugs used for the treatment of these diseases were retrieved. A big strength of Wikidata is the support for a wide range of languages. However, in the proposed prototype, only the English instances of entities were retrieved. Combining multiple languages complicates text operations depending on language structure such as removing stopwords and stemming which will, in turn, hurt performance. On the bright side, the prototype would work just as well with other languages by using text operations adapted for the given language. Listing 4.1 shows the query used to fetch the data and graph 4.2 displays some of the nodes one edge away from the influenza entity in the KG. The disease dataset includes a total of 45 287 RDF triples.

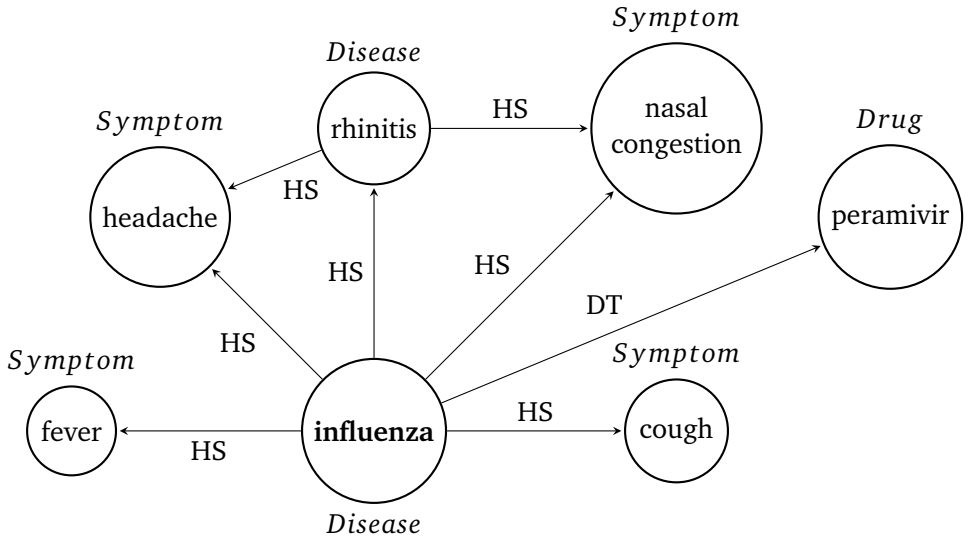


Figure 4.2: Example graph of a disease from disease the knowledge graph. [HS = HasSymptom, DT = DrugTreatment]

To reduce the source of error from result biases in the specific domain of diseases, an additional subsection of Wikidata was used for the testing. Using a similar SPARQL query with updated labels and IDs, a considerable movie dataset was retrieved. This dataset mainly focuses on films and their directors containing a total of 458 488 RDF triples. The plan was originally to include actors with an *ActedIn* relation to a Film. However, Wikidata times out the query if it takes longer than 60 seconds to process. This sets a limitation to the number of triples that can be retrieved. Including *ActedIn* drastically reduces the number of films that are re-

trieved. As a result, actors were excluded from the query in order to optimize the number of films in the dataset. Figure 4.3 is a graph visualization of the nodes one edge away from the American director Lilly Wachowski.

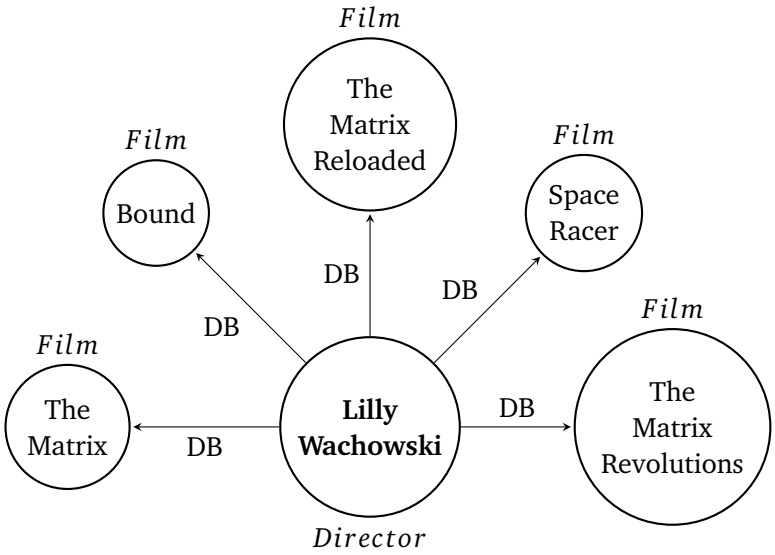


Figure 4.3: Example graph of a director from the movie knowledge graph. [DB = DirectedBy]

```
PREFIX neo: <neo4j://voc#>
CONSTRUCT {
  ?disease a neo:Disease ;
             neo:name ?diseaseLabel ;
             neo:altNames ?diseaseAltLabel ;
             neo:description ?diseaseDescription ;
             neo:usesDrug ?drug ;
             neo:hasSymptom ?symptom .
  ?drug a neo:Drug ;
          neo:name ?drugLabel ;
          neo:altNames ?drugAltLabel .
  ?symptom a neo:Symptom ;
             neo:name ?symptomLabel ;
             neo:description ?symptomDescription ;
             neo:altNames ?symptomAltLabel .
}
WHERE {

  SELECT ?disease ?diseaseLabel ?diseaseDescription ?drug
        ?drugLabel ?symptom ?symptomLabel ?symptomDescription
                  (GROUP_CONCAT(DISTINCT(?altLabelDisease);
                  separator = ",␣") AS ?diseaseAltLabel)
                  (GROUP_CONCAT(DISTINCT(?altLabelSymptom);
                  separator = ",␣") AS ?symptomAltLabel)

  WHERE {

      ?disease wdt:P31/wdt:P279* wd:Q12136 ;
               rdfs:label ?diseaseLabel .
```

```
      FILTER(lang(?diseaseLabel) = "en") .

      ?disease schema:description ?diseaseDescription .
      FILTER(lang(?diseaseDescription) = "en") .

      OPTIONAL {
        ?disease skos:altLabel ?altLabelDisease .
        FILTER(lang(?altLabelDisease) = "en")
        }
      OPTIONAL {
        ?disease wdt:P2176 ?drug . ?drug rdfs:label ?drugLabel .
        FILTER(lang(?drugLabel)= "en")
        }
      OPTIONAL {
        ?disease wdt:P780 ?symptom .
        ?symptom rdfs:label ?symptomLabel .
        FILTER(lang(?symptomLabel)= "en") .
        ?symptom schema:description ?symptomDescription .
        FILTER(lang(?symptomDescription) = "en") .
        ?symptom skos:altLabel ?altLabelSymptom .
        FILTER (lang(?altLabelSymptom) = "en")
        }
    }
  GROUP BY ?disease ?diseaseLabel ?diseaseDescription ?drug
           ?drugLabel ?symptom ?symptomLabel ?symptomDescription
}
```

Code listing 4.1: SPARQL query to fetch disease data from wikidata

4.2 Neo4j Database

Neo4j is a high-performance graph database platform used to store nodes and relations between them instead of storing static tables, while still having benefits like ACID transactions and an easy-to-use query language (Cypher)[6]. Additionally, it is compatible with a wide range of data models, including RDF. This makes it convenient to fetch and import Wikidata subsections to the database using the SPARQL query in code listing 4.1. Furthermore, by using Neo4j it is simple to take advantage of *user-defined procedures*[7] which extend upon Neo4j by adding custom code using the Neo4j Java driver. As a result, customized indexing and search functions can be made, and used directly in the database with Cypher queries. This is important because no single library, plugin, or platform supports all ranking models and required indices used in this research. Consequently, there was a need for a flexible database platform that could easily be expanded upon with custom plugins, which Neo4j delivers. A more detailed view of the user-defined procedures used in this prototype is discussed in section 4.3.2.

Other database options were also considered. A prevailing RDF triple store con-

[6] https://neo4j.com/

[7] https://neo4j.com/docs/java-reference/current/extending-neo4j/procedures-and-functions/procedures/

sidered in this research was GraphDB[8]. It is a scalable RDF database that specializes in making search in semantic data easier and more efficient. At first glance, this seems very promising. However, this is based on already existing functionalities and libraries. Although GraphDB includes a Java API, it is not as seamless to create custom plugins compared to Neo4j. Consequently, Neo4j was a natural choice to develop the indices and ranking models from scratch.

Indexing

To save information about the nodes and their contents, unique nodes were created to store additional information. This way, relevant information was easily available and could be utilized when calculating the relevance scores for the different ranking models. A more traditional approach would be to store a horizontal or vertical index in a separate file. Instead of taking a traditional approach, a new index structure was created directly in the database alongside the data to serve as a self-contained and independent plugin for Neo4j. As a result, everyone running Neo4j 4.0 or above can include the plugin jar file in the *plugin/* directory to utilize it in their projects.

In the prototype, two similar indexes were used. One index for ranking models operating on the document level such as the VSM and BM25, and one index for ranking models operating on the field level like BM25F. These will be referred to as *Document level* index and *Field level* index respectively.

Index Node

The indexNode consist of 5 different fields:

Field Name	Description
ref	the id of the node the indexNode is based on
dl	the length of the document
terms	the unique terms that is present in the node
IDF	the IDF-score for the terms present in the node
TF	the TF for the terms present in the node

Table 4.2: indexNode

Table 4.2 represents how the indexNode is structured at the document level. The prototype also stores indices on the field level where all fields except *ref* are dynamically changed to include the field name of the fields in the node. This means that if a node has two fields, namely *field1* and *field2*, the field length would be called *field1*Length and *field2*Length. This is done for length, terms, IDF, and TF, and ensures each field can have separate values. These nodes are called *fieldedIndexNode*.

[8] `https://www.ontotext.com/products/graphdb/`

Two fielded versions of IDF can be calculated based on the characteristics of the data, one called *fieldName*GlobalIDF and one called *fieldName*LocalIDF. The global IDF is the commonly used IDF calculated across all fields in all entities, while the local IDF bases its calculation per unique field. This means that the IDF score calculated for all instances of *field1* may have a separate value compared to local IDF for all instances of *field2*. It is important to note that local IDF calculations will not be appropriate if the knowledge graph consists of a wide range of different entities and fields. This is because it may lead to many rare fields with a low number of unique terms, thus resulting in volatile and unreliable results. The use of local IDF will be described in more detail in 4.3.2.

Field Name	**Value**
id	2407
name	COVID-19
description	respiratory syndrome and infectious disease in humans, caused by SARS coronavirus 2
altNames	Covid-19, 2019 NCP, nCOVD19, Wuhan respiratory syndrome, WuRS, 2019 novel coronavirus pneumonia, COVID 19, COVID-2019, COVID19, nCOVD 19, nCOVD-19, SARS-CoV-2 infection, seafood market pneumonia, severe acute respiratory syndrome type 2, CD-19, Wuhan pneumonia, 2019 novel coronavirus respiratory syndrome, 2019-nCoV acute respiratory disease, coronavirus disease 2019, Coronavirus disease 2019
uri	`http://www.wikidata.org/entity/Q84263196`

Table 4.3: Fields in the entity node representing COVID-19

Table 4.2 and table 4.5 showcases the indexNode and the fieldedIndexNode respectively for the entity *COVID-19* showed in table 4.3. The fieldedIndexNode in table 4.5 only includes the name and description fields to highlight the key differences to a standard indexNode. The full fieldedIndexNode would also include terms, dl (length), TF, and IDF fields for the *altNames* field. In this example, the fieldedIndexNode uses the global IDF variant. The positions of a term in the term fields match the positions in its corresponding TF and IDF fields. For instance, in table 4.2, the stemmed term $terms[2] = coronaviru's$ TF value is $tf[2] = 5$ and the IDF value is $idf[2] = 11.52$.

A weakness of this design is scalability. To support all ranking models included in the plugin, two new nodes will be created for every node to index; one indexNode and one fieldedIndexNode. For very large datasets and KGs, tripling the number of nodes is not practical. This is a compromise between weak scalability and the strengths of the usability and flexibility of the plugin. This compromise was evaluated as advantageous as, in this prototype, domain-specific datasets are of interest which are smaller by nature. These datasets will never exceed the number of RDF-

Field Name	Value
ref	2407
dl	66
terms (stemmed)	["2019", "respiratori", "coronaviru", "covid", "019", "syndrom", "diseas", "pneumonia", "wuhan", "novel", "19", "ncovd", "sar", "acut", "2", "ncp", "ncovd19", "wur", "02019", "covid19", "0cov", "02", "infect", "seafood", "market", "sever", "type", "cd", "0ncov", "infecti", "human", "caus"]
TF	[6, 5, 5, 4, 4, 4, 4, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
IDF	[12.26, 7.026, 11.52, 11.03, 11.26, 2.388, 0.746, 8.489, 13.84, 13.84, 8.420, 13.84, 12.84, 6.072, 4.599, 13.84, 13.84, 13.84, 13.84, 12.84, 12.84, 8.637, 4.866, 13.84, 12.84, 5.692, 2.274, 11.52 13.84, 6.160, 1.086, 4.824]

Table 4.4: Fields in the indexNode for COVID-19

Field Name	Value
ref	2407
nameLength	2
descriptionLength	9
nameTerms	["covid", "019"]
descriptionTerms	["respiratori", "syndrom", "infecti", "diseas", "human", "caus", "sar", "coronaviru", "2"]
nameTF	[1, 1]
descriptionTF	[1, 1, 1, 1, 1, 1, 1, 1, 1]
nameGlobalIDF	[11.04, 11.26]
descriptionGlobalIDF	[7.02, 2.38, 6.16, 0.74, 1.08, 4.82, 12.84, 11.52, 4.599]

Table 4.5: Fields in the fieldedIndexNode for COVID-19

triples where size will be a problem. However, it is important to be aware of this weakness when considering using the plugin for other environments.

