Ranking of Keyword-Based Search Query Results in Knowledge Graphs

B A C H E L O R T H E S I S
A thesis submitted for the degree of
Bachelor of Science (B. Sc.)
in the degree program
Angewandte Informatik

FRIEDRICH-SCHILLER-UNIVERSITY JENA
Faculty for Mathematics and Computer Science

submitted by Antonio Noack
born 1st July 1997 in Jena, Germany
Supervisors: Sirko Schindler, Leila Feddoul,
and Birgitta König-Ries
Jena, 17th August 2020

Abstract

In this bachelor thesis, we propose a ranking based on learning to rank to order results of queries over knowledge graphs. Queries consist of basic graph patterns with variable entities and the results are bindings of actual entities to the variables. With these bindings, a subgraph of the knowledge graph can be reconstructed from the query.

We discuss existing approaches from information retrieval (e.g. TF-IDF) and show how to adapt them to a graph environment. The main challenge is that information retrieval methods usually rely on rich text documents, which are often missing in knowledge graphs. The extensive text information existing in those graphs is mostly represented only by rather short literal values. To address the issue, we propose a method to extract string data from knowledge graphs in order to make information retrieval methods applicable. In addition, we explore different graph metrics (e.g. PageRank) and their possible application to a ranking scenario.

Moreover, we suggest to use a neural network to create a score for ranking from a series of features. The proposed features are based on information retrieval techniques, our string extraction method and graph metrics.

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit einem Ranking, das auf Learning To Rank (LTR), also Maschinellem Lernen basiert, um Ergebnisse einer Anfrage auf Wissensgraphen nach Relevanz zu ordnen. Die Anfragen bestehen aus Aussagen mit unbekannten Entitäten und die Ergebnisse sind Entitäten, durch die die Aussagen wahr werden. Mit diesen Ergebnissen kann man einen Subgraphen des Wissengraphens rekonstruieren.

Für diese Sortierung untersuchen wir Methoden aus dem Bereich ‘Information Retrieval’ (z.B. TF-IDF). Die Hauptherausforderung der Methoden aus diesem Bereich ist, dass diese Methoden auf umfangreichen Textdokumenten arbeiten, die oft nicht in Wissensgraphen vorkommen. Stattdessen enthalten sie nur recht kurze Literale. Wir schlagen daher zur Extraktion der String-Daten aus dem Wissensgraphen eine Methode vor, damit wir Verfahren aus dem ‘Information Retrieval’-Bereich nutzen können. Des Weiteren erkunden wir Graph-Metriken (z.B. PageRank) und ihre Nutzbarkeit für ein Ranking.

Für das Ranking-System schlagen wir eine Reihe von Features vor, die das LTR nutzt, um einen Score zu berechnen, der dann zum Ranking verwendet wird. Unsere vorgeschlagenen Features basieren auf den vorher untersuchten Gebieten und Methoden.

Contents

Chapter 1

Introduction

Searching though data collections using keywords is part of a lot of systems, with the most popular among them probably being Google search[1] [Net20]. Other popular systems include Amazon[2], YouTube[3], and Bing[4].

Users enter a few keywords and the system returns a ranked list of results deemed relevant. The order of the results is important: the most relevant results should be at the start of the list, and irrelevant entries at the end [Dea19]. Unfortunately, the task of ranking documents according to the query keywords is not a trivial task.

1.1 Background on Keyword Search

Search is traditionally performed on collections of documents containing text. In the first step, documents, that match the search query are retrieved, then they are ranked in order of relevance. Searching over all documents for every query is very time-consuming, so 'inverted indices' [Jai] are created: First all documents are read. For each word appearing in the documents, excluding common so-called stop words, a list is created. The lists will contain a reference to the documents, where the considered word appears. Additionally, the word frequency can be added for later ranking.

When searching for documents, the lists for all query keywords are retrieved. Documents are then ranked based on the number of occurrences in those lists. Only documents, which appear in the lists of all keywords are displayed. Inverted Indices created using this simple approach have the huge disadvantage, that documents, that use only synonyms of the entered keywords, may not be discovered. For example a document might use the word 'father' instead of 'dad' and thus will not be retrieved in response to the query 'dad'. This can be improved by using lists of synonyms, and by associating the group of words with similar meaning to one representative term. For simplicity, the representative term can be member. For example, 'father' can be chosen as the representative term for 'father' and 'dad'. Sub-classes (hyponyms) and super-classes (hypernyms) are similar in meaning as well, and could be used too, like in [LK16]. The document lists of all words are enriched by all documents, which contain synonyms to that word. This is done

[1]https://google.com
[2]https://amazon.com
[3]https://youtube.com
[4]https://bing.com

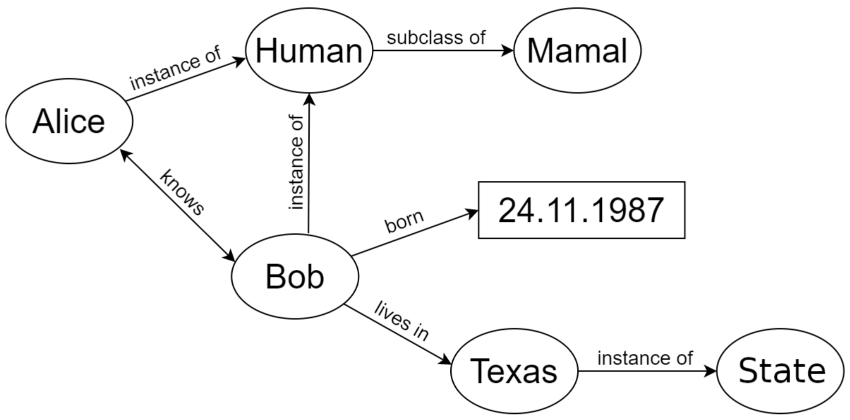

Figure 1.1: A knowledge graph example

for example by Amazon [SCP16]. A disadvantage of adding all synonyms is, that being synonyms does not necessarily imply relevance to the query since the context of the word can be important for its meaning. Often that context is defined by the other keywords of the query. For example, ‘pause’ and ‘shattering’ are synonyms for the keyword ‘break’. But ‘pause’ is very different from ‘shattering’. When searching for ‘break from school’, documents containing the word ‘shattering’ are likely not relevant.

When the lists of documents are expanded with synonyms, false positives are introduced. However, when not adding them, relevant documents may be missing. To find a proper balance, semantic approaches are required to extract the context of a query. Knowledge Graphs (KGs) can help with that.

1.2 Knowledge Graphs

Information has not only become widely available in text form, but also directly as machine interpretable concepts, objects (instances or literals) and their relations. The concepts and objects are called entities. When relations are interpreted as connecting edges between one entity and another, the knowledge is effectively a graph.

An example is given in Figure 1.1. It uses the entities ‘Alice’, ‘Bob’, ‘Human’, ‘Mammal’, ‘Texas’ and ‘State’, the literal value ‘24.11.1987’, and the relation types ‘instance of’, ‘subclass of’, ‘knows’, ‘born’ and ‘lives in’.

The arrows symbolise the flow of a typical English sentence, and indicate that those relations are directed. Entities, the subjects and objects of a sentence, are the nodes of the graph, while the predicates are the directed edges. With this representation, related concepts are closer in the graph than unrelated concepts.

For a general purpose search engine, that used KGs to semantically enhance search, large-scale and cross-domain KGs are needed. A popular open KG is DBpedia[5] [LIJ$^+$14], which extracts knowledge from Wikipedia[6] articles. Another example is Wikidata[7] [VK14], which can be collaboratively edited by humans and machines. Its data is intended to be used directly in Wikipedia, so that an update of a single fact is consistently represented

[5]https://wiki.dbpedia.org/
[6]https://www.wikipedia.org/
[7]https://www.wikidata.org/

across all pages in Wikipedia. The data from both projects can be downloaded and used for free. But those are only a few of many public projects with shared knowledge: the Linked Open Data Cloud[8] gives an overview of more datasets.

There are two popular data formats to represent KGs: Resource Description Framework (RDF) [Swi99] and Labelled Property Graphs (LPGs). In RDF, the vertices and edges of the graph are resources with Internationalized Resource Identifiers (IRIs). Vertices also can be datatyped literals (e.g. strings, numbers, dates) or blank nodes. IRIs are Uniform Resource Identifiers (URIs), but with the Universal Coded Character Set instead of US-ASCII [FUS], so special characters like umlauts are possible. In LPGs all vertices and edges are identified by their ID [Bar17]. They have attribute lists instead of literal neighbour nodes.

Both approaches have their advantages and disadvantages: Querying paths from one node to another, or calculating the number of paths is easier with LPG, because the graph is smaller. However, LPGs do not support inheritance. Additionally, there are no restrictions for the attributes, which makes inference of new knowledge with logical reasoning more complicated. Using IDs instead of IRIs makes merging datasets tedious, because entities may have the same ID, while being different, or be the same, while having different IDs. A work-around is to link the datasets together (e.g. by including a prefix as a source dataset identifier), and then using nodes from multiple graphs inside one system.

SPARQL (*SPARQL* Protocol and Query Language) [SH13] is a popular query language for RDF, and can be used to query both DBpedia[9] and Wikidata[10].

Similar to SQL, a popular query language for databases, a select-query is a set of constraints on the result. Often they are triples, with a couple of unknowns. The query processor then tries to fill in unknown nodes using the KG, such that all constraints are fulfilled. The result of such a query is then a list of all possible matches between the requested structure and the KG at hand.

To retrieve all pairs of entities knowing each other with one of them living in Texas from the KG in Figure 1.1, we can use the query in Listing 1.1.

```
SELECT ?a ?b
WHERE {
  ?a ex:knows ?b .
  ?b ex:knows ?a .
  ?b ex:livesIn ex:Texas .
}
```

Listing 1.1: Example Query.

In this case, there will be only one result. '?a' will be bound to 'Alice' and '?b' to 'Bob'. The corresponding subgraph is presented in Figure 1.2: it is the same three triples from the query, just '?a' and '?b' were replaced with 'Alice' and 'Bob' respectively.

