Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

Language processing techniques for searching on Transparenzportal Hamburg

vorgelegt von

Katrin Caragiuli

MIN-Fakultät

Fachbereich Informatik

Sprachtechnologie (LT)

Studiengang: B. Sc. Mensch-Computer-Interaktion

Matrikelnummer: 6534868

Erstgutachter: Prof. Dr. Chris Biemann

Zweitgutachter: Dr. Lothar Hotz

Abstract

Information Retrieval (IR) systems have greatly improved over the last years. Search engines offer access to millions of documents within seconds. However, the variety of these documents and unspecific user inputs still pose some problems that restrict the number and quality of returned documents. Natural language processing (NLP) is often applied to specifically deal with linguistic difficulties. The Transparenzportal Hamburg employs a search engine for a collection of administration documents. This thesis researches NLP methods to improve the search results on the Transparenzportal from the user's perspective. The focus lies on queries that return only a few results or none. To minimize queries with spelling mistakes, the Levenshtein distance is applied as a spell-check. Also, decompounding is used to tackle queries with long and ambiguous query tokens. Especially for compounding languages such as German, compound words pose a major problem in IR. Both the Levenshtein distance as well as decompounding are evaluated with a small experiment set and compared to a control experiment. They improve the search results with statistical significance and, therefore, should be added to the Transparenzportal.

Outline

1 Introduction

In this day and age, there is hardly any question that could not be answered by trying Google. The human brain filters out a huge portion of our sensory impressions, intending to not overstimulate the individual. Similarly, search engines try to make a selection and ranking of the most relevant data for the user. That is crucial since search engines, in general, have access to a vast number of datasets. However, translating an information need into a query still requires some kind of knowledge of the user about search engines and the search engine in use particular. This includes for example, whether the search engine applies synonyms. If searching on the internet was that easy, we would not have to google tomorrow's weather for our grandparents.

This thesis focuses on ad hoc retrieval on the Transparenzportal Hamburg[1] (TP), a search engine that provides access to documents of public interest concerning the city of Hamburg. It origins from the Transparenzgesetz[2] (transparency law) (2012). Before the adoption of this law, the requests regarding public matters of Hamburg had to go through a complex application process. Since the TP was realized to simplify this request process, it is of importance that all relevant documents for a query are returned.

Document retrieval is faced with several difficulties, two of which are the main focus of this thesis. One major issue is posed by the highly specialized dataset of the TP. The vocabulary used in the documents might either not be familiar to the user, or lead to spelling errors and typing mistakes (which will be used interchangeably from here on). However, the TP does not apply spell-check, which is why faulty queries like *Elpphilharmonie* are ineffective. Another obstacle is imposed by the German language itself. The lack of consensus and the amount of ambiguity regarding some word forms make it difficult to provide every document for a query. For instance, compound words in German are prone to escalating. *Donaudampfschifffahrtsgesellschaftskapitän* is a popular example, which means *captain of the Danube Steam Shipping Company*[3]. Besides, the use of hyphens to connect words is not standardized. *Wi-fi* and *Wifi* are both correct and denote the same thing, but can be written differently nevertheless. Consequently, some search queries are only able to return a few results.

These are difficulties that contribute to the fact that many queries on the TP only match few or even no documents. On the one hand, it could be the case that there are simply no more documents corresponding to the query. On the other hand, the queries or the index could not be well-formed enough to retrieve the desired documents. Given this problematic, this thesis aims to identify queries with low result outcomes and apply two language processing techniques to improve the number of relevant search results. First, a spell-check is applied by using the string-edit metric Levenshtein distance to detect similar words [17]. Second, a decompounding algorithm is used to split compounds and add the constituents to the search. The applied techniques are each individually tested against the original TP algorithm and evaluated with an Information Retrieval (IR) specific formula, the Normalized Discounted Cumulated Gain (nDCG). The ASV-Toolbox is used for the application [2].

1. http://transparenz.hamburg.de
2. https://www.hamburg.de/transparenzgesetz/ (last accessed 12-08-2019)
3. https://en.wiktionary.org/wiki/Donaudampfschifffahrtsgesellschaftskapit%C3%A4n(last accessed 12-08-2019)

The overall structure of this thesis takes the form of nine chapters, including this introductory chapter. First, the Information Retrieval (IR) background concerning natural language processing techniques is presented. Second, the technical foundations are laid out, including a description of the Transparenzportal. Chapter four begins with a summary of the Levenshtein distance and decompounding technique. It then goes on to describe the AVS toolbox that is going to be used to carry out the experiments and the evaluation method nDCG. The fifth chapter lays out the experimental design, including the creation of the test collections. Chapter six presents the results of the experiments. A seventh chapter is included to discuss a possible application of the evaluated techniques in the TP. The last chapters contain the Summary and Conclusion and lastly the Outlook.