Corpus

Some ranking models, like the VSM described in chapter 2.5.3, require an overview of all the unique terms occurring in the whole document collection. In the presented prototype, this information is stored in a singleton node called *Corpus*. The corpus node contains a field with a bag-of-words representation of the document collection stored as an array. An example would be if there are two nodes:

Node1: [field1: <dog, cat, red, red>]
Node2: [field1:<dog, cat, red, red> field2:<dog, dog, rat blue>]

The corpus would look like this:
Corpus: [dog, cat, red, rat, blue]

IDF

Similarly to the corpus node, the IDF scores for all unique terms are stored in a singleton node as an array. Here, the IDF score for a term $Corpus[i]$ will correspond to $IDF[i]$. These two nodes are mainly used to create query vectors for the vector space model procedure which is further described in section 4.3.2.

Data Stats

As seen in section 2.5.4 and 2.5.5, BM25 and BM25F are dependent on average document length and field lengths respectively. These statistics are stored during the indexing in a singleton node called *DataStats*. Table 4.6 shows the DataStats node for the disease dataset containing the fields *name*, *description*, and *altNames*.

Field Name	Mean Length
meanDocumentLength	23.40
name	3.207
description	6.321
altNames	13.87

Table 4.6: Fields for DataStats node

Parameters

Lastly, the database includes a singleton node called *Parameters* to tune and keep track of the free parameters for BM25, BM25F, and BM25FF. These parameters include *b* and *k1* for BM25, and the fielded variations for BM25F and BM25FF. On

the field level, a *boost* factor parameter that represents the weight of a field is also included. This node is included to easily read and tune the free parameters of the ranking models during testing and evaluation.

4.3 Neo4j Plugin - ImprovedSearch

In order to conduct the research in this thesis, a custom Neo4j plugin was developed as part of the prototype. *ImprovedSearch* is a general-purpose fulltext search plugin that works on any Neo4j database. As discussed in section 2.5.6 and 2.7, there are some theoretical reasons and some empirical evidence that indicates ranking models designed for unstructured data may be inferior to their fielded counterparts when dealing with certain data structures. Since vanilla Neo4j only supports fulltext search using Lucene queries[9], ImprovedSearch expands the search options to better fit the specifics of the data structure and research requirements.

This plugin is the heart of the research. It includes custom index procedures as well as custom procedures for ranking models such as the VSM, BM25, BM25F, and a new BM25 variant named BM25FF as it goes one step further on the fielded level.

4.3.1 Model

The architecture of the ImprovedSearch plugin is influenced by being configured using Apache Maven[10] and following the recommended procedure project template provided by Neo4j[11]. Figure 4.4 is a simplified UML class diagram of the plugin. It's simplified in the sense that not all classes, fields, and methods are included; only the most important ones to illustrate the architecture of the plugin that supports the indexing and ranking procedures. The emphasized methods are the custom indexing and ranking procedures provided by the plugin.

The two main classes for the custom procedures are the *Ranker* and the *Indexer*. These are the abstract classes the ranking and indexing procedure classes inherit. Each Indexer constructs a corresponding *Corpus*. These classes contain, among other fields, a BoW and the IDF values, and are mainly used to create the inverted and ImprovedSearch index. To create the indexNodes, the Indexers constructs a *Document* object for each node to index. The Document class, with the help of the *CardKeyword* class, handles all of the text operation steps described in section 2.4.1. The Ranker classes on the other hand are fairly independent. They convert the query string to a Document object to perform the same text opera-

[9] `https://neo4j.com/developer/kb/fulltext-search-in-neo4j/`
[10] `https://maven.apache.org/`
[11] `https://github.com/neo4j-examples/neo4j-procedure-template`

tions on the query as is done on the indexed nodes. Aside from that, these classes retrieve indexNodes and rank results as described in the next section.

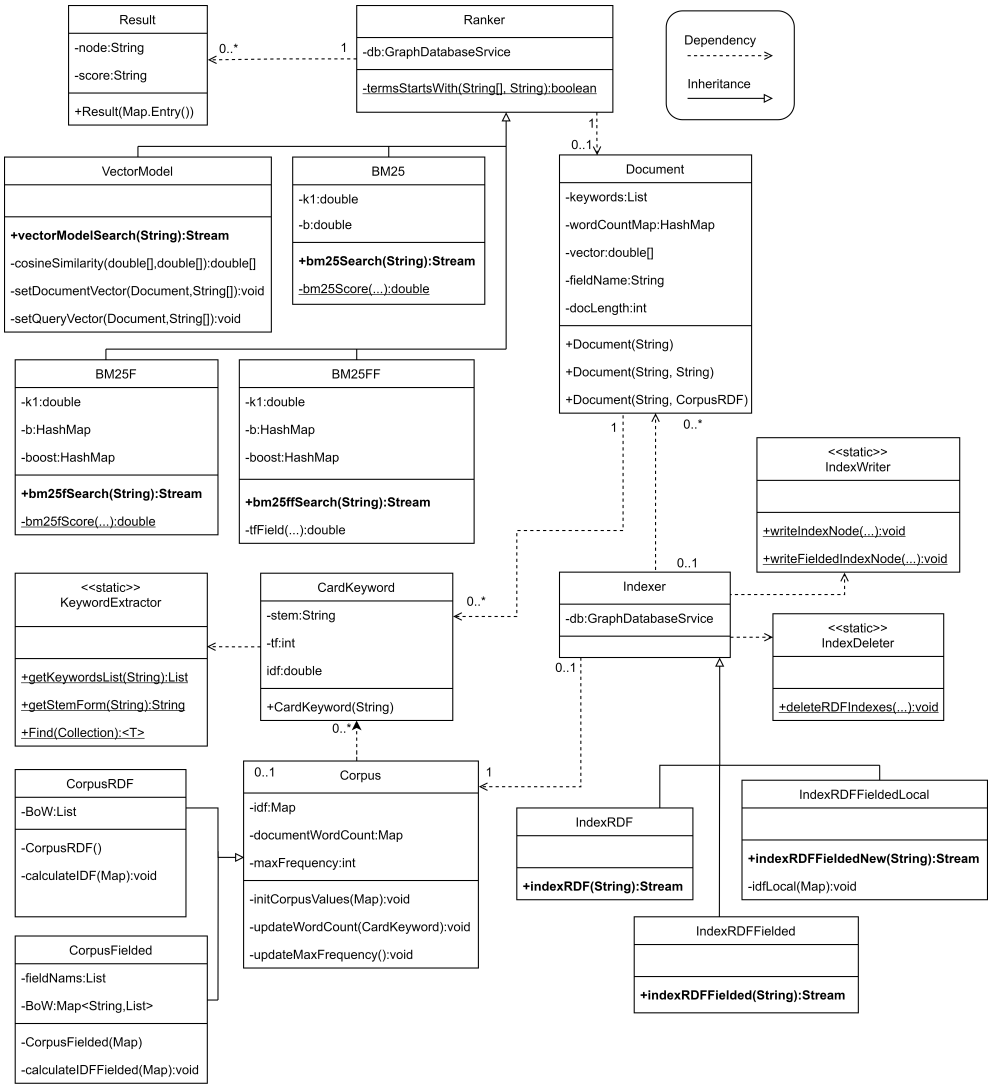


Figure 4.4: An UML class diagram illustrating the core class architecture of the plugin

4.3.2 Neo4j Plugin Custom Procedures

To execute the ranking procedures, both the indexing, preparation for retrieval, and the ranking models themselves were needed. Here the components of each procedure implemented in the plugin will be described in more detail.

Index procedures

The plugin includes four indexing procedures; *indexRDF()*, *indexRDFFielded()*, *indexRDFFieldedNew()*, and finally *deleteIndexes()*. The indexRDF procedure creates the indexNodes previously showed in table 4.4 along with the Corpus, IDF and Data stats nodes. Thus, this procedure creates an index on the document level and is required for VSM and BM25 ranking. The indexRDFFielded() and indexRDFFieldedNew() procedures, on the other hand, indexes on the fielded level, and are required for BM25F and BM25FF ranking respectively. These procedures create fieldedIndexNodes previously showed in table 4.5, where indexRDFFieldedNew() will include localIDF fields as well. Finally, the deleteIndexes() procedure simply deletes all indexes from the ImprovedSearch plugin.

All these index procedures, with the exception of deleteIndexes(), receive a Cypher query that returns the nodes to index as a string argument. These indexes coexist without overlap or conflicts. Listing 4.2 shows the query to index Disease and Symptom nodes on the document level.

```
CALL improvedSearch.indexRDF("
   MATCH (d:Disease) RETURN (d)
   UNION
   MATCH (d:Symptom) RETURN (d)
   ")
```

Code listing 4.2: Example use of indexRDF() procedure

As mentioned in section 2.5.1, there are many different variations of TF and IDF. By default, the plugin uses raw frequency and the standard inverse frequency IDF, but these might not be ideal for every situation. As a result, the other variations were implemented and experimented with. There were, however, no obvious advantages found compared to the standard variations. Consequently, other variants were included in the plugin, but not used in the research for this thesis.

Vector Space Model

Vector space model search can be used by running the following custom procedure as a Cypher query:

```
CALL improvedSearch.vectorModelSearch("<query>")
```

This procedure takes advantage of the indexRDF() index which is a prerequisite. The query vector is constructed by finding the terms' position indices in the Corpus node which will correspond to the terms' IDF value position in the IDF node. The document vector for the relevant documents containing query terms is constructed using the TF- and IDF-field for the corresponding indexNode. The cosine similarity is then calculated between the query and the documents as described in 2.5.3.

When matching query terms to documents, Java's *startsWith()* from the String

class is used. This means that if a query includes a term like "corona", a document containing the term "coronavirus" will be matched. Compared to the *equals()* method, startswith() will include many documents equals() will exclude, all while having the same computational complexity. The *contains()* method was also considered as this will include even more relevant documents and improve the recall. For a query term "virus", the document containing the term "coronavirus" will not be matched when using the startsWith() method, but will when using contains(). However, contains() has much higher computational complexity, thus not scaling as well. Despite the recall improvement when using startsWith() and contains(), there is no guarantee for precision improvements. Irrelevant entities partially containing similar terms by chance may be retrieved hurting the precision score. Overall, efficient retrieval and ranking were regarded as more important, than the extra recall improvements contains() might gain over startsWith(). This applies to all search procedures in the plugin.

As a reminder, Lucene Fulltext Search is a combination of the Boolean model and the vector space model calculated on the field level. The vector model search procedure, however, is originally on the document level. This procedure was created to get an overview of Lucene's performance on fielded documents compared to the standard vector space model.

BM25

BM25 ranking can be used by running the following custom procedure as a Cypher query:

```
CALL improvedSearch.bm25Search("<query>")
```

Similar to the vector model procedure, bm25Search() takes advantage of the indexRDF() index. When running the procedure, indexNodes corresponding to relevant documents containing query terms as well as the meanDocumentLength field from the DataStats node will be fetched. This includes all the variables needed to calculate the BM25 score, as described in section 2.5.4, for all the relevant documents. The default values for k_1 and b are 1.2 and 0.75 respectively but can be tuned by using a setParameter procedure.

```
CALL improvedSearch.setParameter(<k1-value>, <b-value>)
```

BM25F

BM25F ranking can be used by running the following custom procedure as a Cypher query:

```
CALL improvedSearch.bm25fSearch("<query>")
```

As bm25fSearch() is on the field level, it takes advantage of the indexRDFFielded() index. Similarly to bm25Search, it starts by fetching the relevant fieldedIndexNodes

and fielded mean lengths from the DataStats node. It will then calculate the BM25F score for the relevant documents as described in section 2.5.5. The default value for k_1 is 1.2 and the default values for all fielded b_c parameters are 0.75. Lastly, the default boost factor, ω_c, for all fields are 1. These parameters can be tuned by using the setFieldedParameter procedure.

```
1 improvedSearch.setFieldParameter(<k1-value>, <fieldName>, <b-value>, <boost-value>)
```

BM25FF

BM25FF ranking can be used by running the following custom procedure as a Cypher query:

```
1 CALL improvedSearch.bm25ffSearch("<query>")
```

BM25FF is a new BM25F variant developed in this research. The main difference between the two is that the IDF values used in BM25FF are calculated on the field level. While all terms have a global IDF value for BM25F, BM25FF regards terms in different fields as completely independent. This means a term might have different IDF values for each field. As a result, the ranking formula used is the same for both BM25FF and BM25F, but they take advantage of different index strategies. The fundamental idea is that a term might have a different discriminatory factor when appearing in different fields. An example of this is the difference between the 'name' field and the 'altNames' field in the disease dataset. When searching for a specific disease, terms could be more relevant when appearing in the 'name' field, which is generally shorter, than in the 'altNames' field which includes slightly different variances of the same terms. By looking at each field's IDF separately, an entity's relevancy score would be higher if the field that contains the query term has a high IDF score. This means that, in this example, repeated mentions of query terms in the 'altNames' field will have less impact on the TF-IDF score compared to using the global IDF.

The bm25ffSearch procedure uses the fields with the -LocalIDF suffix in the fielded-IndexNodes. Hence, indexRDFFieldedNew() is a prerequisite. The free parameters are tuned the same way as for bm25fSearch.