[8]https://lod-cloud.net/
[9]https://dbpedia.org/sparql
[10]https://query.wikidata.org/

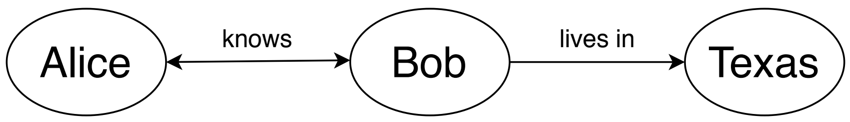

Figure 1.2: The result subgraph

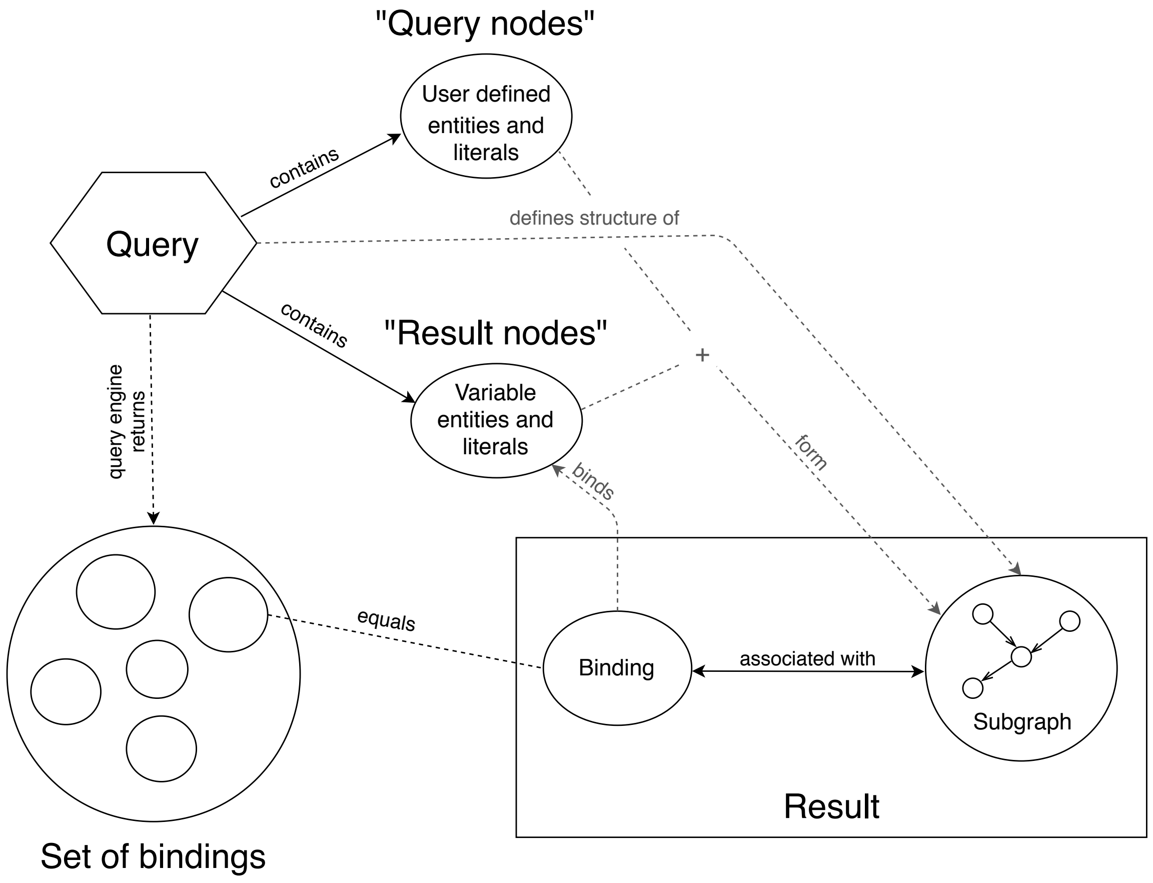

Figure 1.3: Overall overview of the problem

Each result is called a binding. A query can result in subgraphs of different shapes, because SPARQL allows for optional restrictions and logical connections (*and*, *or* and *not* operators) of constraints, so not every single constraint needs to be fulfilled for each binding.

1.3 Naming Conventions

Figure 1.3 gives an overview of the concepts and their relations used throughout this thesis. We call each binding and its associated subgraph 'result'. For simplicity, we call the explicitly defined entities and literals 'query nodes', because they are nodes defined in the query. We call the variable entities and literals 'result nodes', because they are KG-nodes specific to the result. All nodes in between are also considered query nodes to simplify our features[11].

[11]creating a third group would be more complex

We can distinguish between global and local ranking. Global ranking is query independent, while local ranking depends on the issued query. Global rankings face the issue that a result is only rarely relevant to all queries.

In formulas, we use arrows on $\overrightarrow{vectors}$ and hats on $\widehat{matrices}$ to clarify the type of variables. We define the average as the mean value. It is equal to the sum of the values $V = (v_1...v_n)$ divided by the number of values (see Equation 1.1).

$$average\ value = \frac{1}{\sum_{v_i \in V} 1} \sum_{v_i \in V} v_i \tag{1.1}$$

1.4 Thesis Scope

While SPARQL queries effectively deliver a set of subgraphs for the query, they do not automatically rank them in any meaningful way. In this thesis, we first discuss different ranking strategies from the information retrieval (IR) and graph communities. Pure IR techniques often work only on text, while graph metrics ignore the labels and descriptions of the graph's nodes and edges. Based on this comparison, we then propose a concept to rank subgraph-shaped query results over KGs. Typically, entities are described using their label, a description, and additionally contain other string properties. All this text information can be used in the ranking.

In summary, we take a look at existing IR and graph-based approaches. Using these, we propose a ranking method based on LTR (see Section 2.3.3) to rank sets of result subgraphs.

Chapter 2

Related Work

In this chapter, we will present popular ranking techniques, give some examples of how multiple scores can be merged into one, and talk about related approaches. We use these as a basis for our proposed ranking system.

2.1 Ranking Techniques

Information retrieval (IR) techniques in general have existed for quite some time now, while IR techniques working on KGs have become a popular topic in data science just a few years ago (e.g. [LXSL18, DKM18, SGCR19]).

2.1.1 TF-IDF

Term Frequency - Inverse Document Frequency (TF-IDF) [JON72] is one of the most popular IR techniques. It belongs to the Bag of Words (BOW) approaches. A Bag of Words (BOW) is a multiset, so it effectively counts, how often which words appeared. There are IR approaches, which only work on BOWs and ignore the sentence structure. Other approaches may additionally use the sentence structure to get a better result.

TF-IDF tries to measure how well a group of terms (e.g. a keyword query) matches a given document. Each term is assigned a score depending on how often it occurs in a document, weighted by the importance of the term. Then the scores are added together to form a score for the document with respect to the query. The first part of this score, the term frequency (TF), tries to capture how often a term appears in the document. The second part, the Inverse Document Frequency (IDF) measures how significant the term is in the whole document collection. It gives for example stop words like 'the' and 'a' only a small score, because they are more common words, while more rare terms are given more importance.

The TF reflects how often a term appears in the document. It can be the raw value, of how often a term appears in the document, or relative to the most frequent term [MRS09]. The relative variant is shown in Equation 2.1. Using no normalization can give long documents an unfair advantage[1]. Furthermore, values of TF would not be limited to

[1]abuse of the system could be imagined, where multiple copies of the same document are concatenated to get a higher score

a specific range, which can become an issue. Terms can be N-grams (chains of N words), or complex names like ‘Queen Elizabeth II’, but order and grammar of terms are ignored. TF-IDF is calculated as the multiplication of TF and IDF (Equation 2.2), where IDF is given by the logarithm of the inverse fraction of the documents that contain the term. Using the logarithm is similar to an entropy of the terms over all documents. In the equations, q_i is a term from the query, D is the document, p is a term from D, for which the score shall be calculated, and $q_1..q_n$ is the whole group of terms of the query. Note that the resulting value for TF-IDF has no upper bound.

$$TF(q_i, D) = \frac{number\ of\ occurances\ of\ q_i\ in\ D}{\max_{\forall p \in D}(number\ of\ occurances\ of\ p\ in\ D)} \tag{2.1}$$

$$IDF(q_i) = ln\left(\frac{|\forall D|}{|\forall D, q_i \in D|}\right) \tag{2.2}$$

$$TFIDF(q_i, D) = TF(q_i, D) \cdot IDF(q_i) \tag{2.3}$$

$$TFIDF(q_1..q_n, D) = \sum_{i=1}^{n} TFIDF(q_i, D) \tag{2.4}$$

Okapi BM25

Okapi Best Match 25 (BM25) is a ranking function analogous to TF-IDF. BM25 is effectively just another way to normalize the TF-term. However, it is said to give long documents too little weight [LZ11]. The BM25 score of a document D with respect to the keywords $q_1..q_n$ is defined in Equation 2.6.

$$relative\ document\ length = \frac{document\ length}{average\ document\ length} \tag{2.5}$$

$$BM25(D, q_1...q_n) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{TF(q_i) \cdot (k_1 + 1)}{TF(q_i) + k_1 \cdot (1 - b + b \cdot relative\ document\ length)} \tag{2.6}$$

The constant k_1 is usually set to a value between 1.2 and 2.0, and b to 0.75 [Con18]. The parameter b controls the influence of the relative document length, and k_1 the smoothness of the normalization curve. The smaller k_1 is, the sharper the curve becomes when varying TF.

2.1.2 Knowledge Graph Measures

One of the simplest rankings would be to rank subgraphs higher, which have shorter paths from all variables to all explicitly specified entities in the query. Two similar metrics are radius and diameter of the subgraph. The radius and diameter of graph G are defined respectively in Equation 2.7 and 2.8 [Bha], where N are the nodes of the graph G, and the distance from any node $n1$ to another node $n2$ is defined as the length of the shortest path between them.

$$Radius(G) = \min_{n_1 \in N} \max_{n_2 \in N} Distance(n_1, n_2) \tag{2.7}$$
$$Diameter(G) = \max_{n_1 \in N} \max_{n_2 \in N} Distance(n_1, n_2) \tag{2.8}$$

The radius is the longest distance between the center node and the graph nodes, where the center node is the node closest to all others (individually). The diameter is the largest distance between two nodes in the graph [Bha].

There are other measures on vertices and edges, called centralities. They typically are used to find the most important nodes of a graph. In-Degree- and Out-Degree-Centrality are two simple centralities, where the value for node is the number of incoming/out-going edges in the KG. PageRank is another centrality measure.

2.1.3 PageRank

In [Pag98] the PageRank algorithm is proposed, which helped Google to become so successful. The computation of a score for a document incorporates all other documents as well. PageRank is modelled as the average number of users on a node (e.g. website), while moving from node to node via links, or randomly when 'bored'.

The random jumps are often presented as a vector, where each element represents a node, and the value is the chance the current node is chosen randomly. This vector is called the preference vector.
PageRank follows three basic rules:

- If there are no links to other nodes, users will jump to a random node in the graph.
- Users will follow a link to another node with a defined probability $1 - p$.
- Otherwise they traverse to a random node or their preferences in the graph.

The formula for PageRank is given by Equation 2.9, where p is the probability of a random jump, $\overrightarrow{PR}$ is the vector of PageRanks $\overrightarrow{PR} = (PR_1, ..., PR_n)$, where PR_i is the PageRank of node i. $\overrightarrow{P}$ is the preference vector, and $\widehat{A}$ is the adjacency matrix, where the entries are weighted to be probabilities for a user moving from this node to another randomly. The computation of the number of users on a node, called the PageRank, is done iteratively. It usually converges in $\mathcal{O}(log(Number\ of\ Nodes))$ iterations.

$$\overrightarrow{PR} = \widehat{A} \cdot \overrightarrow{PR} \cdot (1 - p) + p \cdot \overrightarrow{P} \tag{2.9}$$
$$PR_i = \text{PageRank of node i} \tag{2.10}$$
$$\widehat{A}_{ij} = \begin{cases} \frac{1}{\text{number of outgoing edges from node i}} & \text{edge from node i to node j exists} \\ 0 & \text{else} \end{cases} \tag{2.11}$$

The algorithm is famous for several reasons: it is rather simple, requires mostly sequential storage accesses and it allows great customizability by changing the preferences for the random jumps. If $p = 0$, $\overrightarrow{PR}$ converges to an Eigenvector of $\widehat{A}$.

In the context of web search the score relies on links from other important pages, like search engines and catalogues, social networks, wikis, or blogs. This symbolic authority is hard to falsify, and it was rather important for good search results, because some people were creating pages with lists of keywords just to get a higher ranking. Pages with less useful content would rarely be pointed to by other pages and would get a lower PageRank. The customizability can be used for a search engine for one's own website: the random jumps can be jumps back to the main page.