2 Related Work

Without search engines, finding what one is looking for on the internet would be nearly impossible. Modern information retrieval techniques go way beyond simple string matching but have to consider several challenges that natural languages present. This chapter introduces state-of-the-art techniques for improving document search results which sets this thesis into relation to previous, widely acknowledged works.

Essential knowledge of information retrieval is covered comprehensively by Manning et al. in [20]. This involves the steps a search engine has to take to process a search query. This process is illustrated in Figure 2.1. Firstly, the user starts a search query, for example, for *How manydays until Chrismas?*. This search query then is modified by the search engine before eventually retrieving results from the database.

Those modification steps are described in [20], page 32, including tokenization, stemming and decompounding. Tokenization is the basis for further linguistical processing. It splits the input into so-called tokens. In the case of the given example, "How manydays until christmas" would get split into the tokens |Howmany| |days| |until| |Chrismas| |?|. Stemming describes the process of reducing a word to its word stem, intending to match all related words. Applied to the tokenized example, the stemming result would be |Howmany| |day| |until| |Chrismas| |?|, with *days* being reduced to *day*. This is a standard approach in IR, for example used by Google [33], and is also applied by the Transparenzportal (TP). In addition, a spell-check would be applied at this point. This could be done by implementing the Levenshtein-Distance, invented by V.I Levenshtein [17]. This technique detects string similarities. *Chrismas* would then be exchanged with *Christmas*. In [20], page 46, Manning et al. also mention how compound-splitting can be beneficial for systems that deal with the German language. Amongst other things, it could be used in English to separate words that were accidentally combined by the user by leaving out a whitespace character. In the given example, *Howmany* would be separated into *How* and *many*.

Obviously, the query input is not the only text that has to be handled. The same techniques as mentioned above can be applied to the texts that eventually form the index. This index is usually realized through an inverted index, a standard data structure in IR. An inverted index is a data structure that contains a mapping from keywords to documents, as described by Carstensen et al. in [5], page 588.

Apart from the standard techniques, other different approaches have to be applied to expand a search grammatically, but also semantically. A way to achieve a semantic refinement of a query is to expand it. Croft et al. differentiate query expansion between automatic and semi-automatic expansion, the latter meaning that user interaction is involved. This bachelor thesis will focus on automatic query expansion. Furthermore, Croft et al. point out that the use of synonyms, respectively a thesaurus alone does not suffice to conquer the challenges that query reformulation creates [9], page 200.

However, the search engine does not only look for matching results but also adds a score to each document, depicting how much it fits the query. This score is then part of the ranking process, that ideally presents the documents with the highest scores on top. Additional rules can be applied, for example which sorting, ascending or descending, should be applied to documents with the same score. The TP handles this by sorting by name alphabetically.