4.4 Web Application For Survey

To assess the different ranking models, it was a priority to have an easy, accessible way to gather as many survey responses as possible. With restrictions on face-to-face meetings during the COVID-19 pandemic, a web application to effectively gather feedback from testers was developed.

4.4.1 Pages

Here is an overview of the different pages in the survey that will be presented in the order the testers encounter them, as well as their overall purpose.

Landing page

The landing page is the first page the testers encounter. The page contains information about the scope of the survey, estimated time for completion, the privacy concerns of the tester, and how to navigate the page. This information is available at any time through the navigation buttons on the top-left-hand side of the page.

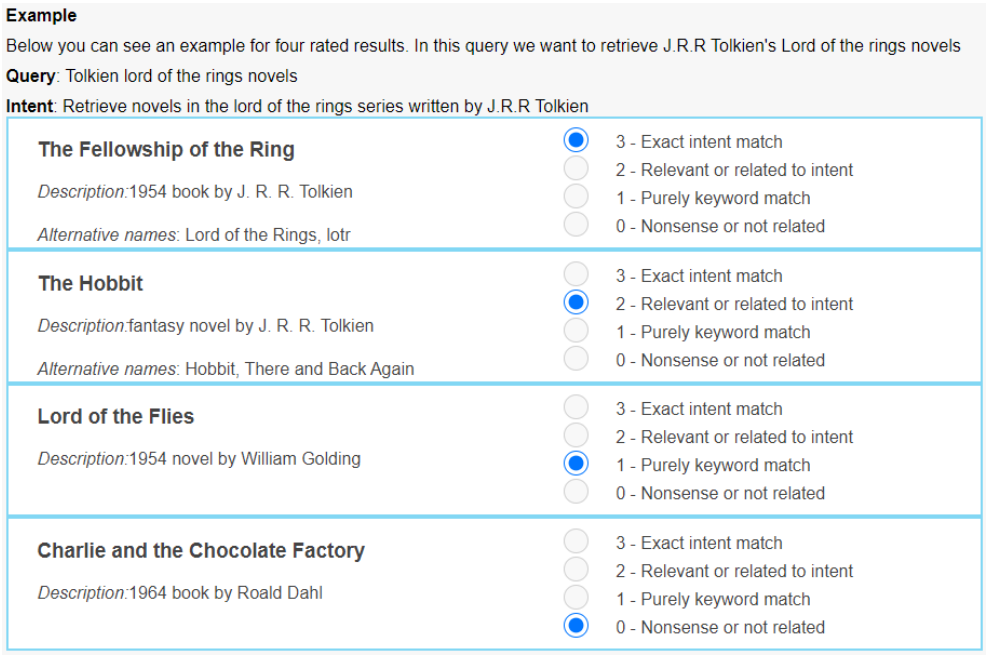


Figure 4.5: Guiding example of possible relevancy assessments

The page also includes information on how to use the ranking system and an example to further the testers' understanding. This can be seen in figure 3.1. The example is present to ensure the tester fully understands the task ahead and is simply in place to further exemplify how each relevancy score should be considered. In order to minimize the influence of evaluations, the given example to the testers concerns books, which is a separate domain to the two datasets in the survey. In addition to the example shown, there is some supplementary text given to explain the reasoning behind each of the ratings.

Home

The home page is where most of the navigation happens. Here the tester is presented with two buttons, one for each dataset. Each button is accompanied by a short description of the contents of each dataset and redirects the tester to the survey page of the chosen dataset as seen in figure 4.6.

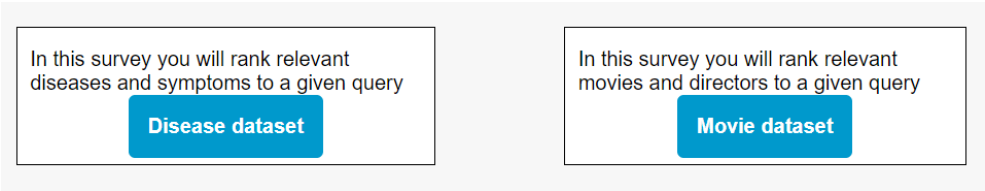


Figure 4.6: Home page buttons

Survey

The survey page consists of three parts. The header, query information, and the form that presents the results. The header displays what type of method is used for the search, namely BM25, BM25F, or Lucene Fulltext Search, and which of the two datasets are queried. The query information indicates which query results are currently being presented, as well as a clarification of the information need and intent behind the query. The form is the main part of the survey and is presented in figure 4.7. Here the tester is presented with the top ten results to the query. Each result consists of the name of the entity, a description of the entity, and, if present, the alternative names. Since all the content is derived directly from Wikidata, the quality of descriptions and alternative names may vary. On the right-hand side of each result, there is a set of radio buttons that allow the tester to give feedback on their perceived relevancy of the result to the query. There are four options, each accompanied by a description of which degree of relevance they represent.

When the tester has rated all results for each query regarding one of the datasets, they are redirected back to the landing page. Here the previously chosen dataset will be unavailable, leaving only the button directing the tester to the queries of the other dataset. This way the options are limited, the tester is guided and the room for user error is minimized.

With an aim of collecting the most survey answers for every query, the order in which the queries were presented to the testers was randomized. If every tester was presented with the same queries in the same order, and only finish half the survey, the queries last to be shown would receive significantly fewer answers. By mixing up the order in which queries were presented, there was an assurance that there would be no major imbalances in the number of answers for each query.

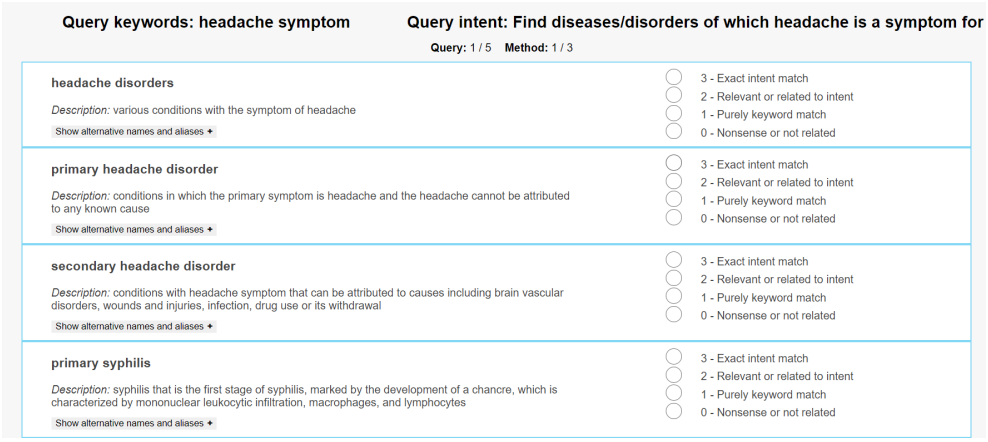


Figure 4.7: Survey page with top four results for BM25F

When working with a survey, how the information is displayed may skew the results. There can be unintentional guiding by the use of words or visual cues that gives one choice unwanted precedence over another. One of these possible cases was the order in which the results were presented to the tester. Traditionally, they are accustomed to the top result being the most relevant, and the last result is the least relevant. This is based on the frequent use of search engines that aim to present you with the most probable result first in order to meet your information needs. This could, after a number of queries, result in noticing a pattern where the earlier results are generally more relevant. This can affect the testers to feel inclined to give the earlier results a better relevance score than the latter results, and as a result not be as thorough with the ranking. One solution could be to mix up the order in which the results are presented. This could help make sure the testers do not have an inherent bias toward the former results. Ultimately it was decided not to change up the order as the placement of the results in each method compared to each other would be taken into consideration. This was because the methods usually result in a lot of the same entities with different relevance scores and therefore other placements in the top-k results.

Another way the results may be unintentionally skewed is by presenting the three ranking models in the same order. As with the individual results, the testers may be inclined to use more time and effort the first time the results to a query are presented, compared to the second and third. This may impact how thoroughly each result is rated, consistently making the latter ranking model less accurate evaluations. To combat this, the order in which the ranking models appeared for each tester was randomized. This would counteract the natural bias created by the order of the ranking models.

Completed survey

When the tester has gone through both datasets, they are redirected to a 'Completed Survey' page. This is a simple page present to express appreciation for the participation and inform them that they have completed the survey.

Additional page

The single search box became popular through search engines like Google, Bing, and DuckDuckGo. It allows users to freely search for information using keywords or even full sentences. To ease the preliminary testing of the different search methods, a 'single search box' page was created to freely query the different methods. The page consists of one single search input and three buttons, one for each of the methods. The page was used to get familiar with the datasets and how each method behaved, noticing potential biases and weak spots through different queries. This testing helped explore different queries and decide upon which types of queries could be interesting to focus on during the user testing. It also helped to get a feeling of which free parameters values for BM25 and BM25F worked best for the datasets.

4.4.2 Implementation of the Web Application

When choosing technologies to develop the web application, a compromise between simplicity and flexibility was regarded as important. Since the web app was a means of collecting results and data, the functionality of the website was prioritized over the design. As a result, front-end frameworks like React and Vue were regarded as excessive. The focus was, therefore, on technologies and frameworks to power the application's backend.

Flask

Flask[12] is a lightweight micro web application framework written in Python. Its strengths are that it's easy to set up and work with while being able to scale to big and complex web applications. Flask applications are split into three parts; *static*, *templates*, and *views* also sometimes called *routes*. The static is where images, javascript, and CSS are located. Templates are where the application's Jinja[13] templates are located. Finally, the views are where the application logic is handled. This includes URL routing and handling HTTP requests. Neo4j includes a well-documented, easy-to-use Python driver that makes the communication between the Neo4j database and the application seamless.

Some deliberate choices were made while developing the application. Firstly the number of survey answers to every query needed to be as high as possible. To

[12] `https://flask.palletsprojects.com/en/1.1.x/`

[13] `https://jinja.palletsprojects.com/en/2.11.x/`

achieve this, the application had to be easy to use and not require a huge effort from the test subjects. For example, user registration and login were considered too high an effort for the testers. As a result, the application remembers each tester and their uncompleted survey answers in Flask's session cookies. This makes it possible to complete parts of the survey at a time, keeping track of where the testers last left off without the need for registrations and logins. Table 4.7 shows the cookies stored in the application.

Session Cookie Name	**Description**
user_id	Unique ID genereated by a *uuid4()* function
disease_queries	Queries for the disease dataset not yet evaluated by user
movie_queries	Queries for the movie dataset not yet evaluated by user
methods	List of Ranking models compared
disease_index	Index in the method list for the current ranking model to evaluate a disease query
movie_index	Index in the method list for the current ranking model to evaluate a movie query

Table 4.7: Session cookies stored in the web application

A disadvantage of this design is that nothing is stopping the testers from either changing browsers or deleting their session cookie to answer the surveys several times. This was not regarded as a huge risk, however, due to the fact that the test subjects were handpicked and therefore regarded as trustworthy to use the application as intended.

SQLite

SQLite[14] is a relational database management system implemented as a C-language library. Instead of a traditional client-server database, SQLite can be used as an on-disk file format to store data. As a result, it is very lightweight and requires no installation while still having the benefits of traditional relational database management systems. With these benefits in mind, SQLite was a natural choice for storing the survey results.

The web application's approach to storing the survey results was to regard each relevancy ranking to a query result independently. Consequently, two tables were created, one for each of the datasets. Here each row represents a tester's relevancy ranking for a single query result. This allows for the necessary analysis described in chapter 6. Figure 4.8 shows an ER-diagram for the data model used to store the results. *DataDisease* and *DataMovie* entities are where the survey results are

[14] `https://www.sqlite.org/index.html`

stored. Here the *method* attribute is the ranking model used for the given result, and *result_rank* is the index for the query result. This means a result rank of 0 is the result with the highest score for the given ranking model and query. Finally, *relevancy* is the relevancy score given by the test subject.

For the *Tester* entity, the *answered_disease* and *answered_movie* attributes are boolean values representing if the tester has gone through all the queries from the respective datasets in the survey.

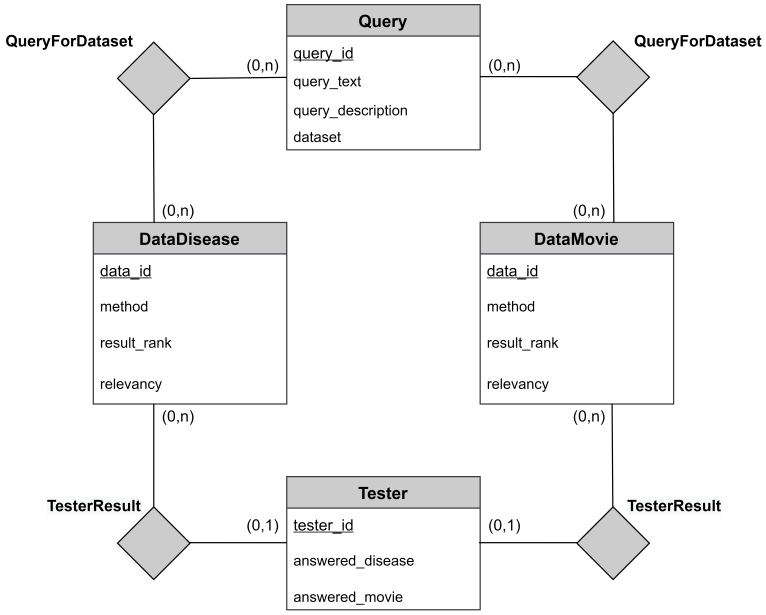


Figure 4.8: ER-diagram for database storing survey results

4.4.3 Preliminary Testing and changes

To mitigate errors, preliminary testing on a few test subjects was conducted. By testing the mainly finished product on actual users with no prior knowledge of the system, the most prominent misunderstandings could be found before presenting it to a larger number of users. The test subjects in the preliminary testing reflected main the group of the population of the full survey. This focused on young adults with previous experience with searching for information through different search engines Since the number of the preliminary testers was limited, the feedback was taken into account, carefully evaluated and changes were made where it was deemed beneficial.