We call PageRank, where the preference vector has the value $\frac{1}{node\ count}$ at every position, global. When only selected nodes[2] have non-zero entries in the preference vector, we call it local, because the best results are more likely to be close to the non-zero entries than far away. In [Pag98] a preference vector is tested, where the only non-zero entry is the website of the computer scientist John McCarthy. Their test works pretty well, and favours the websites of his colleagues over foreign, popular universities.

FolkRank

There are a lot of other methods, that are based on PageRank. One of them is FolkRank [HJSS06], which applies PageRank to users, posts, and tags of a social network. Folksonomies are informal taxonomies or classifications created by non-expert users. They can be realised using tags. These tags are set by the user creating the post. For FolkRank, users, posts, and tags become the nodes of the graph. For each post, undirected edges are created: to the user, who created the post, and to the tags associated with the post. The preference vector (see Section 2.1.3) is changed for a query such that users and tags matching the query terms get a weight of one, while rest is set to zero. The developers of FolkRank had the issue that the undirected edges resulted in minor changes of the PageRank when changing the query. They solved this issue by using the difference between the local (query-dependent) and the global (query-independent) PageRank, which in the end worked quite well.

ObjectRank

ObjectRank [BHP04] is a ranking method based on PageRank, but instead of websites, it is intended for use in KGs. The predicates are assigned two weights, one for each direction[3]. This effectively just doubles the edges in the graph and adds weights to each edge. The weights define the relative jump probability to the next node. For example, the passive form of a predicate is typically assigned the same weights as the active form, just with the weights switched around. For example, the weights of the predicates can be chosen to reflect authority transfer, or a different preference, depending on the use-case. Then the typical PageRank is calculated. The query is performed in the same way by changing the preference vector.

[2]e.g. keyword entities

[3]in direction of the edge, and backwards

2.1.4 HITS

Hyperlink-Induced Topic Search (HITS) [Kle98], also called ‘Hubs and Authorities’, are two rankings, that were developed at a similar time as PageRank. HITS defines hubs (hub-score), which are like search engines in the context of the internet, and authorities (authority-score), which are sites with their own content. The idea of the model is, that nodes are good hubs, if they point to a lot of nodes. They are good authorities, if a lot of hubs point to them. Good hubs/authorities are worth more than bad ones. In the algorithm, hub-scores and authority-scores are updated in an alternating manner, until the scores are no longer changing significantly. Notice the link directions in Equation 2.14 and 2.15, so the hub-score depends on the out-links, and the authority-score on the in-links. $HubScore_i(n)$ and $AuthScore_i(n)$ define the scores of a node n in the i-th iteration.

$$HubScore_0(n) = 1 \tag{2.12}$$

$$AuthScore_0(n) = 1 \tag{2.13}$$

$$HubScore_{i+1}(n) = \sum_{\forall m,\ n \to m} AuthScore_i(m) \tag{2.14}$$

$$AuthScore_{i+1}(n) = \sum_{\forall m,\ n \leftarrow m} HubScore_{i+1}(m) \tag{2.15}$$

After each iteration, the hub- and authority vectors are normalized by dividing them by their Euclidean length. The procedure is stopped, when the values have converged. Both scores can be equally interesting, so the best approach is probably to combine them (see Section 2.3). A known issue with all purely graph based algorithms is topic drift [vu]: just because of good connections to a site mentioning the keywords, a site gets a good score, even if it is not really related.

In summary and in comparison to PageRank, the HITS scores not only consider the incoming edges of a node, but also the outgoing edges with their second score. In both, the scores are distributed over the whole network until they converge, so nodes are not only dependent on their neighbours, but the whole community around them.

2.1.5 Word Embeddings

Machine learning models can only work on vectors, so we need to transform our terms to them. Typically, terms are encoded as 1-of-N vectors. In this one-hot encoding, each term gets a dedicated index of the vector. All values, except at the dedicated index, are set to zero. The value at the dedicated index is set to one. The problem of the one-hot encoding lays in the vastness of our vocabulary, such that the vector is very long and most values are zero. Additionally, there are many terms for similar, or even the same things. Like ‘car’ and ‘automobile’, which have no similarity in character nor vector appearance at all.

Word2Vec [MCCD13a] is a popular group of machine learning models. It uses continuous-bag-of-words or skip-grams and huge amounts of text, to calculate more efficient word representations. It additionally incorporates that similar terms have close representations, because the embeddings are optimized in a way, that terms, which appear close to each other in the text, have similar embeddings. The dimension of the vector representing a term is typically chosen to be between 100 and 1000.

Skip-grams [MCCD13b] is a machine learning model, that given terms from a sentence, it is trained to predict a missing term of that sentence. When similar words appear next to each other, they result in a similar prediction, because they are more likely to be from the same sentence.

Another algorithm to generate embeddings, specifically for knowledge bases, is RDF2Vec [RRN$^+$17], which generates entity embeddings. In RDF2Vec the triples are converted to sequences using graph-walks, and graph kernels. Graph-walks is the process of creating sentences by traversing the graph. Graph kernels calculate the structural similarity between two graphs. The sequences then are used to train a neural language model with an internal representation to estimate the probability of a triple appearing in the graph. The internal representation is used as embedding.

Amazingly, embeddings often turn out to be near-linear. The most famous example is that the closest vector to $(king - man) + woman$ was found to be *queen* in Word2Vec. However, examples like that sometimes only work, if the word embeddings in the formula are excluded from the result set [NvNvdG19]. Word embeddings are especially interesting to find represent words in different languages for translation, or to cluster similar concepts.

To compare the embedded terms, the dot product, or an l-norm of the difference can be used. Typically, the dot product on the normalized vectors is used, which is equivalent to the cosine of the angle between the vectors. This approach to compare terms is called Cosine Similarity [SG01].

The Equations 2.16 to 2.27 show example word embeddings, where $\overrightarrow{WE}(i)$ is the word embedding of the term i, and $\overrightarrow{nWE}(i)$ is the normalized word embedding of i. Using these word embeddings and the cosine similarity, we can deduce that 'Dog' is closer to 'Wolf' than 'Cat'.

$$\overrightarrow{WE}(Dog) = [3, 1, 0] \tag{2.16}$$
$$\overrightarrow{WE}(Cat) = [3, 5, 0] \tag{2.17}$$
$$\overrightarrow{WE}(Wolf) = [3, 2, 1] \tag{2.18}$$

$$\overrightarrow{WE}(Cat) \cdot \overrightarrow{WE}(Dog) = 3 \cdot 3 + 5 \cdot 1 + 0 \cdot 0 = 14 \tag{2.19}$$
$$\overrightarrow{WE}(Wolf) \cdot \overrightarrow{WE}(Dog) = 3 \cdot 3 + 2 \cdot 1 + 1 \cdot 0 = 11 \tag{2.20}$$

$$\begin{aligned}\overrightarrow{nWE}(Dog) &= \frac{\overrightarrow{WE}(Dog)}{|\overrightarrow{WE}(Dog)|} \\ &\approx [0.95, 0.32, 0.00] \end{aligned}\tag{2.21}$$
$$\overrightarrow{nWE}(Cat) \approx [0.51, 0.86, 0.00] \tag{2.22}$$
$$\overrightarrow{nWE}(Wolf) \approx [0.80, 0.53, 0.27] \tag{2.23}$$

$$|\overrightarrow{WE}(Cat) - \overrightarrow{WE}(Dog)|_1 = |[0, 4, 0]|_1 = 4 \tag{2.24}$$
$$|\overrightarrow{WE}(Wolf) - \overrightarrow{WE}(Dog)|_1 = |[0, 1, 1]|_1 = 2 \tag{2.25}$$

Method	Basis	Simple	Fast Init.	Fast Query	Quality
TF-IDF	text	●	◕	◕	◑
Okapi-BM25	text	●	◕	◕	◑
KG-Distance	graph	●	●	●	◔
Global PageRank	graph	◕	○	◕	◔
Global FolkRank	graph	◕	○	◕	◔
Global ObjectRank	graph	◕	○	◕	◔
Local PageRank	graph	◕	●	○	◑
Local FolkRank	graph	◕	●	○	◑
Local ObjectRank	graph	◕	●	○	◑
HITS	graph	◕	●	○	◑
Word Embeddings	text	◑	◔	◑	◕

Table 2.1: Summarized comparison of different ranking strategies.
○: one of the worst; ●: one of the best

$$\begin{aligned} \mathit{Cosine\ Similarity}(\mathit{Cat}, \mathit{Dog}) &= \overrightarrow{nWE}(\mathit{Cat}) \cdot \overrightarrow{nWE}(\mathit{Dog}) \\ &\approx 0.76 \qquad (2.26) \\ \mathit{Cosine\ Similarity}(\mathit{Wolf}, \mathit{Dog}) &\approx 0.93 \qquad (2.27) \end{aligned}$$

If an embedding of a word- or term group is required, the average of the word embeddings can be used. The closer the vector representations are, the better is the result. In [KBdR16], the authors state that using the average has proven to be 'surprisingly successful and efficient'. In their paper they use a neural network to create word embeddings, which are especially suitable to compare sentences using the average of the terms for a sentence embedding. They reached state-of-the-art performance using their approach in 2015. In [NFS17], TF-IDF is used to weight rare words higher, which significantly improves the correlation results on the Arabic Monolingual Pairs Semantic Textual Similarity task (see [Qat17]).

2.1.6 Comparison Overview

In Table 2.1, we provide an overview over the previously discussed ranking strategies. The column 'Basis' show whether the techniques work on text documents or a graph. 'Simple' is an estimation of how easy is this method to implement. 'Fast Init.' is a relative estimation on how much computation time is required before documents can be ranked, and 'Fast Query' of the computational time required at query time. Indexes can be used to reduce the time of queries, but they need additional computation time and storage. The last column 'Quality' is an estimation of the quality of the ranking. All these assessments are based on the subjective opinion of the author.

2.2 Evaluation of Ranking Algorithms

When evaluating ranking strategies, it is necessary to define a measure of how good the result of an algorithm is in comparison with a ground truth ranking. Usually only the top ten results are ranked, because most of the time, only the first page is visited [Dea19], and sites like Google show ten results per page. Typical examples for evaluation functions are precision, recall [Pow08], F-measure [VR79, Sas07], and Normalized Discounted Cumulative Gain (NDCG) [JK00]. The input consists of pairs of keywords and results with relevance rankings. Precision, recall and F-measure are defined by the Equations 2.28, 2.29 and 2.30 respectively.

$$Precision = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Positives} \tag{2.28}$$

$$Recall = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Negatives} \tag{2.29}$$

$$F\text{-}measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.30}$$

True Positives and *False Positives* are documents, which were labelled as being relevant. *True Positives* were correctly labelled, *False Positives* not. *False Negatives* are documents, which were labelled non-relevant although they are relevant.

The metric Hits@N (Equation 2.31) is the percentage of relevant documents in the top-N results. N is typically one or ten for Hits. The Cumulative Gain (CG), shown in Equation 2.32, is a similar metric, which adds a state in-between being relevant or not: being related. The relevance score is defined as two, if the result is relevant, one if it is related, and zero otherwise. Being related can be defined as being part of the same topic, or similar. The judgement of relevancy for all results is usually done manually by human volunteers. The Discounted Cumulative Gain (DCG) divides the relevance by the rank of the respective entry within the list of results (see Equation 2.33).

$$Hits = \frac{1}{N} \sum_{rank=1}^{N} 1 \text{ if relevant, 0 else} \tag{2.31}$$

$$CG = \sum_{rank=1}^{N} relevance_{rank} \tag{2.32}$$

$$DCG = \sum_{rank=1}^{N} \frac{relevance_{rank}}{ln(rank + 1)} \tag{2.33}$$

The Normalized Discounted Cumulative Gain (NDCG) was introduced, because for a query, there might not be N relevant documents. It is the DCG divided by the highest DCG observed for that query, such that the optimal result gets a one as score, independently of N and the number of matching documents available. The formula for NDCG is given in Equation 2.34. The variables r, and r_2 stand for a result.

$$NDCG(r) = \frac{DCG(r)}{\max_{\forall r_2 \in known\ results} DCG(r_2)} \in [0..1] \tag{2.34}$$

2.3 Combining Rankings

To achieve better results, multiple methods are combined. Google have used a lot of ranking signals in their algorithm [Goo]. Combining rankings requires scores to be somehow merged into one, because the goal is a single ranking. To avoid working on very different scales of magnitude, it is a good idea to normalize the scores of each ranking into a comparable range.

Besides combinations to improve the ranking quality, in production a pipeline with simple ranking methods like TF-IDF is used to extract the top-N relevant results, and then a computationally more expensive approach to re-rank those top-N results to the top k, k « N.

The assumption, that there must be a perfect rank merging is simply made, but it has been proven invalid by Arrow's Theorem [Arr12]. The fairness criteria non-dictatorship, independence of irrelevant alternatives, and unanimity can not always be fulfilled together. Besides that, merging ranks always needs to consider, how the ranks were created, such that irregularities do not give unintended advantages to single voters. The following collection consists of algorithms, which could be used to join arbitrary ranks.

In the following, $R_i(e) \in \mathbb{N}$ is the rank of the i^{th} method on entity e, while $S_i(e) \in \mathbb{R}$ is its normalized score, $S_{comb}(e) \in \mathbb{R}$ is the combined score, and $w_i \in \mathbb{R}$ is the weight of the i^{th} rank or score. Ranks are determined by ordering the individual results by their score. So both values represent similar concepts, but use inverse scales: a relevant document is supposed to receive a higher score, but a lower rank/index.

If proper training data is available, LTR can be used to determine good weights by using a linear model.

2.3.1 Weighted Sum of Indices or Scores

One of the simplest methods is to add the indices of the rankings together, and use this as a penalty (lower is better). This is more stable for measures with large gaps, but at the same time, the information about the gap width is lost, which might be important too.

Weighting can be done by multiplying the indices or scores with weights.

$$S_{comb}(e) = -\sum_i w_i \cdot R_i(e), \text{ higher is better} \tag{2.35}$$

or

$$S_{comb}(e) = \sum_i w_i \cdot S_i(e), \text{ higher is better} \tag{2.36}$$

The issue with this algorithm on indices is that a single very bad ranking can seriously impact the overall ranking, even if all other scores disagree. A way to prevent this is to use the inverse rankings:

$$S_{comb}(e) = \sum_i w_i / R_i(e), \text{ higher is better} \tag{2.37}$$

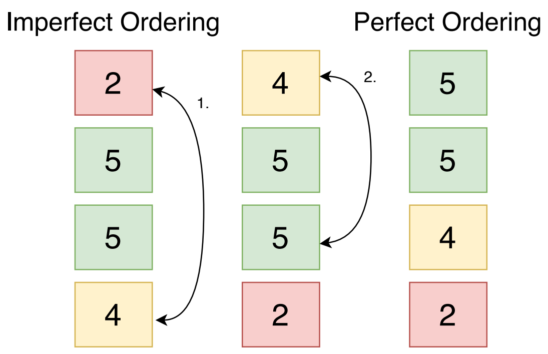

Figure 2.1: Counting Required Swaps: Here two swaps are needed.

2.3.2 Weighted Product of Scores

Another simple approach is to multiply the individual normalized scores, like in the geometric mean. Contrary to the geometric mean, applying the root is unnecessary here, because only the order of values is relevant for the ranking. It is very similar to the weighted sum. In this case, the weighting can be done by raising the scores to the w_i-th power (see Equation 2.38). E.g. squaring a score is equal to giving it two votes. A bias term b_i can be added to avoid annihilation by a score with a value of zero, which would result in a total score of zero, even if all other scores are high.

$$S_{comb}(e) = \prod_i (S_i(e) + b_i)^{w_i} \tag{2.38}$$

2.3.3 Learning to Rank

Learning To Rank (LTR) [Li11] is a large group of methods using machine learning on feature vectors to rank search results. A very simple machine learning model is the linear model, where all features are multiplied by a weight specific to this feature, and then all values are added (see Section 2.3.1). The optimal parameters (or weights) for the model are found by training the model. If results are generally better for small values of a score, rather than for large values, the machine learning model will learn this as well. This training is done by effectively testing different variations of the parameters on sample/training data and iterative optimization of the model parameters. In machine learning, the 'loss function' is the function, that measures how good or bad the model parameters are.

Popular examples of LTR methods are RankNet, LambdaRank and LambdaMART from Microsoft [Bur10]. They all are way more complex than the linear model to get better results. In RankNet the loss function is equal to the minimum number of swaps required to get a perfect result, as shown in Figure 2.1, where numbered squares represent search results and the numbers represent the ground truth relevances.

It is optimized using stochastic gradient descent [Bur10]. LambdaRank is a derivative of RankNet, where the difference of the NDCG (see Section 2.2) is included in the gradient descent. This resulted in faster training and better results. Further improvement was achieved using LambdaMART, which is a combination of the machine learning model Multiple Additive Regression Trees (MART) [Fri01, Fri02] and a loss function similar to the one used by LambdaRank.

Problems with LTR

A problem with machine learning compared to traditional algorithms is that a lot of high-quality training data is required. Established search engines, like Google, YouTube, or Watson from IBM can use their existing user base to train their model, but new systems need to collect the data themselves. One possibility is to use existing datasets like [QL13] or [CC11]. Both Google (RankBrain [Cla15]), and Watson [Wat20] by IBM[4] use LTR at least partially in their search. Another requirement for LTR, or machine learning in general, is that the loss function needs to be defined, which is subsequently optimized in training. The evaluation metrics from Section 2.2 can be used as loss functions, because they indicate how good a model with its parameters is. However, the function needs to be inverted in some way to be a 'real' loss function, because a high value is good for the discussed evaluation metrics, while a high loss is bad. A simple way to invert a function is to use the negated value.

Another problem of machine learning is to find a model with the correct amount of flexibility: a restrictive model has not enough degrees of freedom (underfitting), while a too complex one might have too many (overfitting). If underfitting is a problem, the model needs to be altered. Strategies to avoid overfitting include regularization (smoothing the loss function) and stopping training as soon as overfitting is detected. Overfitting can be detected by comparing the loss on the training and the validation data: if the error on the training data is much smaller than on the validation data, the model is too well adapted to the training data.

In the context of LTR, any scoring function (e.g. PageRank) can be used as a feature in the model. For a simple, linear model, LTR is a way to automatically find near-optimal weights to combine different rankings. Features, where small values are better, get a negative weight, and features, where large values are better, get a positive weight.

To train the model, there are two types of training data: explicit and implicit. Explicit data requires users, often domain experts to manually assign a relevance score to pairs of queries and results. Implicit rankings are relevance scores estimated by a click-model. The simplest model is the click-trough-rate: what is the percentage of users, who have seen the ranked entity and have clicked on it? Disadvantages are that it is more an estimation of attractiveness than usefulness and that it does not consider the position bias[5] [WGB$^+$18]. A more complex model is the Dynamic Bayesian Network Click Model [CZ09], which observes the order of clicks as well, and models attraction and satisfaction.

When temporal data, or certain flags, e.g. a product being free of charge, are available, and they correlate with preferred choices, they can be used as ranking signals too [SCP16]. A linear machine learning model can help to find the best weighting for them.

If performance is critical, feature filtering based on usefulness and computational cost can be used to reduce the number of features. E.g. a mediocre, inexpensive feature could be included, while a mediocre, expensive feature should be discarded. If the entries to be ranked differ a lot, it might be advantageous to train a model for each major category, and join the results of those groups [SCP16].

[4]International Business Machines Corporation

[5]the first results are easier to see and reach, and therefore clicked more often

2.4 Related Approaches

We are not the first ones with the goal to rank query results over KGs. In [BM05], they propose a series of scores on the nodes, the edges, and the semantic classes of the nodes, which then are combined into a total score. They work purely on the graph structure and its class hierarchy, and do not use features from the IR community. They only rank by importance, not relevance.

In their calculations, they give the relations of the KG weights of importance-transfer manually. There are two weights per predicate, one for each direction, as in ObjectRank (see Section 2.1.3). In the same way as HITS (see Section 2.1.4), they then assign each node two scores. In HITS, they are called Hub- and Authority-Score, and in [BM05] they are called Subjectivity- and Objectivity-Score. Additionally, they assign each class an importance score. These three scores then are combined into a fourth score called 'importance score'. This score is still independent of the query.

To calculate the score of the subgraph, its edges and nodes are assigned a score, which then are combined into the score for the ranking. The formula can be seen in Equation 2.42. The subgraph is r, $decayFactor$ is the base for the exponential decay, which has a value between zero and one. $REoI$ (result entities of interest) is the set of result entities (variables), which were part of the SELECT clause, $Distance(node\ a, node\ b)$ is the number of edges on the shortest path from a to b. $IPF(predicate\ p)$ is the Inverse Property Frequency of the property. Analogous to IDF, it equals $log(N/n)$, where N is the number of triples in the KG and n is the number of triples containing this property in the KG. It has the purpose to make rare properties more valuable.

$$NodeScore(r) = \sum_{n \in nodes(r)} ImportanceScore(n) \cdot pow(decayFactor, \min_{m \in REoI} Distance(n, m)) \tag{2.39}$$

$$EdgeScore(r) = \sum_{(n_1,n_2)=e \in edges(r)} IPF(e) \cdot pow(decayFactor, \min_{m \in REoI} Distance(n_1, m)) \tag{2.40}$$

$$Score(r) = w_1 \cdot \frac{NodeScore(r)}{|nodes(r)|} + w_2 \cdot \frac{EdgeScore(r)}{|edges(r)|}, \tag{2.41}$$
$$\text{where } w_1 + w_2 = 1 \text{ and } w_1, w_2, decayFactor \in [0..1]$$

The algorithm of [BM05] is modified in [IKIT14] and then a small study is conducted to evaluate the results. Their second algorithm defines a score on all triples, called 'TripleScore'. Its definition can be seen in Equation 2.43. PageRank is used instead of HITS as the measure for importance of a node. To avoid having zero as PageRank for literals, they assign the average importance of all nodes to the literals instead. s, p, o are respectively subject, predicate, and object of a triple. C_s is the class of the subject, and $linkNum(node\ n)$ is the number of triples in the KG, where n is the subject. dst is the minimum distance from the subject and object to the entities of the SELECT-clause. $PFIPF(subject\ s, predicate\ p)$ (Property Frequency - Inverted Property Frequency) is a derivation of TF-IDF, which contrary to IPF only works on relations, where the subject is of the same class as s, so the value becomes more specific to the used class. In the paper, they set *decayFactor* to a value of 0.5.

$$TripleScore(s,p,o) = \frac{PR(C_s) \cdot PFIPF(s,p) \cdot PR(o)}{linkNum(s) + linkNum(o) - 1} \tag{2.42}$$

$$Score(G) = \frac{\sum_{(s,p,o)\in G} TripleScore(s,p,o) \cdot pow(decayFactor, dst)}{|triples \in G|} \tag{2.43}$$

In their study, twelve people evaluate the results from the base algorithm and two modified versions on four queries, and the top ten results. Both improved algorithms deliver better results, while their second algorithm is the best. However, they still only rank importance, not relevance, because they do not compare topics in any way.