Survey progress

One note gathered from the testing was the lack of visual representation of the current status of the survey. How much of the survey was finished and how much was left? With no indication of progress, the testers can feel lost and therefore less motivated to finish the entirety of the survey. To clarify this, an indication of the current progress was added by displaying which query and model the survey was currently presenting.

Clarifications

During the preliminary testing, quite a few queries were present in the database, and the formulation of the query intents was in their first iteration. This meant that some formulations were a bit unclear and needed to be specified further. The vague formulations left the test subjects with a feeling of uncertainty when ranking the results. This prompted a realization of how much the described intent could impact the rankings.

Another aspect that caused some confusion was related to the results. Instead of looking at each result as an entity, some expected each result to be a subsection of an article, such as the results presented by a Google search. They, therefore, found the experience a little frustrating because they assumed some information was potentially withheld, making it more difficult to fairly rank the results. To combat this, what kind of results were expected, and what they represented was to a larger degree explicitly explained in the introductory text on the 'about' page.

4.5 Complete Flow of Data

The object interactions and data flow of the prototype are illustrated in figure 4.9. As indicated by the red line, it is split in two; periodical and real-time. The periodical part regards importing and indexing data from a SPARQL endpoint. This needs to be initiated by a system administrator and thus happening periodically. The real-time part regards retrieving and ranking nodes from queries, as well as storing survey results. These are events triggered by web application activity, thus happening in real-time.

Periodical

The periodical part of the diagram happens when the database is initially filled with data, or when there is data to be updated. This is not done automatically, rather manually when there is a desire to update the data. First, Neosemantics is used to fetch data from the chosen data endpoint, namely Wikidata. This data is then stored in an RDF-triple format in the Neo4j graph database. Before the data can be queried using the different ranking models, it has to be indexed. The knowledge graph then utilizes the developed plugin, *ImprovedSearch*, to index the

recently fetched data. First of all, ImprovedSearch preprocesses the data given to it. As explained in 2.4, there are several ways to preprocess data. Here the chosen ones are tokenization, removal of stopwords, and stemming. To further prepare for ranking, IDF values are calculated for each of the preprocessed keywords. Now, the data is ready to be indexed as described in 4.3.2, and indexNodes to be queried are created and stored in the graph database.

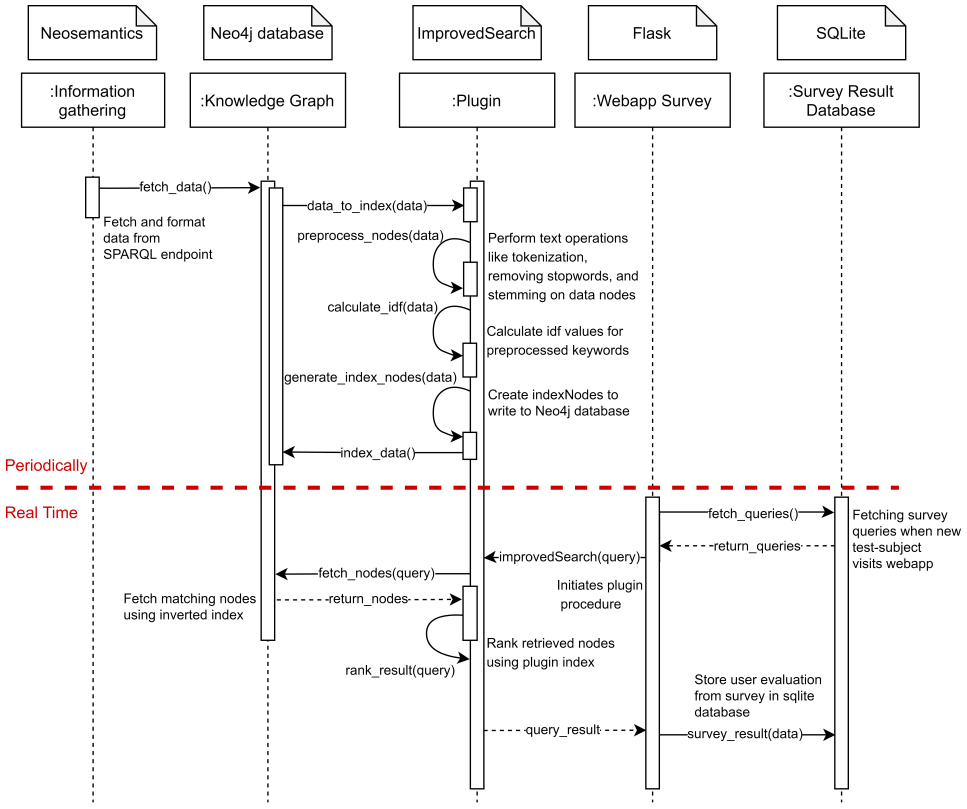


Figure 4.9: A sequence diagram illustrating the data flow of the system

Real Time

When testers visit the web application for the first time, a request to fetch the queries is sent from the Flask application to the SQLite database. The database then returns the queries, and the shuffling of query order mentioned in 4.4.1 occurs in the web application. As this is done, the survey can begin. When a tester begins their first rating by choosing a dataset, a ranking model is randomly chosen from the three available, and a request for search results are sent with a query to the ImprovedSearch plugin. This query is then further forwarded to the graph database to fetch the possible results to be ranked. These results are then sent as a response from the graph database to the plugin where the results are ranked

using the chosen ranking model. When this is done, the ranked results are sent to the web application to be displayed and rated. These ratings are then sent to and stored in the SQLite result database. This cycle is then repeated consecutively for every query the tester rates the results of.

Chapter 5

Data gathering and Evaluation

The objective of this chapter is to present the methodological approach to the research. This includes the method for the collection of data and the evaluation metrics for data analysis. It will give an overview of how and why the chosen methods are suited to answer the research questions.

5.1 Evaluation Strategy

The research is conducted using the survey strategy. To answer the research questions, quantitative research is needed to pragmatically evaluate and compare the ranking models' performances. Quantitative data will give a better and more concrete representation of how the ranking models perform and behave compared to a qualitative approach. The use of quantitative data lays the foundations for statistical analysis to clarify whether possible links between data results are actual, or in fact, present just by chance [44, p. 254]. Additionally, this research method generally accommodates a bigger sample size deemed advantageous for the statistical analysis.

As there is a lack of automated benchmark tests for search in RDF data, the research takes advantage of the proposed prototype platform to gather users' relevancy assessments. This was done by conducting a user study instead of solely basing evaluations on a predefined result set with 'right' and 'wrong' answers. As a result, traditional evaluation approaches such as precision, recall, MAP, and F-measure were not used throughout the evaluation process.

When choosing how many query results to present to the user, the size of each dataset was taken into consideration. A simple estimation was conducted on how many retrieved entities had a significantly higher score than others, indicating they were clearly deemed a better fit for the query and its intent. It was then noted generally how many results appeared before the score approached 0, or stagnated with little to no difference between entities. The research conducted in [5] searched for triples including the relations between entities which could

lead to a lot more potentially relevant matches than an entity-oriented approach. This was especially prevalent regarding the disease dataset where each disease often has unique domain-specific names. As a result, searching for a specific disease would lead to a handful of relevant entities rendering the rest irrelevant. Due to this, only the top ten query results were presented to testers.

5.1.1 Data Gathering

The chosen method for data gathering was the questionnaire method. In the web application described in detail in section 4.4, the test subjects were presented with a set of queries and the queries' top ten results for different ranking models. They went through each of these results rating its relevancy in accordance with the information need of the query. Other data gathering approaches, such as interviews, were also considered. The advantage of conducting interviews is that there is less room for misunderstandings when it comes to the information needs for the queries resulting in inaccurate relevancy assessments. During an interview, if there are any doubts, these can be answered consecutively. However, a questionnaire was ultimately deemed the most fitting for this research when examining the trade-off between the two. This is largely because the questionnaire is a more economical method to generate large amounts of data [44, p. 229] in addition to being more practical and feasible considering the research took place in the middle of the COVID-19 pandemic.

Queries

To gather data, each test subject had to assess the relevancy of every result to a query using all the ranking models. With this in mind, the survey consisted of ten queries, five for each dataset. Each query has its own characteristics and focuses on different aspects. The queries differ in the number of terms, common and uncommon terms in the dataset, as well as what type of answer is expected such as only one entity match or a number of entities. This way, how the models perform with a diverse set of queries could be explored. The queries are presented in table 5.1 and 5.2.

ID	Query	Query intent
D_Q1	Yellow fever	Find the disease named Yellow Fever
D_Q2	Covid-19	Find the disease named covid-19
D_Q3	Headache symptom	Find diseases/disorders of which headache is a symptom for
D_Q4	Influenza pandemic	Find pandemics caused by influenza
D_Q5	Fear of social interaction	Find phobias related to social interaction

Table 5.1: Queries and query intents for the disease dataset

ID	Query	Query intent
M_Q1	Matrix movies	Find movies in the “The Matrix” franchise
M_Q2	Lord of the rings	Find movies in the “The lord of the rings” franchise
M_Q3	Movies by Christopher Nolan	Find movies directed by Christopher Nolan
M_Q4	The circus Chaplin	Find the 1928 Charlie Chaplin movie “The Circus”
M_Q5	Wachowski directors	Find the Wachowski sisters (directors of The Matrix franchise)

Table 5.2: Queries and query intents for the movie dataset

Here, D_Q1 and M_Q4 are queries with the intent of only finding one specific entity. D_Q3 and M_Q3, on the other hand, have the intent to find multiple relevant entities. M_Q1 and M_Q2 are a middle ground searching for a few entities. This also includes D_Q5 if there are only a few such phobias present in the dataset.

Assessment of relevance

To assess how relevant a tester perceives each query result, they were asked to score each entity in the result set between 0 and 3. The definition of the four scores was given as seen in table 5.3:

Score	Description
0	The result is seen as nonsense or not related to the query keywords or intent in any way.
1	The result matches query keywords but does not correlate with the intent.
2	The result is relevant or related to the intent. It is not a perfect match but adds valuable information to the user.
3	The result is a perfect match for the query intent.

Table 5.3: Relevance scoring system

5.1.2 Sampling

Since the research questions aim to get a users’ perspective on the performance of different models ranking structured data, the population consists of people who are familiar with executing information search. Information search in this context was defined loosely as searching for information through search engines and looking for information in articles and books. The test subjects were found by reaching out to fellow students, peers, family members, and so on. These were people who

had experience with information search, both recreationally and academically, and would dedicate their time to answer the survey. This way of choosing test subjects is within the *non-probability sampling methods*, more specifically called *convenience sampling* [44, p. 98].

This method is very affordable to execute, ensures the group is easily accessible and lays the foundation for a higher response rate. The method is, however, prone to be biased as the chosen individuals might not be representative of the entire population. To counteract this, individuals from different age groups, genders, and educational backgrounds were included. This choice, however, means that the results can, while still shedding light on some possible answers, not be fully generalized and applied to the entire population.

5.2 Evaluation Metrics

There are several ways of evaluating ranking models. As mentioned in 2.7.1, there are two main approaches, namely automatic and user-based. The approach taken in this thesis is user-based, and the evaluation metrics chosen for this purpose are DCG, NDCG (@5 and @10), and the Kappa coefficient.

5.2.1 Normalized Discounted Cumulative Gain

When NDCG was introduced in section 2.6.4, it was mentioned that the normalized version of DCG utilizes both the DCG score and an ideal DCG referred to as IDCG. In this project, however, there are no readily available ideal result sets. Consequently, the ideal result sets had to be created for each query in order to utilize this evaluation measure. With the rating specifications in mind, ideal result sets were created based on the experience gained from working with the data, and the knowledge about the queries and intents. Because the IDCG is constructed for the purpose of this research, there is no way to be fully impartial. This could skew the results in some way, but should still give an indication of how the ranking models compare to each other as all ranking models would be equally affected.

The Ideal Result Set

To decide which entities fulfilled the requirements for each rating (0-3) in the ideal result set, detailed specifications for each rank had to be defined. This was particularly necessary as 'startsWith' was used to compare the query terms to entities, which resulted in partial query matches. How strict should the difference between a rating of 1 and a rating of 0 be? It is important that the testers make these decisions individually, as it is their perceived relevance that should be measured. To avoid DCG scores for test results being higher than the DCG scores from the ideal results set, the ideal set was created with the benefit of the doubt. This

meant when there was some uncertainty about the rating of an entity, it was rounded up to the higher score.

Table 5.4 visualizes the top five ideally ranked results for D_Q5. In this instance, two results were considered an "exact intent match", two results considered as "relevant or related to intent" and one as a "pure keyword match". Both "gelotophobia", and "specific social interaction" contain a type of fear or phobia in relation to social interaction. The two results given a score of two, on the other hand, do not perfectly match the intent but are still related to difficulties with social interaction or a fear involving other people. Lastly, "fear of frogs" is only a query match due to it being a phobia, while not having anything to do with social interaction.

Name	**Description**	**Alternative names**	**Score**
gelotophobia	type of social phobia consisting in the fear of being laughed at	-	3
specific social phobia	experiencing anxiety only in specific social situations	-	3
social emotional agnosia	agnosia that is a loss of the ability to perceive facial expression, body language and intonation, rendering them unable to non-verbally perceive people's emotions and limiting that aspect of social interaction	expressive agnosia	2
fear of medical procedures	any experience that involves medical personnel or procedures involved in the process of evaluating or modifying health status in traditional health care settings	medical fear	2
fear of frogs	phobia known as frog phobia or ranidaphobia	ranidaphobia, fear of frogs and toads, frog phobia, batracho-phobia	1

Table 5.4: Top five ideal set for "fear of social interaction" query

Evaluating Top N Results

Since the top ten results to each query were rated, it was natural to focus on top ten results when calculating the NDCG scores. In addition to this, calculations for the top five results were made. This was based on the notion that when searching

for information, a user will be less likely to look at results the further down in the result set it is. This is supported by Järvelin and Kekäläinen [33, p.424] that state “the greater the ranked position of a relevant document, the less valuable it is for the user because the less likely it is that the user will ever examine the document”. By shifting the focus onto the first five results, it could highlight how the models vary when it comes to result positioning.

5.2.2 Kappa Coefficient

The Fleiss Kappa coefficient was calculated to consider the degree of agreement between all the testers. It can be used to figure out the agreement among tester taking into account how likely testers are to agree on results being relevant or not happening by chance. This can give an idea of how probable each outcome is, and shed light on how divided testers’ opinions may be regarding some results.

The agreement was computed in two ways for each of the datasets. First, the original four categories for rating were used, namely a score between 0 and 3. This will be referred to as *Quaternary relevancy* in the following chapters. In addition to this, a simplified version of the results was employed. Similar to the Boolean model in section 2.5.2, the results were considered as relevant, 1, or not relevant, 0. To convert the previously used scale into the binary version, the levels of 3 and 2 were considered relevant to the query and its intent, and results of 1 and 0 were considered not relevant. This will be referred to as *Binary relevancy*.

Based on the visualization in section 2.6.5 a goal agreement among testers was set. The authors of [37, p. 362] said "with a large enough sample size, any kappa above 0 will become statistically significant". Because of the relatively small sample size, an average above 0.4, was seen as a reasonable goal. Anything above 0.4 was described as "Moderate agreement" which in the context of the smaller sample size meant the results would still have a statistical significance.

Chapter 6

Analysis

This chapter will give a detailed overview of the results from the survey detailed in the previous chapter. The findings will be evaluated using the chosen metrics, and any possible patterns will be examined. The impact of the evaluation metrics and the validity of the research and results will also be discussed.

6.1 Findings

The survey results consist of answers from 26 individuals rating a total of 8130 entities. Since convenience sampling was used, the population mainly consisted of computer science students between the age of 20 and 30 years old. In addition to this, about a third of the population consisted of participants with a wide range of backgrounds with ages ranging from 28 to 66 years.

6.1.1 NDCG

The NDCG score presented for each query is calculated as a mean of all survey answers. Each table showcasing the NDCG scores for the top ten and top five results is accompanied by a bar chart illustrating the three ranking models' performance in regards to each other.

Disease Dataset

As mentioned in section 4.1.1, the disease dataset is the smaller one with significantly fewer RDF-triples compared to the movie dataset. On the other hand, it contains the highest number of fields. Table 6.1 shows the average NDCG score for the top ten results for each query. It is notable that BM25F performs the best on query Q1, Q2, Q4, and Q5, while BM25 performs the best on Q3. However, the margins between these performances are relatively small.

Query	Lucene Full-text@10	BM25@10	BM25F@10
Covid-19 (D_Q1)	0.868	0.868	**0.914**
Yellow fever (D_Q2)	0.835	0.816	**0.853**
Headache symptom (D_Q3)	0.717	**0.831**	0.792
Influenza pandemic (D_Q4)	0.903	0.938	**0.952**
Fear of social interaction (D_Q5)	0.622	0.703	**0.779**

Table 6.1: NDCG score for topten queries on the disease dataset

All NDCG scores are well within 80% of a perfect score of 1 on Q1, Q2, and Q4. This indicates that all models predominantly rank the most relevant entities in the top ten result set, and the biggest differentiator is how high these entities are ranked. The differences in how saturation functions behave on multiple fields, which was discussed in section 2.5.6 (Comparing Saturation Functions), may give a theoretical explanation for these results. This is because the *AltNames* field is quite influential for the ranking results as it can contain valuable information in this dataset.

Lucene Fulltext Search might score entities highly because terms are matched across multiple fields like Name, Description, and AltNames. As a result, terms with lower IDF values might end up affecting the ranking more than terms with high IDF values, consequently, ranking the most relevant entities lower. Using Q2 as an example, "fever" is a common term in the dataset and will naturally appear in multiple fields. Thus, entities with "fever" (or "yellow") in multiple fields will be prioritized before entities with both "yellow" and "fever" in the same field.

As previously discussed, BM25F implements non-linear saturation for terms across fields. Consequently, the opposite of Lucene Fulltext will be true for BM25F; entities with fields containing both "yellow" and "fever" will score higher. BM25's saturation will behave similarly to Lucene Fulltext, but it appears that either the default parameter values of BM25 handle the saturation better for multiple fields, or the ranking algorithm is just a better fit for the dataset.

The NDCG scores for Q3 and Q5 are slightly lower, especially for Lucene Fulltext, indicating that either some relevant entities are not retrieved or ranked quite low. Looking at Q5, it could have the same saturation explanation. Q3, however, has no clear explanation why BM25 performed as well as it did.

Figure 6.1 shows a bar chart representation of the NDCG scores @10. Instead of scoring consistent NDCG values across all the queries, there is a wide range in the models' performance. All models seem to follow a similar trend, but there are

significant differences in the deviations between the best and worst performing. The smallest deviation being 4.53% for Q2 between BM25F and BM25, and the largest being 25% for Q5 between BM25F and Lucene Fulltext.

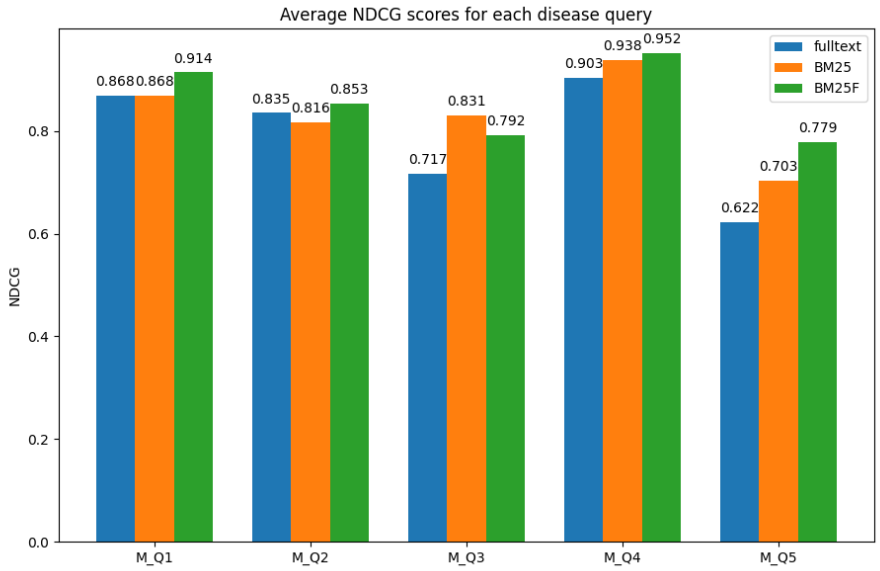


Figure 6.1: Chart overview of the models' NDCG scores @10 on the disease dataset

Under the assumption that the queries are representative for both the dataset and real-world use-cases, the fact that all the models are following the same trend is an indication they all share similar strengths and weaknesses. Looking at the deviations, however, we can see that the degree they are affected by these strengths and weaknesses differs.

Both Q1 and Q2 aim to retrieve only one single entity with an exact match taking the intent into account. We see that all models behave similarly relative to each other with small deviations. In contrast to Q1 and Q2, Q3 has several entities fulfilling the requirements for an 'exact match'. In fact, the ideal ranked set has a total of 8 entities with a relevancy score of 3. Interestingly, the deviation between Lucene Fulltext and the other models increases for a query like this. Finally, the biggest deviation is encountered with the drop in scores for Q5. This is a query with only a few entities matching the intent completely, but quite a few potentials to partially match the intent or keyword match the query. These factors will evidently make it much harder to retrieve and rank the most relevant entities first. Looking at the results, Lucene Fulltext is affected by this to a larger degree than the other models, particularly BM25F.

Looking at only the first five results, we get the scores presented in table 6.2 and figure 6.2. The results are quite similar to the top ten results with the biggest difference being lower scores across all models for Q1. However, this is not regarded as significant since the differences are relatively small and the rest of the queries follow the previous trend. Overall, the points discussed for the top ten results still hold ground here.

Query	Lucene fulltext@5	BM25@5	BM25F@5
Covid-19 (D_Q1)	0.817	0.830	**0.860**
Yellow fever (D_Q2)	0.846	0.853	**0.882**
Headache symptom (D_Q3)	0.818	**0.908**	0.833
Influenza pandemic (D_Q4)	0.910	0.990	**0.997**
Fear of social interaction (D_Q5)	0.609	0.696	**0.760**

Table 6.2: NDCG score for top 5 query results on the disease dataset

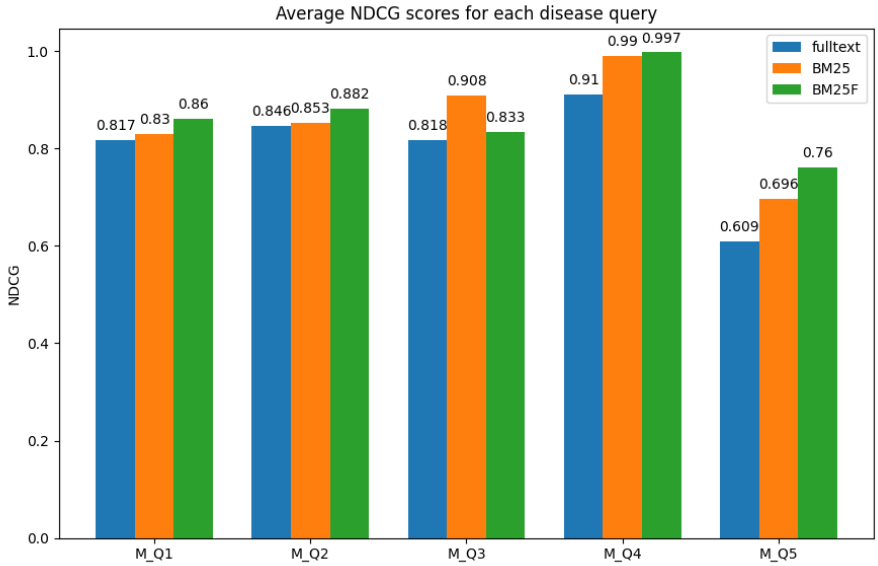


Figure 6.2: Chart overview of the models' NDCG scores @5 on the disease dataset

Movie Dataset

Looking at the results for the movie dataset there are considerably larger deviations between the best performing and the worst performing model. The results in table 6.3 reveal that Lucene Fulltext performs the best for all queries except Q3 where BM25 scored the highest. The smallest deviation between the best performing and the worst performing model is present with Q3 being 15.6% between BM25 and BM25F. On the other hand, the biggest deviations is found with Q2 and Q5 between Lucene Fulltext and BM25F, being 73.5% and 61.9% respectively.

Query	Lucene fulltext@10	BM25@10	BM25F@10
Matrix movies (M_Q1)	**0.778**	0.697	0.600
Lord of the rings (M_Q2)	**0.857**	0.790	0.494
Movies by Christopher Nolan (M_Q3)	0.877	**0.941**	0.814
The circus Chaplin (M_Q4)	**0.841**	0.674	0.651
Wachowski directors (M_Q5)	**0.829**	0.535	0.512

Table 6.3: NDCG score for top 10 queries on the movie dataset

An outside factor that can have affected these ranking results is the use of *startsWith* for retrieval of entities in the ImprovedSearch implementation. This was substantial for BM25 and BM25F by retrieving entities fulfilling the startsWith promise, while still being irrelevant in regards to the intent. As an example, for Q1 ("Matrix movies") and Q3 ("Movies by Christopher Nolan"), BM25 and BM25F retrieve movies with the titles "Movin' In" and "In movimento". Additionally, for Q2 ("the circus Chaplin"), they retrieve several movies with the terms "circumstance" or "circulation". The startsWith strategy works well for words composed of two, such as "coronavirus". It was deemed beneficial for the disease dataset but ended up being detrimental for the movie dataset. This begs the question of how comparable these ranking results are between Lucene Fulltext Search and BM25/BM25F as they do not necessarily have a comparable basis of retrieved entities to rank. Differences between BM25 and BM25F, however, are valid regardless.

A key finding in the movie dataset is that Lucene Fulltext performs very consistently for all queries. The difference between its best and worst score only being 12.7%. This is in contrast to the other models' performances in the movie dataset as illustrated in figure 6.3. This is also in contrast to all the models' performances in the disease dataset where all models had much higher deviations in their best and worst scores. Figure 6.3 also shows that BM25 and BM25F to a large degree follow the same trend, with the exception of Q2 where BM25 and Lucene Fulltext behaves similarly.