Chapter 3

Concepts

A lot of recent papers, that address document ranking, use Learning To Rank (LTR). The latter can achieve state-of-the-art results and is an important part of the most popular search engine Google [Cla15, Net20], and commercial search services like Watson [Wat20]. This motivated us to adopt LTR for ranking graph-shaped search results using features from both IR- and graph communities. We concentrate on feature engineering, rather than the machine learning model.

As a running example, we consider the KG depicted in Figure 3.1 with associated descriptions as given in Table 3.1. Literals are not seen as nodes in the following, because usually entities are most important. Entities and nodes are therefore equivalent in the following. ‘Elvis’ is the pitcher of the baseball club ‘Super Bats’, which got him the nickname ‘The Bat’. ‘David’ is more popular than ‘Elvis’ or ‘Cesar’, because he knows more people. Because of this, the probability for a random person looking for him may be higher than for ‘Elvis’ or ‘Cesar’.

Let’s consider the scenario, where ‘Alice’ is looking for someone from the ‘Super Bats’. She additionally knows that he must be a direct or indirect friend of her friends. This information need can be translated to the SPARQL query in Listing 3.1. The plus operator after ‘ex:isFriendOf’ allows for a path of arbitrary length using that property. This query has the following bindings: result node (see Figure 1.3) ‘Elvis’, ‘Cesar’ and

Label	Alias	Description
Alice	Allie	A cryptography girl
Bob	-	Loves Decryption Riddles
Cesar	-	Playing with the Super Bats
David	Dave	Great for partying!
Elvis	The Bat	Pitcher of Super Bats
Super Bats	-	Baseball Club in Texas
Baseball Club	-	Organization for playing baseball
Friend 1	-	-
Friend 2	-	-
Friend 3	-	-

Table 3.1: Labels, aliases and descriptions for the entities in the example KG (Figure 3.1).

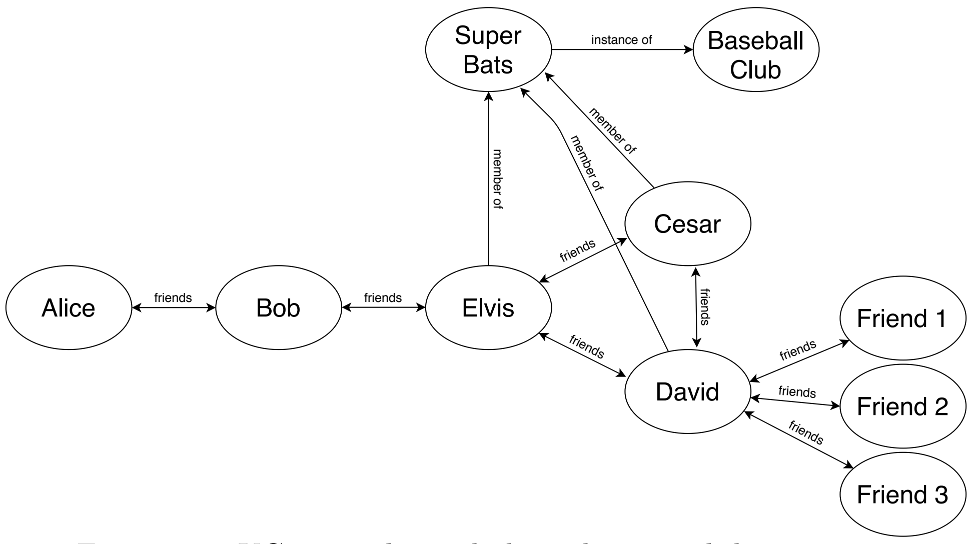

Figure 3.1: KG example, excluding aliases and descriptions.

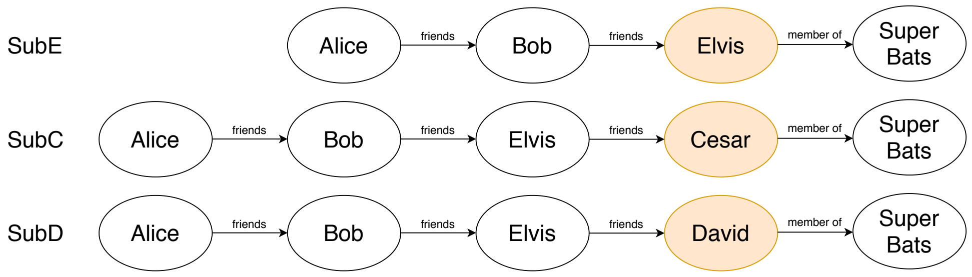

Figure 3.2: Result subgraphs SubE, SubC and SubD.

'David' with the subgraphs in Figure 3.2. The white and orange nodes are query and result nodes respectively. The subgraph structure is the same for SubC and SubD, because the length of their friendship chain is the same, and the query does not allow for more variation in the subgraph.

```
SELECT ?searched
WHERE {
ex:Alice ex:isFriendOf+ ?searched .
?searched ex:isMemberOf ex:SuperBats .
}
```

Listing 3.1: SPARQL query corresponding to the considered scenario.

Based on methods discussed in Section 2.1, we propose the following features to be used with LTR. We categorize these features into five groups, depending on their origin and characteristics:

- based on the whole KG structure,
- based on the subgraph structure only,
- TF-IDF-based,

Entity	SC1	SC2
Alice	(Alice, Allie)	(Alice, Allie, A, cryptography, girl)
Bob	(Bob)	(Bob, Loves, Decryption, Riddles)
Cesar	(Cesar)	(Cesar, Playing, with, the, Super, Bats)
David	(David, Dave)	(David, Dave, Great, for, partying)
Elvis	(Elvis, The, Bat)	(Elvis, The, Bat, Pitcher, of, Super, Bats)
Super Bats	(Super, Bats)	(Super, Bats, Baseball, Club, in, Texas)
Baseball Club	(Baseball, Club)	(2 x Baseball, Club, Organization, for, playing)

Table 3.2: SC1 and SC2 of entities of the example KG (see Figure 3.1).

- embedding-based and
- domain-specific properties of entities.

Some of the proposed features work on string data, so we need at least one function, which creates string data from a node. We define the following three functions to capture different levels of closeness to the entity:

- String Class 1 (SC1) is the Bag of Words (BOW) (see Section 2.1.1) of all labels and aliases of a single entity.
- String Class 2 (SC2) is the BOW of all literal values of an entity. It includes labels, aliases, descriptions and comments.
- String Class 3 (SC3) is SC2 of the entity itself, and all its direct neighbours joined.

Table 3.2 presents SC1 and SC2 of entities of the example KG in Figure 3.1 and Table 3.3 shows the SC3s. 'Labels' (primary name(s)), 'Aliases' (secondary names) and 'Descriptions' (descriptions or comments) are placeholders, which have to replaced with the corresponding predicates from the KG. In the RDF schema [BG14], 'rdfs:label' and 'rdfs:comment' are the predicate values for aliases and descriptions. There is no separate predicate for a preferred label, but that does not matter for the SC values, because we only use labels and aliases together.

Using case-insensitive bags is fine for the example, but in scientific projects, it should be considered, because for example 'm' and 'M', which are the prefixes for 0.001 and 1,000,000 in the SI system[1], are very different from each other.

The SC values for a set or list of nodes N is defined as all the BOWs of the individual nodes combined[2]:

$$SC(N) = \bigcup_{n \in N} SC(n) \tag{3.1}$$

Training a model for LTR, or machine learning in general, works best, if the features are normalized. For simplicity, we propose to use a single method for all not-already

[1]International System of Units

[2]the term frequencies for the words are added together from all source BOWs

Entity	SC3
Alice	(Alice, Allie, A, cryptography, girl, Bob, Loves, Decryption, Riddles)
Bob	(Bob, Loves, Decryption, Riddles, Alice, Allie, A, cryptography, girl, Elvis, The, Bat, Pitcher, of, Super, Bats)
Cesar	(Cesar, Playing, with, 2 x the, 3 x Super, 3 x Bats, Baseball, Club, in, Texas, David, Dave, Great, for, partying, Elvis, Bat, Pitcher, of)
David	(David, Dave, Great, for, partying, 3 x Super, 3 x Bats, Baseball, Club, in, Texas, Cesar, Playing, with, 2 x the, Elvis, Bat, Pitcher, of, 3 x Friend, 1, 2, 3)
Elvis	(Elvis, 2 x The, Bat, Pitcher, of, 3 x Super, 3 x Bats, Baseball, Club, in, Texas, Bob, Loves, Decryption, Riddles, Cesar, Playing, with, David, Dave, Great, for, partying)
Super Bats	(3 x Super, 3 x Bats, 3 x Baseball, 2 x Club, in, Texas, Organization, 2 x for, 2 x playing, Cesar, with, 2 x the, David, Dave, Great, partying, Elvis, Bat, Pitcher, of)
Baseball Club	(3 x Baseball, 2 x Club, Organization, for, playing, Super, Bats, in, Texas)

Table 3.3: SC3 of entities of the example KG (see Figure 3.1).

normalized features: the minimum (*min*) and maximum (*max*) values are used as anchor points, and then a transform is applied. Equivalently to the min-max normalization, the 20th and 80th percentile of the values could be used, if the values follow a power-law-, or a Gaussian distribution, or have far outliers. The values to consider for minimum, maximum, 20th, or 80th percentile can be calculated on a per query basis[3] or for the whole training data set[4]. The case sensitivity consideration and the normalization method can influence the accuracy, so it should be tested on data samples beforehand. A simple method would be a linear transformation:

$$linear = \frac{value - min}{max - min} \in [0, 1] \tag{3.2}$$

An alternative would be the sigmoid curve, applied after the linear normalization. It additionally requires a curve-sharpness parameter σ.

$$sigmoid = \frac{1}{1 + \exp^{-(linear - 1/2)\cdot\sigma}} \tag{3.3}$$

A few, far outliers can reduce the numerical precision of the normalized values, if the minimum and maximum values are used as anchor points. A distribution, that has most values in [0,1], and a few outliers at huge positive and negative values would map the entire [0,1] range to only a tiny range of values, which might even vanish completely due to the limited floating point numbers in today's computers. This can be shown with Equation 3.2[5] while calculating $value - min$: when a large (absolute) and a small (absolute) floating point number are added[6], numerical precision from the smaller number may be lost, because the relative precision is finite, and all more-significant bits from the large number need to be kept. In the example, the number losing precision is our input. If both anchor points are equal, a constant value like 0.5 should be used instead of NaN (Not-a-Number, undefined), which would be the result because of the division by zero in Equation 3.2. Otherwise, the NaN value would propagate through the neural network[7] and the score would be NaN.

In the following we will discuss how to apply the techniques of Chapter 2 to a graph environment. The results are features that, pending a normalization, can be used as inputs to our LTR approach to derive a final, combined score and ranking.

3.1 Knowledge Graph-Based

The first group of features works on the KG as a whole. The graphs can become very large, so caching the features is a good idea. Our selected KG-based features are all query-independent, and try to capture either importance or popularity of entities. We propose the in-degree of a node and the out-degree, separately averaged (see Section 1.3) over query nodes (QN) and result nodes (RN). The in-degree and out-degree of node n is

[3] called 'local' normalization

[4] called 'global' normalization

[5] the formula can be written differently, e.g. as $(value - (max + min)/2)/(max - min) + 0.5$, where $(max + min)/2$ may be zero, but the issue remains

[6] or subtracted

[7] if it works with numbers supporting NaN; e.g. IEEE 754 floating point numbers do, integers do not

Node	In-Degree	Out-Degree	PageRank	HubScore	AuthScore
Alice	1	4	0.20	0.84	0.21
Bob	2	4	0.23	0.84	0.42
Cesar	2	5	0.22	1.05	0.42
David	5	9	0.38	1.90	1.05
Elvis	3	7	0.27	1.47	0.63
Super Bats	3	3	0.26	1.00	0.63
Baseball Club	1	2	0.22	1.00	0.33

Table 3.4: KG-based features for the example graph in Figure 3.1.

Subgraph	QN/RN	Average In-Degree	Average Out-Degree	Average PageRank	Average HubScore	Average AuthScore
SubC	QN	2.25	4.50	0.24	1.04	0.47
SubD	QN	2.25	4.50	0.24	1.04	0.47
SubE	QN	2.00	3.67	0.23	1.05	0.42
SubC	RN	2.00	5.00	0.22	1.05	0.42
SubD	RN	5.00	9.00	0.38	1.90	1.05
SubE	RN	3.00	7.00	0.27	1.47	0.63

Table 3.5: Averaged in-degree, out-degree, PageRank and HITS scores for QN and RN.

the amount of relations, where n is the object or subject respectively. All relations from the KG are counted, not just from the subgraph. Additionally, we propose to use both PageRank and HITS scores (see Section 2.1.3 and Section 2.1.4), averaged over query and result nodes (see Figure 1.3) separately again. All features indicate popularity: a high in-degree or out-degree may indicate interest of many people in the topic. High PageRank and HITS score may indicate that better on a global scale: if the topic itself is popular.

Table 3.4 shows the values of the five KG-based features of some entities for our example KG (see Figure 3.1). PageRank and HITS scores were calculated using our online tool[8]. The tool is open source and available in [Noa20]. No normalization was applied, no preferences were set, and the random jump probability of PageRank was set to 0.15. The JSON representation of the graph can be found under 'calc/sample.json'[9].

In Table 3.5, we have calculated the average of the query and result node values for all subgraphs. The averages for the result nodes are identical to the values in Table 3.4, because there is only one result node in each subgraph. The values for the QN for SubC and SubD are the same, because the two groups of nodes are identical.

[8]https://phychi.com/pagerank

[9]https://github.com/AntonioNoack/WebPageRank/blob/master/calc/sample.json

Subgraph	Radius	Diameter	*DistanceScore*
SubE	2	3	$1+1+2=4$
SubC	2	4	$1+1+2+3=7$
SubD	2	4	$1+1+2+3=7$

Table 3.6: Radius, Diameter, and *DistanceScore* for the example subgraphs.

3.2 Subgraph-Based

The second group of features works on the subgraphs of the results. It is intended to represent how compact the subgraph is. Our idea behind it is, that the smaller, and denser the subgraph is, the better is the result. In the case of our example, there is the property chain 'is friend of' between 'Alice' and the result node, which can become arbitrarily long. A closer friend would probably be more likely the more relevant result for 'Alice'.

We propose three features to capture the compactness: radius, diameter (see Section 2.1.2), and a custom distance subgraph score for the subgraph G. Our custom distance score uses the sum of the distance from every query node q to all result nodes r. Using the sum captures two compactnesses: the amount of nodes, and the average distance, which both should be small. The distance score is defined as:

$$DistanceScore(G) = \sum_{q \in QN} \sum_{r \in RN} Distance(q, r) \tag{3.4}$$

For SubE, the *DistanceScore* is calculated as follows, where r is 'Elvis':

$$\begin{aligned} DistanceScore(SubE) &= Distance(‘Alice‘, r) + Distance(‘Bob‘, r) + \qquad (3.5)\\ &\quad Distance(‘Super\ Bats‘, r) \qquad (3.6)\\ &= 2 + 1 + 1 = 4 \qquad (3.7) \end{aligned}$$

The structure of SubC and SubD are the same, so all scores are the same for them. Although SubE is smaller than SubC and SubD, their radius is still the same. The diameter shows the difference. With the idea, that more compact subgraphs are better results, a smaller value for these features is better. If there are no logical combinations of constraints in the query, and all variables are always distinct, the radius, diameter and DistanceScore will unfortunately be constant. With only a few constraints, it still may have only a small range of scores. In combination with other features, these measures still can be useful.

Figure 3.3 shows an exemplary case, where *DistanceScore* delivers different results for two cases, while radius and diameter of the two subgraphs A and B are identical. The white nodes are query nodes and the orange node is the result node. The result from *DistanceScore* is that subgraph A is more compact than subgraph B, which matches our intuition, because in subgraph A the result node is more central.

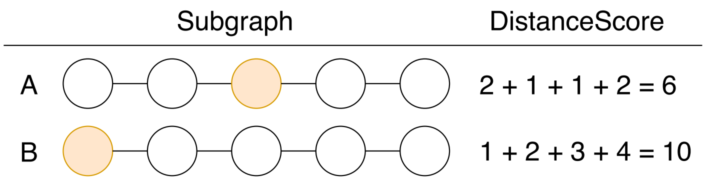

Figure 3.3: *DistanceScore* example case, where radius and diameter are the same.

3.3 TF-IDF-Based

Our first proposed features to actually capture the topic-related relevance, are TF-IDF based approaches: they attempt to find exactly matching words like in traditional IR. Since these approaches require no training, they can be directly used, when the knowledge graph was changed[10] (contrary to word embeddings). By using TF-IDF (see Section 2.1.1), we attempt to capture the importance of a word in the node's string data[11]. We propose to use the following feature ($\mathit{TFScore}(QN, RN, SC)$) for all three SC, because matches in labels may be more important than matches in arbitrary attributes, or the description of a neighbour node.

With this feature, we want to get a metric for the matching of words between query nodes and result nodes. For our proposed subgraph metric *TFScore* (Equation 3.8), we combine the term frequency TF of term w from all query nodes QN and result nodes RN separately. $\mathit{TF}(B, w)$ is the number of occurrences of term w in the BOW B and $SN = QN \cup RN$ is all subgraph nodes together. For simplicity, we use the words directly as terms. This combination from the QN and RN is then averaged over all occurring words in those nodes. $SC(SN)$ is the set of words occurring in the string data of SC from all subgraph nodes SN combined.

$$\begin{aligned} \mathit{TFScore}(QN, RN, SC) = & \frac{1}{|SC(SN)|} \sum_{w \in SC(SN)} \mathit{IDF}(w) \cdot \\ & \mathit{combine}\Big(\mathit{TF}\Big(SC(QN), w\Big), \mathit{TF}\Big(SC(RN), w\Big)\Big) \end{aligned} \tag{3.8}$$

$$\mathit{IDF}(w, KG) = \ln\left(\frac{|\mathit{nodes}(KG)| + 1}{|\{n \in \mathit{nodes}(KG), w \in SC2(n)\}| + 1}\right) \tag{3.9}$$

$\mathit{IDF}(w)$ is the natural logarithm of the inverse ratio of entities, whose directly connected literals (SC2) contain this word. Additionally, we normalize it to avoid division by zero. The one is added to the nominator as well, so $\mathit{IDF}(w)$ always stays non-negative[12]. The range of values of $\mathit{IDF}(w)$ is $[0, \infty)$.

If $\mathit{combine}(a, b)$ is not a linear function in a and b, then the combination of averages (Equation 3.10) is different from $\mathit{TFScore}(QN, RN, SC)$ (average of combinations). This means, that a simple machine learning model, that uses a feature vector as input and

[10] by updates, e.g. extensions or corrections
[11] e.g. stopwords have no importance
[12] a word, which appears in all nodes, gets a negative IDF value, because $\ln(0 < \mathit{value} < 1) < 0$

outputs scores, is not flexible enough for this general task, because the amount of words in $SC(QN)$ can change.

$$\begin{aligned} &\mathit{combine}\Big(\mathit{averaged}(QN, SC), \mathit{averaged}(RN, SC)\Big), \text{ where} \\ &\mathit{averaged}(N, SC) = \frac{1}{|SC(N)|} \sum_{w \in SC(N)} \mathit{TF}\Big(SC(N), w\Big) \cdot \mathit{IDF}(w) \end{aligned} \tag{3.10}$$

Therefore, we propose to either decide on a finite set of combination functions, whose results are then used as individual features, or to train two machine learning models joint. One model for $\mathit{combine}(a, b)$, and the second for the score generation from features. A joint loss function is then created to train the two models together. We propose to use the minimum as a combination function, because then only matching words[13] from the query nodes and the result nodes are evaluated to non-zero.

When using SC3, the string data of direct neighbour nodes is included. This means, that there will always be matches between the string data of query and result nodes, because they are neighbours of each other. Additionally, artificial matches occurs, when query and result nodes share external (not in the subgraph) neighbours. We propose to exclude these parts of the SC3 data. For example, in SubE, 'Bob''s SC3 data should no longer include 'Elvis'' SC3 data, because 'Bob' belongs to the QN and 'Elvis' to RN. In Table 3.7 are the values for all entities in the three subgraph results.

If we intended our TF-IDF features to capture semantic context, we would propose to use stemming or lemmatization to find the word families of the words. Stemming removes endings by rules, while lemmatization uses a large dictionary to look up the word family [MRS08]. Word embeddings are more suitable for this task, so we use TF-IDF only for 1:1 matches of words.

In our example, there are no matching terms in SC1, so those scores would be zero, and would show no preference for any of the results. The only matching words for SC2 are 'Super' and 'Bats'. 'Super' and 'Bats' are part of the nodes 'Super Bats', 'Cesar' and 'Elvis'. Because of this, the results 'Elvis' and 'Cesar' would get the best $\mathit{TFScore}(QN, RN, \mathit{SC2})$.

SC3('Cesar'/'David') needs to be removed from the SC3 values of SubD/SubC, because it is a neighbour of query and result nodes. In SubE, SC3('Cesar') and SC3('David') need to be removed, because they appear as neighbours of 'Elvis' and 'Super Bats', which are part of RN and QN respectively.

In Table 3.8, we have summarized the values, and chose TF to be the absolute term frequency. To get meaningful IDF values, a KG or text corpus larger than ours is needed. For the example, we choose $\mathit{IDF}(\mathit{Super}) = 1.0$, $\mathit{IDF}(\mathit{Bats}) = 1.5$, $\mathit{IDF}(\mathit{the}) = 0.1$, $\mathit{IDF}(\mathit{for}) = 0.2$ and $\mathit{IDF}(\mathit{playing}) = 1.0$. In this case, SubC has the highest $\mathit{TFScore}(QN, RN, \mathit{SC3})$.