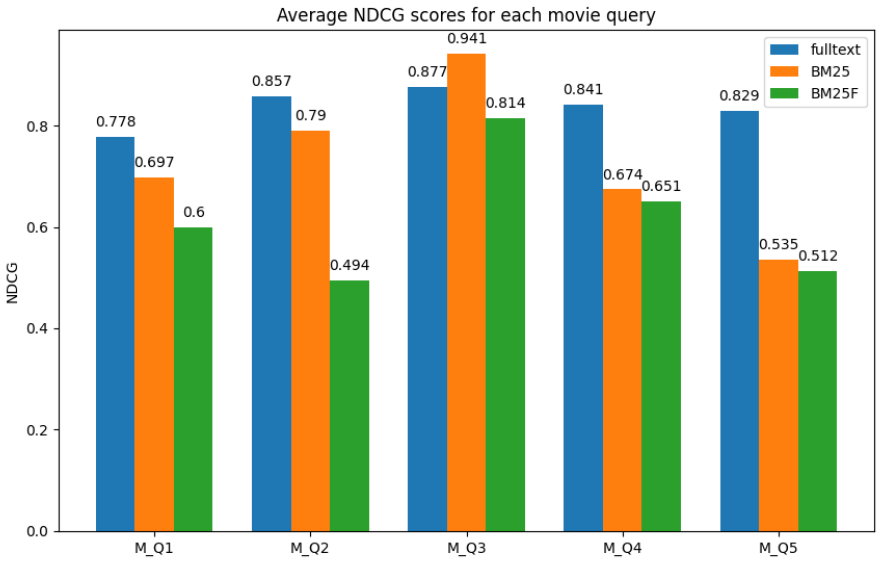


Figure 6.3: Chart overview of the models' NDCG scores @10 on the movie dataset

In contrast to Lucene Fulltext Search, BM25 and BM25F take entity text length into account, prioritizing shorter text lengths compared to the average. This is especially prevalent in this dataset for BM25F looking at text lengths per field with only two fields. This means that an entity with a term match and very short fields (1-3 terms) will score excessively high. This could give an explanation for BM25F's low NDCG score for Q2. There are several movies named "The Ringer", "Ringtone", "Ringmaster", etc. with almost no description getting matched with the query "Lord of the rings". The results show that BM25F might be affected by this. Fine-tuning the parameters using the steps in [40, 41] to perfectly fit the dataset, however, might improve the results.

Interestingly, BM25 is not as affected by this even if it takes entity text length into consideration. Q1 and Q2 are the same types of queries where the intent is to retrieve three specific entities. For BM25, instead of a decreased score from Q1 to Q2 like BM25F, the NDCG score increases similarly to Lucene Fulltext. Since the deviation between Lucene Fulltext and BM25 are marginally different for Q1 and Q2, it is implied that BM25 is not affected with these two queries like BM25F because Lucene Fulltext does not use entity text length in its calculation.

Table 6.4 and figure 6.4 show the results looking at NDCG @5. The most notable difference between the top ten results and top five results is BM25F's performance. We see a significant drop in the NDCG score for Q3 and Q5. Lucene Fulltext performs slightly better and BM25 performs fairly similar with slightly different

scores for certain queries. As a result, there is no clear resemblance in the trends between the models.

Query	Lucene fulltext@5	BM25@5	BM25F@5
Matrix movies (M_Q1)	**0.842**	0.676	0.672
Lord of the rings (M_Q2)	**0.908**	0.853	0.450
Movies by Christopher Nolan (M_Q3)	0.774	**0.922**	0.670
The circus Chaplin (M_Q4)	**0.820**	0.628	0.641
Wachowski directors (M_Q5)	**0.933**	0.559	0.436

Table 6.4: NDCG score for top 5 query results on the movie dataset

These results further illustrate BM25F's struggles with this dataset. Lower scores @5 compared to @10 in the aforementioned queries imply highly relevant entities were ranked below the first five results. The opposite will be true for Lucene Fulltext. A higher score @5 compared to @10 implies the highly relevant entities are ranked in the top five results, and the rankings below are mainly the ones slightly reducing the score. These differences are in contrast to the disease dataset where the difference between the @5 and @10 scores was not considered significant.

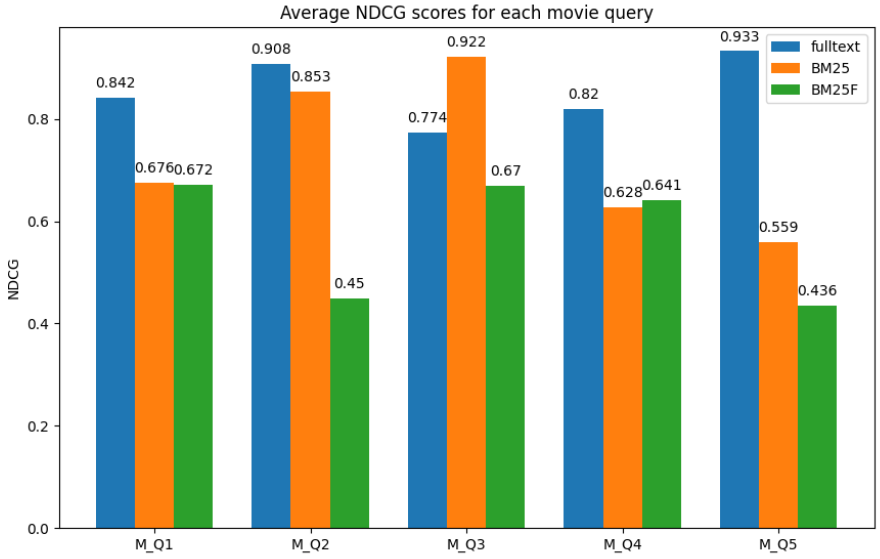


Figure 6.4: Chart overview of the models' NDCG scores @5 on the movie dataset

The chart representation in figure 6.4 no longer shows any similarity in the trend between BM25 and BM25F. BM25 still follows the same trend as it did @10 to a certain degree while BM25F deviates.

Mean Average NDCG and DCG

Table 6.5 shows the mean of the average NDCG scores across all the queries. BM25F performs the best on the disease dataset with an 8.75% better score than Lucene Fulltext search for top ten results. For the movie dataset, Lucene Fulltext performs the best with a 36.2% deviation to the lowest scoring model, B25F. This makes BM25F's overall average performance the worst, while Lucene Fulltext gets the highest mean average NDCG score. However, as the two datasets are vastly different with varying results, it does not seem advantageous to combine these results, but rather look at them individually as a basis for model comparisons.

Ranking model	**NDCG@10**	**NDCG@5**
Disease dataset		
Lucene fulltext	0.789	0.800
BM25	0.831	0.855
BM25F	**0.858**	**0.867**
Movie dataset		
Lucene fulltext	**0.836**	**0.856**
BM25	0.727	0.728
BM25F	0.614	0.574
Both datasets		
Lucene fulltext	**0.813**	**0.828**
BM25	0.779	0.792
BM25F	0.736	0.721

Table 6.5: Mean Average NDCG values

Figure 6.5 and 6.6 present the development of DCG scores as each entity is retrieved. When looking at the disease dataset, each ranking model follows a similar trend, not deviating significantly from each other's path. BM25F is slightly above the others with Lucene Fulltext performing the worst. This is on par with the Mean Average NDCG scores. In contrast, the models on the movie dataset appear more diverse in performance, with Lucene Fulltext being the closest to the ideal, and BM25F scoring lower overall. The models however still follow a similar trend to each other, not deviating further from each other as more results are retrieved.

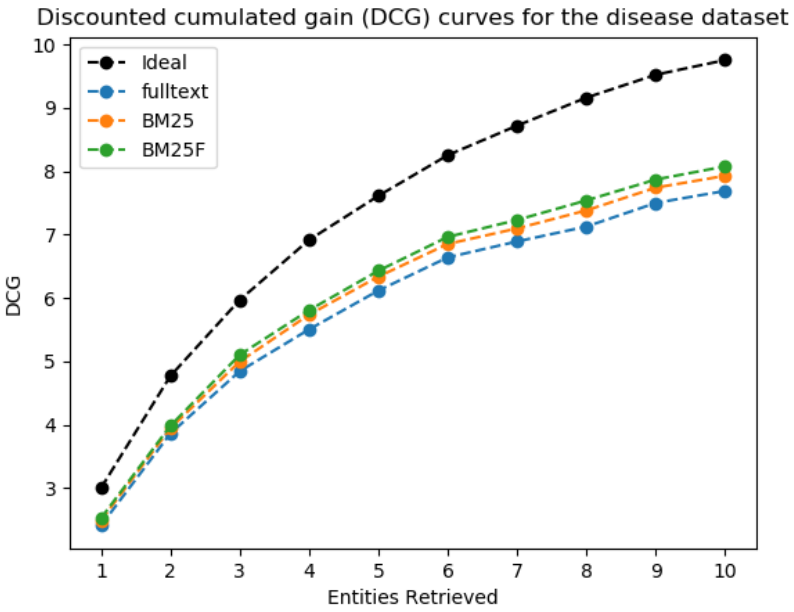


Figure 6.5: DCG development as entities are retrieved for the disease dataset

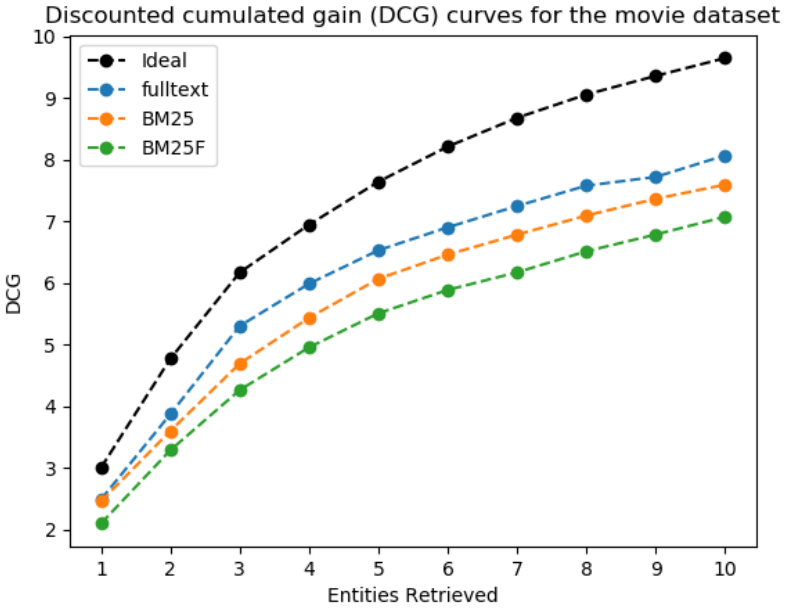


Figure 6.6: DCG development as entities are retrieved for the movie dataset

6.2 Discussion

The heterogeneous results between the two datasets make it challenging to interpret the overall results. Focusing on the disease dataset, the dataset with the most fields and overall content in the fields, BM25F performs the best. There is, however, no indication of the significant improvement over unfielded variants like Tonon *et al.* [9] and Blanco and Vigna [6] reports. Considering the results from the movie dataset, the results deviate even further. Here, BM25 and Lucene Fulltext significantly outperformed BM25F. As previously discussed, several outside factors may affect these results, specifically regarding any comparisons with Lucene Fulltext. However, it is still notable that BM25 outperformed BM25F to this extent.

The results are more in consensus with Pérez-Agüera *et al.* [11]. Analogously with their results, BM25 and BM25F performed better than Lucene Fulltext in an RDF retrieval setting taking the disease dataset into account. However, the margins were small. The poor performance of BM25F on the movie dataset, on the other hand, may in part be explained by the lack of different fields, and the quality of information within those fields. This is in compliance with Pérez-Agüera *et al.* conclusion that states "[...] BM25F is not able to take profit from the semantic information contained in fields with less text" [11, p. 7].

With the differences between the two datasets in mind, it is not surprising the difference is reflected in the results. A challenge with ranking entities in the movie dataset was that several entity descriptions were generic and similar to each other making them less discriminatory. For instance, a typical entity description for a director was "<nationality> film director, screenwriter, and producer". The same applies to movie entities. This is in contrast to the disease dataset where entity descriptions were more unique and discriminatory for each symptom and disease. There is a profusion of semantic information in the fields and relationships in the disease dataset compared to the movie dataset. BM25F is able to take advantage of this. When the information is present to a lesser degree, however, BM25F struggles. Lucene Fulltext and BM25 on the other hand perform much more consistently independently of the semantic information present in the dataset. Thus, BM25F is a better fit for datasets with several fields containing unique and discriminatory terms, while Lucene Fulltext and BM25 are more generic and safe options when there is less insight into the semantic information in the dataset.

A weakness both BM25 and BM25F encountered was an exaggerated prioritization of entities with a smaller amount of text than the average, making entities with more text receive a lower ranking than expected. This is due to the big difference in field lengths in the datasets. In the disease dataset, some entities have

numerous altNames, while others have none. In the movie dataset, very short descriptions resulted in some entities only containing 2-4 terms across all fields. Lucene Fulltext Search does not take text length into account, while BM25 and BM25F will scale term weights based on average length. This is generally a good thing but happened to be somewhat problematic when working with partially incomplete data. Since Wikidata is based on community contributions, some entities are incomplete missing descriptions and altNames resulting in an excessively small amount of text. This can to a certain degree be accounted for by tuning BM25's and BM25F's b and b_c parameters, respectively, for less length normalization. However, this can lead to other challenges. Another way to handle the lack of text is to supplement the query with synonyms. This may expand the number of relevant entities retrieved by additionally matching entities that do not contain exact query terms but share similar meanings. Overall, the incomplete data did not greatly affect the research results, but slightly altered BM25's and BM25F's ranking leading to lower scores.

All the ranking models achieved high NDCG scores and consistent curves. This indicated that they all managed to rank the most relevant results highly. There is no single model that stands out as the best performing from this research. This is because the ranking models' performances are evidently highly dependant on, among other things, the type of dataset, amount and quality of fields, and type of query. The ranking models' strengths and weaknesses are summarized in table6.6.

6.2.1 Other Explored Ranking Models

Before deciding upon ranking models for the survey, preliminary testing for all potential models was conducted. The intention was to get an early indication of their behavior on the datasets in order to choose which models were favorable for further testing in the survey. This was done by querying the data using the models and looking at the ranking results, ranking scores, etc. to compare their behaviors with different types of queries. Each ranking model was implemented in the ImprovedSearch plugin and could easily be included at a later point.

Vector Space Model

Through the preliminary testing, the VSM did not particularly stand out, performing similarly to other models on several queries. As mentioned in section 2.5.3, Lucene Fulltext uses a combination of the Boolean model and the VSM to perform ranking. This, in addition to the overall similar performance, meant the original Vector Space Model was not included in the survey but was still included as part of the plugin.

Strengths	Weaknesses
Lucene fulltext	
Overall consistent performance and behavior	No saturation function across fields
Suitable for a wide range of queries and environments	Not able to tune term relevance saturation and length normalization to fit dataset needs
BM25	
Overall consistent performance	No saturation function across fields
Consistently ranks most relevant results early (@5)	Performs slightly worse when the query consists of very common terms for the dateset such as 'movies, directors, symptom'
Tuneable term relevance saturation and length normalization parameters to fit dataset	Exaggerated prioritization of short entities when there is a large difference between text lengths and length normalization is included
BM25F	
Can perform exceptionally well with a larger number of fields	Not as beneficial with a low number of fields
Saturation function across fields takes advantage of underlying structure when there is quality information in the fields	Struggles when fields are not very unique and discriminatory
Tuneable term relevance saturation and length normalization parameters to fit dataset	Exaggerated prioritization of short entities when there is a large difference between text lengths and the length normalization is not 0

Table 6.6: The observed strengths and weaknesses for each of the ranking models

BM25FF

BM25FF is the new ranking model developed for the purpose of this research and is included as a procedure in the ImprovedSearch plugin. As described in section 4.3.2, the difference between BM25FF and BM25F is the use of local IDF and global IDF calculations respectively.

When performing preliminary testing with BM25FF on the two datasets, a new challenge was revealed. Since the model is based on IDF per field, the model seemed prone to be influenced by the large IDF difference between fields. Originally, it was deemed a good idea since the data was domain-specific. A characteristic of such data is consistency in the structure meaning they share the same fields. However, both the datasets in this thesis consist of a low number of fields, each

with a small amount of text. This meant that in some instances, a term would appear only a few times in certain fields throughout the entire dataset. This would make these terms get excessively high IDF scores. With this in mind, the experimental model would not be able to consistently perform on the same level as Lucene Fulltext, BM25, and BM25F, making the comparison between them skewed.

The hypothesis developed from the preliminary testing with BM25FF was that it could potentially perform more consistently in a different environment. A prerequisite is that the documents/entities in the data collection share the same structure to a large degree. Ideally, the data collection should be larger with more triples consisting of multiple common fields. It would also be beneficial if each field consisted of more text than some of the current fields, which commonly contain between three and five terms. With these ideal conditions, there would be fewer outliers that would drastically affect the IDF-score. This is because there would be more data to take into consideration when calculating IDF per field. To expand upon this hypothesis, further in-depth testing in different environments would be required.

6.2.2 Evaluation of Methods for Analysis

Several papers encountered in the research phase of the thesis, such as [5, 6, 9, 10], utilized NDCG to evaluate ranking models. The Method was also one of the official evaluation metrics for 'SemSearch initiative' and 'TREC-COVID' [39] meaning its widely accepted and used in the research community. The evaluation method gives an indication of how close the ranked results of a model are to an ideal result set. This is done by providing a percentage, such as an NDCG score of 0.8 accounting for an 80% match to the ideal set. In addition to taking the top-k result placements into account, the method can highlight the difference in a single model's top-k scores. An example of this is the notable difference between the top ten and top five NDCG scores for BM25F on the movie dataset. When only considering a single result set from BM25F when querying the movie dataset, some results could indicate that relevant entities are ranked low. With the use of NDCG score, this can be confirmed statistically.

The NDCG scores also help visualize the difference in performance and behavior between the ranking models on the two datasets separately. This is done by representing the data through tables, with indications of the highest scores, and bar charts which help visualize differences in performance. In addition to this, the difference between the highest and lowest performing models is further emphasized through a calculated percentage.

6.2.3 Validity of Research

The use of 'startsWith' for comparisons had a clear impact on the pool of potential entities for the ranking models. This heavily affected BM25 and BM25F for

the movie dataset. As a result, comparing Lucene Fulltext with BM25 and BM25F became more intricate. 'StartsWith' was originally chosen because it was deemed beneficial for the disease dataset as it contains a lot of unique terms and compound words. An unfortunate consequence was its impact on the movie dataset.

One other possible method of comparison is the stricter 'equals'. This was not chosen due to the fear of the exact query formulation being too impactful, missing out on potentially relevant entities. An example of this is compound words such as 'coronavirus', where entities containing only 'corona' would not be deemed a match.

One possible way to modify 'startsWith' for better potential comparisons, would be to reward a higher score to longer matches. This way a term such as 'movies' which would result in matches such as 'moving' and 'movie' would reward 'movie' higher, as it is a longer character match. With this technique, more potential entities would be found than with 'equals', while the less relevant partial matches of 'startsWith' would be less prevalent in the top retrieved results. This was, however, not experimented with in this thesis, as the challenge was not recognized until later in the project.

Another choice that affected the results was the size of the datasets in addition to their number of fields. As Wikidatas endpoint times out if the query takes too long, there was a limit to how many triples and fields were collected for a single query. Originally the number of triples and fields was thought to have little impact as the datasets were domain-specific. This seems to have had a bigger impact than anticipated. If this was apparent from the beginning, another approach could have been chosen, with the use of a different data source or even more domain-specific datasets with a higher number of fields.

Kappa coefficient

Table 6.7 shows the Kappa coefficients for each of the datasets. If the scores are compared to the visualization proposed in section 2.6.5, each result can be categorized in a more understanding way.

- The Quaternary relevancy of the Disease dataset, fall within the *fair agreement*.
- The disease dataset Binary relevancy and the Movie dataset Quaternary relevancy falls within the *moderate agreement*.
- The Movie dataset Binary relevancy falls within the *almost perfect agreement*.

The overall Kappa scores are within an acceptable range, with the binary relevancy on the movie dataset being the highest with a score above 0.8. The goal set for the Kappa scores was an average above 0.4, which was achieved with an average of 0.53. The agreement of the Quaternary relevancy of the Disease dataset was the only one not meeting this standard on its own. This means that the possibility of

some agreement by chance being higher within these survey answers. As the overall scores are within the "moderate agreement" category, the results are deemed reliable, leading to some insight and discovering possible trends. The number of individual answers, at 26, and the number of datasets used in testing, at two, is not large enough to definitively generalize conclusions to all environments.

Relevancy	**Kappa**
Disease dataset	
Quaternary	0.331
Binary	0.537
Movie dataset	
Quaternary	0.434
Binary	0.824

Table 6.7: Kappa coefficient scores for the two datasets

There is a noticeable distinction between the Kappa agreements, as both the Binary scores are significantly higher than their Quaternary counterpart. This most likely stems from uncertainty. The testers were uncertain about how to distinguish between a 0 and 1 score, and a 2 and 3 score.

A possible reason for the differing agreement scores could be due to the *startsWith* comparison between the query terms and possible results. Some testers would rate a result higher when an entire query term was present in the result, while others valued partial matches. This distinction could be what leads to the difference between a rating of 'non-relevant' 0 and a 'pure query match' 1.

When looking at the choice between a 2 or a 3, it could be because the testers interpreted the intent differently. Each individual test subject had to consider how closely the intent should match the result before it is considered an exact match, and what qualifies as being only related to the intent.

Section 4.4.1 presented the example of rating shown to testers before the survey begun. This was to help testers understand how each rating could be interpreted with regard to both the query and the intent of the query. Had this example been more extensive, or several examples were present, the results of the agreement would probably have noticeably increased. The reason this was not done was due to the concern of directing the testers too much affecting the ratings.

Chapter 7

Conclusion

The main goal of this thesis was to research the difference in behavior, and the strengths and weaknesses of IR ranking models when searching in semantic data. More specifically, both fielded ranking models and ranking models traditionally used for unstructured document retrieval were explored for entity retrieval in domain-specific environments.

A platform to import and index semantic data, retrieve and rank entities using different models, and gather relevancy assessments for evaluation was developed. This platform was used to conduct a user study evaluating BM25, BM25F, and Lucene Fulltext Search in two different domain-specific environments. With previous research in mind, the findings highlight how fielded ranking models take advantage of the underlying structure of the data, but struggles when the contents within the structure are not unique and discriminatory. On the other hand, traditional ranking models perform more consistently regardless of the quality of the information in entity fields and relations.

7.1 Contributions

During this thesis, the focus has been to answer the research questions specified in Chapter 1 and 3. Here, each question will be shortly answered by explaining how they were approached and what results were found.

RQ1: How do IR ranking models perform for users searching in semantic data?
The research conducted in this thesis shows that all the tested ranking models show promising results for users searching in semantic data. BM25F performs the best on the disease dataset with a mean average NDCG score of 0.858 and Lucene Fulltext performs the best on the movie dataset with a mean average NDCG score of 0.836.

RQ1.1: What are the strengths and weaknesses of different IR ranking models?

To begin with, approaches from related research and their results were taken into account. This provided preliminary insight into which ranking models could be beneficial in different circumstances. More detailed advantages and disadvantages were discovered through preliminary testing and a more extensive user study. This made it possible to examine the ranking models' behavior more closely in two different environments with several types of queries. An overview of the strengths and weaknesses was presented in table 6.6.

RQ1.2: How do the ranking models behave in an entity retrieval setting?
With the proposed approach to indexing, even if each entity contains a lot less text compared to unstructured documents, most or all of the relevant entities appear in the top ten results. The only model that encountered some notable challenges was BM25F on the movie dataset. The results can indicate that the lack of text might not be as damning as previously thought as long as all the text is utilized to its full potential. A way to further take advantage of the full potential could be to expand upon the existing content by supplementing the query with synonyms.

RQ1.3: How do fielded ranking models' performances compare?
Given the research results, fielded ranking models', like BM25F, are highly dependant on the quality and discriminatory factor of the information in the fields. Fielded ranking models have the potential of outperforming their unstructured counterparts. This happens if the dataset has several fields with descriptive and unique text. The margins of improvement are, however, small. When the quality of the information in the fields is low, BM25F gets significantly outperformed. This is because there is no underlying structure to take advantage of. These results are consistent with previous research.

RQ2: What methods are suitable for evaluating entity retrieval? Since entity retrieval and search in semantic data is a currently growing field, there is a lack of standardized test sets for different environments. A user study gathering users' relevancy assessments in a questionnaire is a good alternative to automatic testing based on test sets. The methods chosen for evaluation must reflect the spectrum of perceived relevancy of retrieved entities and that it is not simply binary. Thus, DCG and NDCG are suitable measures to evaluate models' performances, behavior, as well as strengths and weaknesses. However, this requires the researchers to be familiar with the dataset in order to calculate the ideal DCG.

7.2 Limitations

Throughout the thesis, a number of potential limitations to the research have come to light.

- Implementation of retrieval based on the StartsWith function in the proposed Neo4j plugin impacted the pool of potential entities to rank. BM25

and BM25F were negatively affected by this in the movie dataset by retrieving entities that fulfilled the StartsWith promise while still being regarded as irrelevant to the query and intent.
- The number of RDF-triples was limited due to the Wikidata SPARQL endpoint timing out after 60 seconds. This impacted both the number of entities and the number of entity fields included in the datasets.
- As each test subject in the user-evaluation had to rate every single result, the time to complete the survey rapidly increased with the inclusion of new models and datasets. With this in mind, the number of models was kept to three, and the number of datasets two.
- Since convenience sampling was used as the method for data gathering, the survey participants might not necessarily be representative of the entire population.

These limitations can have affected the results in some way, making the detailed evaluation metrics less definitive. However, the results of the thesis still provide another needed perspective to the ambiguous results regarding which ranking model performs the best with keyword search in semantic data.

7.3 Future work

Based on the limitations in addition to the results described in the previous chapter, some research to further examine the ranking models' behavior in different environments would be beneficial.

One interesting aspect is how the dataset affects the performance of each ranking model. Based on the findings in this thesis, BM25F seems to be the model that is the most affected by the low number of fields and lack of unique terms. To further expand upon this theory, research regarding the ranking models' performance in other environments should be performed. Some aspects of the dataset that could be changed are the number of fields, amount of text, number of unique terms in each field, etc.

Larger datasets with a consistent structure are needed to fully explore the potential of BM25FF. A similar study comparing BM25FF with BM25F and other models using such datasets can be an interesting extension to this research. This can be beneficial as the further exploration of models can lead to a better understanding of when to use each model.

In this thesis, the standard parameter values for each ranking model were used. To expand the comprehension of ranking models' behavior, the tweaking of parameters should be examined more in-depth. This would give a better understanding of which values are beneficial in each environment, and how each change impacts the ranking.