[13]words that appear in the string data from both groups of nodes

Subgraph	Entity	SC3, always-matching terms excluded
SubE	Alice	(Alice, Allie, A, cryptography, girl, Bob, Loves, Decryption, Riddles)
SubE	Bob	(Bob, Loves, Decryption, Riddles, Alice, Allie, A, cryptography, girl)
SubE	Elvis	(Elvis, The, Bat, Pitcher, of, Super, Bats)
SubE	Super Bats	(Super, Bats, 3 x Baseball, 2 x Club, in, Texas, Organization, for, playing)
SubC	Alice	(Alice, Allie, A, cryptography, girl, Bob, Loves, Decryption, Riddles)
SubC	Bob	(Bob, Loves, Decryption, Riddles, Alice, Allie, A, cryptography, girl, Elvis, The, Bat, Pitcher, of, Super, Bats)
SubC	Elvis	(Elvis, The, Bat, Pitcher, of, 2 x Super, 2 x Bats, Bob, Loves, Decryption, Riddles, Baseball, Club, in, Texas)
SubC	Cesar	(Cesar, Playing, with, the, Super, Bats)
SubC	Super Bats	(2 x Super, 2 x Bats, 2 x Baseball, Club, in, Texas, Elvis, the, Bat, Pitcher, of, David, Dave, Great, 2 x for, partying, Organization, playing)
SubD	Alice	(Alice, Allie, A, cryptography, girl, Bob, Loves, Decryption, Riddles)
SubD	Bob	(Bob, Loves, Decryption, Riddles, Alice, Allie, A, cryptography, girl, Elvis, The, Bat, Pitcher, of, Super, Bats)
SubD	Elvis	(Elvis, The, Bat, Pitcher, of, 2 x Super, 2 x Bats, Bob, Loves, Decryption, Riddles, Baseball, Club, in, Texas)
SubD	David	(David, Dave, Great, for, partying, 3 x Friend, 1, 2, 3)
SubD	Super Bats	(2 x Super, 2 x Bats, 3 x Baseball, Club, in, Texas, Elvis, the, Bat, Pitcher, of, Organization, for, playing)

Table 3.7: SC3 of entities of the example query with excluded nodes taken into account.

Subgraph	String Class	TFScore
SubC	SC1	0.0
SubD	SC1	0.0
SubE	SC1	0.0
SubC	SC2	$2.5 = IDF(Super) + IDF(Bats)$
SubD	SC2	0.0
SubE	SC2	$2.5 = IDF(Super) + IDF(Bats)$
SubC	SC3	$3.6 = IDF(the) + IDF(Super) + IDF(Bats) + IDF(playing)$
SubD	SC3	$0.2 = IDF(for)$
SubE	SC3	$2.5 = IDF(Super) + IDF(Bats)$

Table 3.8: Example TFScores for SubC, SubD and SubE.

3.4 Embedding Based

Additionally to our TF-IDF based approaches, we also want to use topic modelling to capture the meaning of a group of nodes (see Section 2.1.5). We encode the meaning as a vector of numbers. The basic idea is that the embeddings of closely related groups of nodes are similar, while embeddings for unrelated groups have a greater distance. The idea of this topic modelling is that relevant result nodes are closely related to the query nodes. The first feature for topic modelling, we propose, is to use the cosine similarity between an embedding of the query nodes and the result nodes.

This requires that we define an embedding of a group of nodes. For this purpose we propose a node group embedding $\overrightarrow{NGE}(SC, N)$ of node group N[14] for each string class SC:

$$\overrightarrow{NE}(SC, n) = normalize\left(\sum_{w \in SC(n)} \overrightarrow{WE}(w) \cdot TF\Big(SC(n), w\Big) \cdot IDF(w)\right) \tag{3.11}$$

$$\overrightarrow{NGE}(SC, N) = \frac{1}{|N|} \sum_{n \in N} \overrightarrow{NE}(SC, n) \tag{3.12}$$

The node group embedding is the average of our node embedding $\overrightarrow{NE}(n)$. The node embedding is a weighted linear combination of the word embeddings $\overrightarrow{WE}(w)$ of the words from $SC(n)$. We chose TF-IDF as weight for the word w in an attempt to reflect the relative importance of a word. The components of $\overrightarrow{NGE}$ are in $[-1, 1]$, because of using the average, and normalized vectors. In our proposed feature, we do not weight the nodes in $\overrightarrow{NGE}$, because it is meant as a topic modelling over the whole query. A possible weighting scheme would be to use an exponential decay depending on the distance from the query nodes to the result nodes, as in [IKIT14].

The popular word embedding Word2vec [MCCD13a, Kar18] has issues with rare words, because from low occurrence follows uncertainty [SS15, GHT+18]. There are

[14]N is either QN or RN

projects like FRequency-AGnostic word Embedding (FRAGE) [GHT+18], which try to improve word embeddings in this case. FRAGE tries to fool the generating model into ignoring the word frequency. In our example, we have the issue, that a lot of words are proper names, e.g. ‘Alice’ or ‘Bob’. Using word embeddings for proper names is problematic, because many different entities can share the same name. Simply replacing all names with one word loses information like a probability for sex[15] and origin/nationality. For example, the name ‘Albert’ has German origins and is typically given to males, so an entity with the name ‘Albert’ has a high probability to be male, and has a relatively high chance to be German. Keeping the proper names introduces a partially arbitrary bias, which assigns every individual of name X the average semantic meaning of all Xs. To avoid this issue, we propose a fourth embedding based feature, which uses an entity embedding like RDF2Vec [RRN+17] instead of our node embedding. In the RDF2Vec paper, literals are excluded, so ideally only entities, which are closely connected by relations, get assigned a similar embedding.

To get custom word embeddings with RDF2Vec, there is already pre-trained entity embeddings [Pau20], which can be used to accelerate the training process. Pre-trained word embeddings exist as well, e.g. from Google [wor13].

3.5 Domain-Specific Properties

KGs or additional metadata often contain interesting properties, for example the last time an entry was updated, the price of a certain product, or an assigned category. These fields are always domain-specific. If users generally prefer a high/low value of a numeric property, or a specific category, it can be useful for the ranking. If there is more than one result node per subgraph[16], we propose to use the average value as the feature. Nodes, which do not have the property assigned, could be excluded from the average, or given a default value.

3.5.1 Numeric Properties

The easiest type of features is numerical properties, as they theoretically can be directly used as a feature. Boolean properties could be used as features by encoding them as numbers, e.g. zero and one. In our example (see Figure 3.1), the search system, ‘Alice’ is using, could be a social network. There the number of contacts or friends may be especially interesting as a separate feature, separated from the other KG-based features in Section 3.1. Whether a movie on Amazon is free with ‘Amazon Prime’ is taken as example in [SCP16] ($value \in \{0, 1\}$).

3.5.2 Categorical Properties

Categorical properties could be used as features, too, if available. Possible examples are class membership (each class is a category), the sex of an animal, the marital status of people, or the level of their academic degree. However, especially when some categories

[15]sex chromosomes, allosomes
[16]by multiple variables in the query

Group	Feature
KG	Average In-Degree QN
KG	Average In-Degree RN
KG	Average Out-Degree QN
KG	Average Out-Degree RN
KG	Average PageRank QN
KG	Average PageRank RN
KG	Average HITS-Hub-Score QN
KG	Average HITS-Hub-Score RN
KG	Average HITS-Authority-Score QN
KG	Average HITS-Authority-Score RN
Subgraph	Subgraph Radius
Subgraph	Subgraph Diameter
Subgraph	DistanceScore
TF-IDF	$\mathit{TFScore}(QN, RN, \mathit{SC1})$
TF-IDF	$\mathit{TFScore}(QN, RN, \mathit{SC2})$
TF-IDF	$\mathit{TFScore}(QN, RN, \mathit{SC3})$
Embeddings	$\mathit{CosineSimilarity}\Big(\overrightarrow{NGE}(\mathit{SC1}, QN), \overrightarrow{NGE}(\mathit{SC1}, RN)\Big)$
Embeddings	$\mathit{CosineSimilarity}\Big(\overrightarrow{NGE}(\mathit{SC2}, QN), \overrightarrow{NGE}(\mathit{SC2}, RN)\Big)$
Embeddings	$\mathit{CosineSimilarity}\Big(\overrightarrow{NGE}(\mathit{SC3}, QN), \overrightarrow{NGE}(\mathit{SC3}, RN)\Big)$
Embeddings	$\mathit{CosineSimilarity}\left(\frac{1}{\lvert QN\rvert}\sum_{q\in QN}\mathit{RDF2Vec}(q), \frac{1}{\lvert RN\rvert}\sum_{r\in RN}\mathit{RDF2Vec}(r)\right)$
Domain-Specific	*Domain-specific*

Table 3.9: Overview of the proposed features.

are rare, it is hard to train these cases. In our example KG (see Figure 3.1), using the sex as a feature from an external database[17], 'Alice' is the only woman. Statistically, because she is the only individual of this category, all women could be friends with 'Bob'. If there are only few values for a group[18], a one-hot-encoding (see Section 2.1.5) can be used as a group of features for LTR. If there are multiple result nodes in a subgraph, the average of the one-hot-encodings could be used.

3.6 Overview

Table 3.9 summarizes the previously proposed features, where QN are the query nodes, and RN are the result nodes.

These features are then the input-vector for the LTR network. It outputs a score[19]. The result subgraphs then are sorted by their scores. To compare different models (like

[17]it is not included in the KG example to keep other example calculations simple

[18]e.g. 'male', 'female' and 'not applicable' for the sex of an entity

[19]it would be no issue to use a pair-wise or list-wise approach instead; the features stay the same

the linear model or LambdaMART[20]) and parameters for a model, a metric like NDCG can be used (see Section 2.2). When training data is available, the model can be trained, using NDCG as a target function. The required training data is the KG and a set of queries, results and their human-rated relevances. The results contain their subgraph and the binding, which effectively defines whether an entity is a result node or a query node.

3.7 Limitations

Still, all features have their limitations. Word embeddings (see Section 2.1.5) work only on known words, so unknown[21] words, called 'Out-Of-Vocabulary words', cannot be used correctly, until a word embedding is created for them. This may require retraining of many or all words, which would be expensive. There are more complex methods, which e.g. estimate unknown words by known parts inside them [LM16]. The same is valid for new nodes and relations in the KG as well.

The KG-based scores (see Section 3.1) need to be updated, when the KG structure is modified. The old PageRank and HITS scores can be used as starting points for faster convergence, because for small modifications the old scores already should be close to the optimum.

The subgraph-based scores (see Section 3.2) often vary little, because they work solely on the structure and often the structure of different results is similar or even the same for a query. How much the result's structures can differ, always depends on the query.

Our TF-IDF-based scores (see Section 3.3) partially depend on good descriptions or string data. As described in Section 1.1, matching raw words has sometimes issues with ambiguous words. Structural nodes for metadata, e.g. links to other databases, or notes by users, may introduce noise with unintentional matches. If such nodes exist in the KG, they should be removed for the algorithm.

Domain-specific features (see Section 3.5) need to be hand selected by domain experts, which makes them not universally applicable in contrast to all others. Adding or removing a feature in a complex model (see Section 2.3.3) is complicated and may require the whole model to be retrained. Alternatively an ensemble could be created [KD15], which may reduce ranking quality though[22]. A feature could be removed by always replacing the input with the same value (e.g. 0.5), but this is not ideal either[23] and the model should be retrained instead.

[20]see Section 2.3.3

[21]e.g. foreign or new words

[22]because it is more limited

[23]in a non-linear model; a linear model would always create the same sub-score for this part, so it would not matter

Chapter 4

Conclusion and Future Work

In this Bachelor Thesis, we compared different existing approaches to rank documents based on IR techniques, and graph metrics in Chapter 2. We then proposed a ranking system based on LTR in Chapter 3. In this system, multiple features are combined by a non-linear system to create a better scoring function. We proposed several features, which we consider to be useful for a relevance reflecting ranking. There are two points, which need to be tested to achieve best results, when training data is available: the normalization method and whether the normalization should be done on a per query basis.

In the future, there are two obvious tasks: to collect a large amount of high quality training data, and to implement the LTR system with our proposed features (see Chapter 3). For evaluation, NDCG (see Section 2.2) could be used. Then feature selection may be conducted, if it is needed for performance reasons.

We already proposed to use numerical features, but for future work, a more complex system could be created for numerical literal values, which may match user-defined conditions like *larger than*, *smaller than*, and which is able to work with different units for users of imperial or USCS units. These constraints could be soft constraints and additionally given importance by the searcher, which then would be needed to be included in our ranking.

Another topic for future work is to include user interests to improve their satisfaction. These interests could be extracted from their search history, or be given explicitly by the user itself. Features based on these preferences then could be added to the LTR model. However, the training data needs to include user data as well. The user could additionally give significance-indications to specific parts of the query graph. These weights could be then included in the weighting for the node group embeddings (see Section 3.4).

Bibliography

[Arr12] Kenneth J. Arrow. *Social Choice and Individual Values.* Yale University Press, 2012.

[Bar17] Jesús Barrasa. *RDF Triple Stores vs. Labeled Property Graphs: What's the Difference?* `https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/`, 2017. Accessed 15.06.2020, 09:05.

[BG14] Dan Brickley and R.V. Guha. Rdf schema 1.1. `https://www.w3.org/TR/rdf-schema/`, 2014. Accessed 01.08.2020, 15:08.

[Bha] Vaishali Bhatia. *Graph measurements: length, distance, diameter, eccentricity, radius, center.* `https://www.geeksforgeeks.org/graph-measurements-length-distance-diameter-eccentricity-radius-center/`. Accessed 16.06.2020, 09:39.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. *ObjectRank: Authority-Based Keyword Search in Databases. Proceedings of the 30th International Conference on Very Large Data Bases*, 30, 10 2004.

[BM05] Bhuvan Bamba and Sougata Mukherjea. *Utilizing Resource Importance for Ranking Semantic Web Query Results.* In Christoph Bussler, Val Tannen, and Irini Fundulaki, editors, *Semantic Web and Databases*, pages 185–198, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Bur10] Chris J.C. Burges. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report MSR-TR-2010-82, June 2010.

[CC11] Olivier Chapelle and Yi Chang. *Yahoo! Learning to Rank Challenge Overview.* In *Yahoo! Learning to Rank Challenge*, 2011.

[Cla15] Jack Clark. *Google Turning Its Lucrative Web Search Over to AI Machines.* `https://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines`, October 2015. It is referenced by https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip; both accessed 04.05.2020, 15:36.

[Con18] Shane Connelly. *Practical BM25 - Part 3: Considerations for Picking b and k1 in Elasticsearch.* *https://www.elastic.co/de/blog/practical-bm25-part-3-considerations-for-picking-b-and-k1-in-elasticsearch*, 2018. Accessed 07.07.2020, 12:31.

[CZ09] Olivier Chapelle and Ya Zhang. *A dynamic Bayesian network click model for web search ranking.* In *Proceedings of the 18th international conference on World wide web - WWW '09.* ACM Press, 01 2009.

[Dea19] Brian Dean. *We analyzed 5 Million Google Search Results. Here's What We Learned About Organic CTR.* *https://backlinko.com/google-ctr-stats*, 2019. Accessed 27.04.2020, 11:15.

[DKM18] Laura Dietz, Alexander Kotov, and Edgar Meij. *Utilizing Knowledge Graphs for Text-Centric Information Retrieval.* In *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '18, page 1387–1390, New York, NY, USA, 2018. Association for Computing Machinery.

[Fri01] Jerome H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics*, 29(5):1189–1232, October 2001.

[Fri02] Jerome H. Friedman. *Stochastic gradient boosting. Computational Statistics & Data Analysis*, 38(4):367–378, February 2002.

[FUS] FUSION. *IRI, URI, URL, URN and their differences.* *https://fusion.cs.uni-jena.de/fusion/blog/2016/11/18/iri-uri-url-urn-and-their-differences/*. Accessed 15.06.2020, 08:56.

[GHT+18] ChengYue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. *FRAGE: Frequency-Agnostic Word Representation. CoRR*, abs/1809.06858, 2018.

[Goo] Google. *So funktioniert die Google-Suche | Suchalgorithmen.* *https://www.google.com/search/howsearchworks/algorithms/*. Accessed 15.06.2020, 08:52.

[HJSS06] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. *FolkRank : A Ranking Algorithm for Folksonomies.* pages 111–114, 01 2006.

[IKIT14] Shiori Ichinose, Ichiro Kobayashi, Michiaki Iwazume, and Kouji Tanaka. *Ranking the Results of DBpedia Retrieval with SPARQL Query.* In Wooju Kim, Ying Ding, and Hong-Gee Kim, editors, *Semantic Technology*, pages 306–319, Cham, 2014. Springer International Publishing.

[Jai] Saurav Jain. *Inverted Index.* *GeeksForGeeks*, *https://www.geeksforgeeks.org/inverted-index/*. Accessed 15.06.2020, 08:53.

[JK00] Kalervo Järvelin and Jaana Kekäläinen. *IR Evaluation Methods for Retrieving Highly Relevant Documents.* In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, page 41–48, New York, NY, USA, 2000. Association for Computing Machinery.

[JON72] KAREN SPARCK JONES. *A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL. Journal of Documentation*, 28(1):11–21, jan 1972.

[Kar18] Dhruvil Karani. *Introduction to Word Embedding and Word2Vec.* `https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa`, 2018. Accessed 25.07.2020, 17:40.

[KBdR16] Tom Kenter, Alexey Borisov, and Maarten de Rijke. *Siamese CBOW: Optimizing Word Embeddings for Sentence Representations.* In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 941–951, Berlin, Germany, August 2016. Association for Computational Linguistics.

[KD15] Vijay Kotu and Bala Deshpande. *Ensemble Modeling - an overview | ScienceDirect Topics.* `https://www.sciencedirect.com/topics/computer-science/ensemble-modeling`, 2015. Accessed 01.08.2020, 14:50.

[Kle98] Jon M. Kleinberg. *Authoritative Sources in a Hyperlinked Environment (HITS).* 1998.

[Li11] Hang Li. *A Short Introduction to Learning to Rank. IEICE Trans. Inf. Syst.*, 94-D:1854–1862, 2011.

[LIJ+14] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. *DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal*, 6, 01 2014.

[LK16] Felicitas Löffler and Friederike Klan. *Does Term Expansion Matter for the Retrieval of Biodiversity Data?* 09 2016.

[LM16] Minh-Thang Luong and Christopher D. Manning. *Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. CoRR*, abs/1604.00788, 2016.

[LXSL18] Zheng-Hao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. *Entity-Duet Neural Ranking: Understanding the Role of Knowledge Graph Semantics in Neural Information Retrieval. CoRR*, abs/1805.07591, 2018.

[LZ11] Yuanhua Lv and ChengXiang Zhai. *Lower-Bounding Term Frequency Normalization.* In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 7–16, New York, NY, USA, 2011. Association for Computing Machinery.

[MCCD13a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space.* 2013.

[MCCD13b] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space.* pages 1–12, 01 2013.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Stanford University, 2008. `https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html`, Accessed 20.07.2020, 09:15.

[MRS09] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Stanford Univerity, 2009. `https://nlp.stanford.edu/IR-book/pdf/06vect.pdf`, Accessed 06.07.2020,14:14.

[Net20] NetMarketshare. *Search engine market share.* *`https://www.netmarketshare.com/search-engine-market-share.asp`*, 2020. Accessed 25.07.2020, 17:10.

[NFS17] El Moatez Billah Nagoudi, Jérémy Ferrero, and Didier Schwab. *LIM-LIG at SemEval-2017 Task1: Enhancing the Semantic Similarity for Arabic Sentences with Vectors Weighting.* 08 2017.

[Noa20] Antonio Noack. *AntonioNoack/WebPageRank: v1.0.* Zenodo, 2020. DOI: 10.5281/zenodo.3931283.

[NvNvdG19] Malvina Nissim, Rik van Noord, and Rob van der Goot. *Fair is Better than Sensational: Man is to Doctor as Woman is to Doctor. CoRR*, abs/1905.09866, 2019.

[Pag98] Larry Page. *The PageRank Citation Ranking: Bringing Order to the Web.* 1998.

[Pau20] Heiko Paulheim. *RDF2Vec.org.* *`http://www.rdf2vec.org/`*, 2020. Accessed 27.07.2020, 15:40.

[Pow08] David Powers. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation. Mach. Learn. Technol.*, 2, 01 2008.

[Qat17] Qatar Computing Research Institute. *Semantic Textual Similarity < SemEval-2017 Task 1.* *`http://alt.qcri.org/semeval2017/task1/`*, 2017. Accessed 09.08.2020, 16:51.

[QL13] Tao Qin and Tie-Yan Liu. *Introducing LETOR 4.0 Datasets. CoRR*, abs/1306.2597, 2013.

[RRN+17] Peter Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. *RDF2Vec: RDF Graph Embeddings and Their Applications. Semantic Web Journal 2017*, 2017.

[Sas07] Yutaka Sasaki. *The truth of the F-measure. Teach Tutor Mater*, 01 2007.

[SCP16] Daria Sorokina and Erick Cantú-Paz. *Amazon Search: The Joy of Ranking Products. MLconf SF 2016*, 2016.

[SG01] Amit Singhal and I. Google. *Modern Information Retrieval: A Brief Overview. IEEE Data Engineering Bulletin*, 24, 01 2001.

[SGCR19] Uma Sawant, Saurabh Garg, Soumen Chakrabarti, and Ganesh Ramakrishnan. *Neural architecture for question answering using a knowledge graph and web corpus. Information Retrieval Journal*, 22(3-4):324–349, jan 2019.

[SH13] Eric Prud'hommeaux Steve Harris, Andy Seaborne. *SPARQL 1.1 Query Language.* *https://www.w3.org/TR/sparql11-query/*, 2013. Accessed 15.06.2020, 09:00.

[SS15] Irina Sergienya and Hinrich Schütze. *Learning Better Embeddings for Rare Words Using Distributional Representations.* In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 280–285, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[Swi99] Ralph R. Swick. *RDF Standard.* *https://www.w3.org/1999/.status/PR-rdf-syntax-19990105/status*, 1999. Accessed 02.08.2020, 11:09.

[VK14] Denny Vrandečić and Markus Krötzsch. *Wikidata: A Free Collaborative Knowledgebase. Communications of the ACM*, 57:78–85, 09 2014.

[VR79] C. J. Van Rijsbergen. *Information Retrieval (2nd ed.).* Butterworth-Heinemann, 1979.

[vu] vzn (https://cstheory.stackexchange.com/users/7884/vzn). *HITS and PageRank, topic drift problem.* Theoretical Computer Science Stack Exchange. Accessed 27.06.2020, 9:15, https://cstheory.stackexchange.com/q/18557 (version: 2013-08-05).

[Wat20] International Business Machines Corporation Watson. *IBM Research | IBM.* *https://www.ibm.com/watson*, 2020. Accessed 04.05.2020, 15:19, The website is all about AI and machine learning.

[WGB+18] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. *Position Bias Estimation for Unbiased Learning to Rank in Personal Search.* In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 610–618, 2018.

[wor13] *Google Code Archive - Long-term storage for Google Code Projects.* *https://code.google.com/archive/p/word2vec/*, 2013. Accessed 27.07.2020, 12:50.

List of Figures

Listings

List of Tables