Bibliography

[1] C. Manning, P. Raghavan and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[2] J. Pound, P. Mika and H. Zaragoza, 'Ad-hoc object retrieval in the web of data,' Jan. 2010, pp. 771–780. DOI: 10.1145/1772690.1772769.

[3] J. Guo, G. Xu, X. Cheng and H. Li, 'Named entity recognition in query,' in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09, Boston, MA, USA: Association for Computing Machinery, 2009, pp. 267–274, ISBN: 9781605584836. DOI: 10.1145/1571941.1571989. [Online]. Available: https://doi.org/10.1145/1571941.1571989.

[4] T. Lin, P. Pantel, M. Gamon, A. Kannan and A. Fuxman, 'Active objects: Actions for entity-centric search,' in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12, Lyon, France: Association for Computing Machinery, 2012, pp. 589–598, ISBN: 9781450312295. DOI: 10.1145/2187836.2187916. [Online]. Available: https://doi.org/10.1145/2187836.2187916.

[5] S. Elbassuoni and R. Blanco, 'Keyword search over rdf graphs,' in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, ser. CIKM '11, Glasgow, Scotland, UK: Association for Computing Machinery, 2011, pp. 237–242, ISBN: 9781450307178. DOI: 10.1145/2063576.2063615. [Online]. Available: https://doi.org/10.1145/2063576.2063615.

[6] R. Blanco, P. Mika and S. Vigna, 'Effective and efficient entity search in rdf data,' in *The Semantic Web – ISWC 2011*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy and E. Blomqvist, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 83–97.

[7] R. Blanco, P. Mika and H. Zaragoza, 'Entity search track submission by yahoo! research barcelona,' SemSearch, 2010.

[8] N. Craswell, H. Zaragoza and S. Robertson, 'Microsoft cambridge at trec 14: Enterprise track,' in *TREC*, 2005.

[9] A. Tonon, G. Demartini and P. Cudre-Mauroux, 'Combining inverted indices and structured search for ad-hoc object retrieval,' *SIGIR'12 - Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, Aug. 2012. DOI: 10.1145/2348283.2348304.

[10] A. Esteva, A. Kale, R. Paulus, K. Hashimoto, W. Yin, D. Radev and R. Socher, *Co-search: Covid-19 information retrieval with semantic search, question answering, and abstractive summarization*, 2020. arXiv: 2006.09595 [cs.IR].

[11] J. R. Pérez-Agüera, J. Arroyo, J. Greenberg, J. P. Iglesias and V. Fresno, 'Using bm25f for semantic search,' in *Proceedings of the 3rd International Semantic Search Workshop*, ser. SEMSEARCH '10, Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, ISBN: 9781450301305. DOI: 10.1145/1863879.1863881. [Online]. Available: https://doi.org/10.1145/1863879.1863881.

[12] Z. Nie, Y. Ma, S. Shi, J. Wen and W. Ma., 'Web object retrieval,' WWW, 2007.

[13] P. Castells, M. Fernández and D. Vallet, 'An adaptation of the vector-space model for ontology-based information retrieval,' *IEE TRANSACTIONS ON KNOWLEDGE DATA ENGINEERING*, vol. 19, no. 2, 2007.

[14] T. Tran, H. Wang, S. Rudolph and P. Cimiano, 'Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data,' in *2009 IEEE 25th International Conference on Data Engineering*, 2009, pp. 405–416. DOI: 10.1109/ICDE.2009.119.

[15] P. Mika, 'Distributed indexing for semantic search,' in *Proceedings of the 3rd International Semantic Search Workshop*, ser. SEMSEARCH '10, Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, ISBN: 9781450301305. DOI: 10.1145/1863879.1863882. [Online]. Available: https://doi.org/10.1145/1863879.1863882.

[16] C. C. Kuhlthau, 'Information search process,' *Hong Kong, China*, vol. 7, p. 226, 2005.

[17] K. Balog, *Entity-Oriented Search*. Springer International Publishing, 2018.

[18] K. Balog, 'Semistructured data search,' in. Jan. 2014, pp. 74–96, ISBN: 9783642547973. DOI: 10.1007/978-3-642-54798-0_4.

[19] S. P. Gardner, 'Ontologies and semantic data integration,' *Drug Discovery Today*, vol. 10, no. 14, pp. 1001–1007, 2005, ISSN: 1359-6446. DOI: https://doi.org/10.1016/S1359-6446(05)03504-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S135964460503504X.

[20] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann and I. Horrocks, 'The semantic web: The roles of xml and rdf,' *IEEE Internet Computing*, vol. 4, no. 5, pp. 63–73, 2000. DOI: 10.1109/4236.877487.

[21] T. Berners-Lee, F. R.T. and L. Masinter, 'Uniform resource identifier (uri): Generic syntax,' Apr. 2002.

[22] J. Guia, V. G. Soares and J. Bernardino, 'Graph databases: Neo4j analysis,' in *ICEIS*, 2017.

[23] B. DuCharme, *Learning SPARQL*. O'Reilly Media, Inc., 2011, ISBN: 1449306594.

[24] R. Reinanda, E. Meij and M. de Rijke, 'Knowledge graphs: An information retrieval perspective,' *Foundations and Trends® in Information Retrieval*, vol. 14, no. 4, pp. 289–444, 2020, ISSN: 1554-0669. DOI: 10.1561/1500000063. [Online]. Available: http://dx.doi.org/10.1561/1500000063.

[25] K. Balog, 'Entity retrieval,' in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. New York, NY: Springer New York, 2017, pp. 1–6, ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_80724-1. [Online]. Available: https://doi.org/10.1007/978-1-4899-7993-3_80724-1.

[26] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, 2nd ed. Pearson Education Limited, 2011.

[27] C. Galvez, F. Moya-Anegon and V. Herrero-Solana, 'Term conflation methods in information retrieval: Non-linguistic and linguistic approaches,' *Journal of Documentation*, vol. 61, Aug. 2005. DOI: 10.1108/00220410510607507.

[28] W. Frakes and Baeza-Yates, *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, 1992.

[29] D. Knuth, *The Art Of Computer Programming, vol. 3: Sorting And Searching*. Addison-Wesley, 1973, pp. 391–392.

[30] G. Salton and C. Buckley, 'Term-weighting approaches in automatic text retrieval,' *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988, ISSN: 0306-4573. DOI: https://doi.org/10.1016/0306-4573(88)90021-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0306457388900210.

[31] I. H. Witten, A. Moffat and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition*. Morgan Kaufmann, 1999, ISBN: 1-55860-570-3. [Online]. Available: http://www.cs.mu.oz.au/mg/.

[32] S. Robertson, H. Zaragoza and M. Taylor, *Simple bm25 extension to multiple weighted fields*, 2004.

[33] K. Järvelin and J. Kekäläinen, 'Cumulated gain-based evaluation of ir techniques,' *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, Oct. 2002, ISSN: 1046-8188. DOI: 10.1145/582415.582418. [Online]. Available: https://doi.org/10.1145/582415.582418.

[34] K. Järvelin and J. Kekäläinen, 'Ir evaluation methods for retrieving highly relevant documents,' in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '00, Athens, Greece: Association for Computing Machinery, 2000, pp. 41–48, ISBN: 1581132263. DOI: 10.1145/345508.345545. [Online]. Available: https://doi.org/10.1145/345508.345545.

[35] J. Cohen, 'A coefficient of agreement for nominal scales,' *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, Apr. 1960, ISSN: 0013-1644. DOI: 10.1177/001316446002000104. [Online]. Available: https://doi.org/10.1177/001316446002000104.

[36] J. L. Fleiss, 'Measuring nominal scale agreement among many raters,' vol. 76, no. 5, pp. 378–382, 1971. [Online]. Available: https://doi.org/10.1037/h0031619.

[37] A. Viera and J. Garrett, 'Understanding interobserver agreement: The kappa statistic,' vol. 37, no. 5, pp. 360–363, 2005.

[38] F. Song and W. B. Croft, 'A general language model for information retrieval,' in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, ser. CIKM '99, Kansas City, Missouri, USA: Association for Computing Machinery, 1999, pp. 316–321, ISBN: 1581131461. DOI: 10.1145/319950.320022. [Online]. Available: https://doi.org/10.1145/319950.320022.

[39] K. Roberts, T. Alam, S. Bedrick, D. Demner-Fushman, K. Lo, I. Soboroff, E. Voorhees, L. L. Wang and W. R. Hersh, *Searching for scientific evidence in a pandemic: An overview of trec-covid*, 2021. arXiv: 2104.09632 [cs.IR].

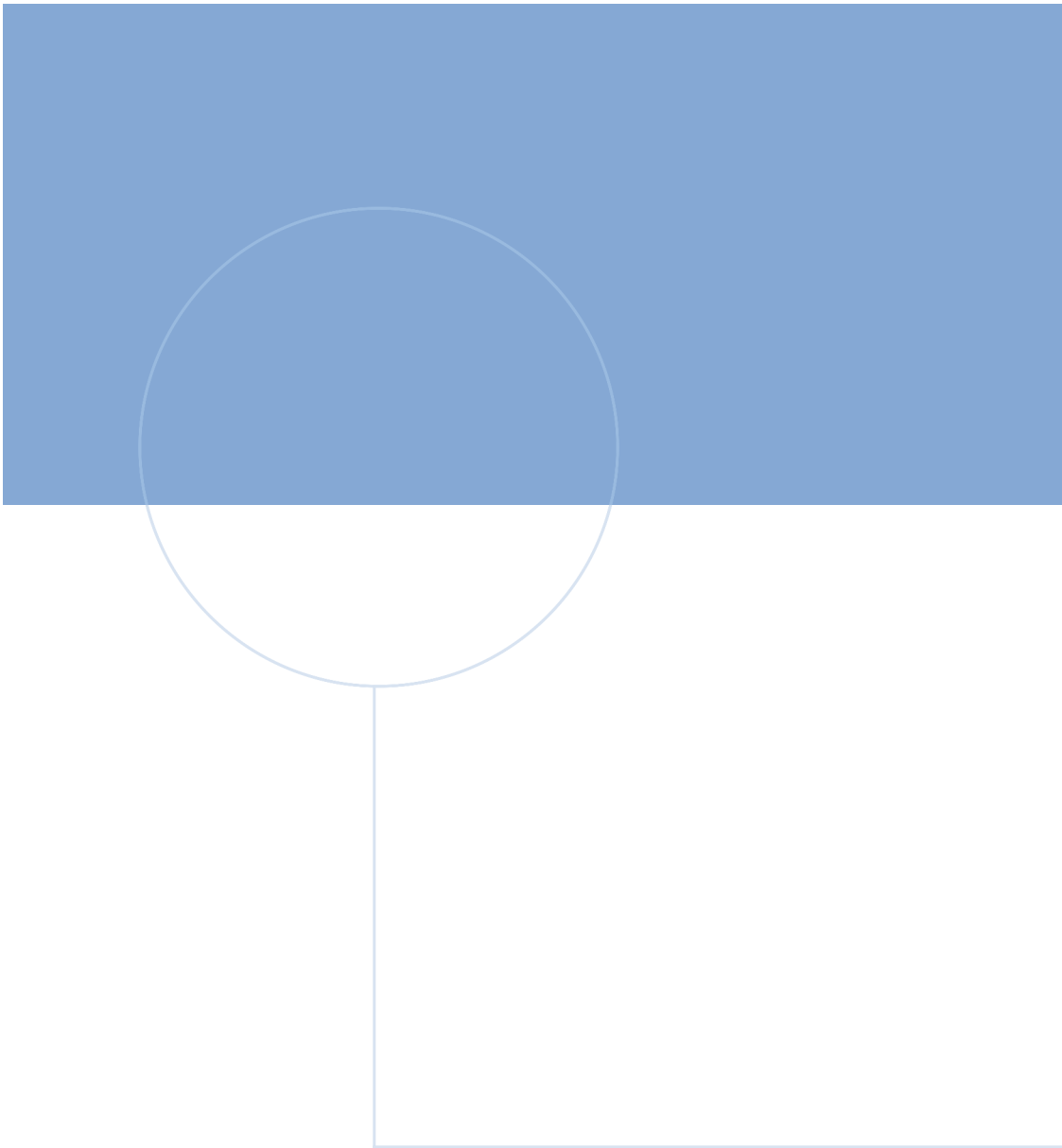
[40] B. HE and I. Ounis, 'A study of parameter tuning for term frequency normalization,' in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03, New Orleans, LA, USA: Association for Computing Machinery, 2003, pp. 10–16, ISBN: 1581137230. DOI: 10.1145/956863.956867. [Online]. Available: https://doi.org/10.1145/956863.956867.

[41] B. He and I. Ounis, 'Term frequency normalisation tuning for bm25 and dfr models,' in *Proceedings of the 27th European Conference on Advances in Information Retrieval Research*, ser. ECIR'05, Santiago de Compostela, Spain: Springer-Verlag, 2005, pp. 200–214, ISBN: 3540252959. DOI: 10.1007/978-3-540-31865-1_15. [Online]. Available: https://doi.org/10.1007/978-3-540-31865-1_15.

[42] Y. Wang, L. Wang, Y. Li, D. He, T.-Y. Liu and W. Chen, 'A theoretical analysis of ndcg ranking measures,' 2013.

[43] D. Vrandečić and M. Krötzsch, 'Wikidata: A free collaborative knowledgebase,' *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Sep. 2014, ISSN: 0001-0782. DOI: 10.1145/2629489. [Online]. Available: https://doi.org/10.1145/2629489.

[44] B. J. Oates, *Researching Information Systems and Computing*. Sage Publications Ltd., 2006, ISBN: 1412902231.



NTNU

Kunnskap for en bedre verden