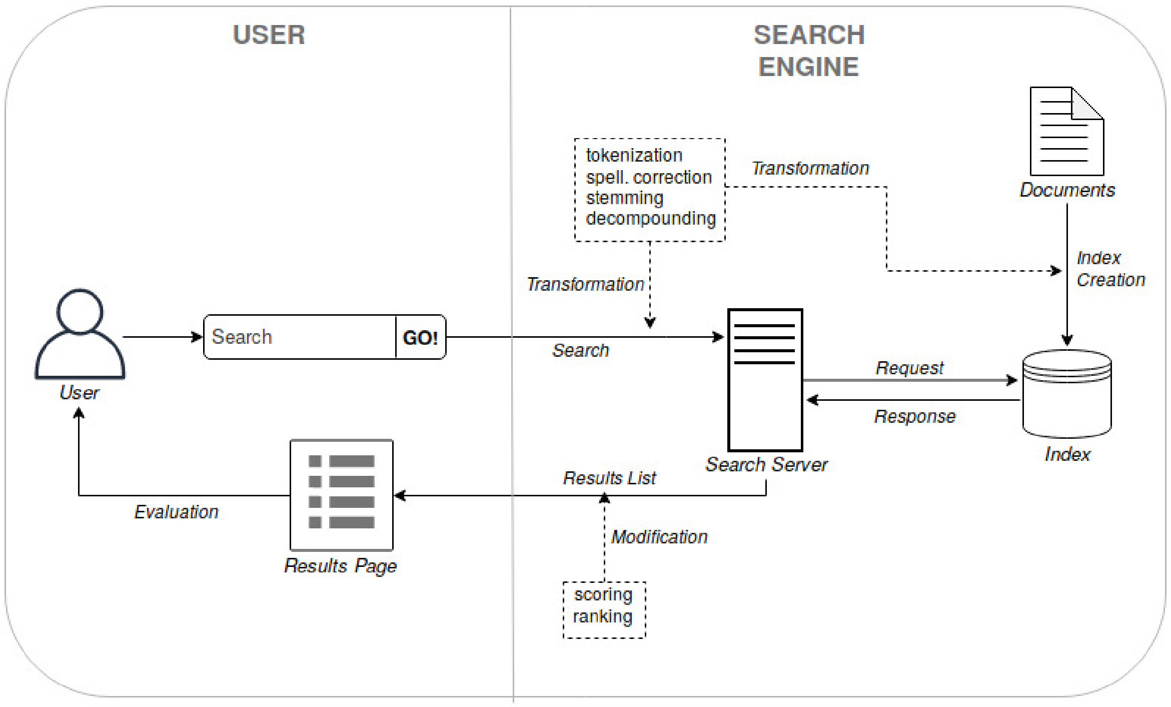


Figure 2.1: A flow chart of how a search engine might process a user query. The user starts a query, the search engine transforms the input with tokenization, spell. correction, stemming and decompounding and retrieves results from the inverted index. The same techniques are applied to the documents that form the index. The results that are returned from this index get displayed on the website and the user can choose to modify their input and place another query.

Jerry R. Hobbs described the phenomenon of the "topic drift", which should in later terms gain significance for information retrieval on the internet [28]. The so-called "query drift" describes the dilemma, that query expansion, when done suboptimal, may sway the results in a completely different direction than initially intended by the user. Mitra et al. describe some techniques to avoid a (hard) query drift, in [21]. They state that the most evident approach would be to improve the search results at the top of the list.

Another approach would be to consult the query logs of a search engine. Albakour et al. have taken to using query logs to avoid the cost of involving user interactions. By analyzing a huge set of search logs, they are able to train different algorithms with a technology called "AutoEval" and produce query modification suggestions [1]. T. Joachims also uses clickthrough data to successfully train a learning algorithm to improve search retrieval quality [14].

For all these actions taken to improve IR, an evaluation technique has to be applied to confirm their effectiveness. Manning et al. [20], page 5, characterize the fundamental methods in IR called "precision" and "recall". Precision defines the percentage of documents returned that are relevant and recall defines the percentage of relevant documents that are returned. They conclude that recall needs to be above a certain threshold while the number of false positives should be kept as low as possible. Given the fact that C. Carpineto and G. Romano found out that about 54% of the users view only one results page, and 53% only enter one query, the measures taken in this thesis will concentrate on increasing the recall, while measuring the precision of the top returned documents.

Considering the work done to date, this thesis applies a selected number of techniques to influence and evaluate the search results of the Transparenzportal. First of all, a spell-check will be applied by using the the Levenshtein-Distance. Second, decompounding will be used to take advantage of the grammatical nature of the German language. This thesis only focuses on queries with less than 5 results. As a result, query drift is not the main concern but will be adressed by trying to improve the top results and manually evaluating the relevance of the search results and evaluating it. This thesis cannot make good use of the session information in the query logs, since the session information has not been recorded. However, the query logs are analyzed to extract the different categories of search queries and use them to evaluate the search engine adjustments made. It also aims to implement two different natural language processing techniques by trying to avoid obvious IR mistakes. The chosen evaluation method is the Normalized Discounted Cumulated Gain (nDCG), which is described by Manning et al. as becoming more and more popular and will be explained later in this thesis [20]

3 Foundations

The previous chapter gave an insight into the state of the art of query processing and query refinement. Consequently, the technologies that form the foundation of the Transparenzportal (TP) are laid out in the following chapter. Therefore, the Transparenzportal is introduced in detail to set this thesis in context. Furthermore, the TP mechanisms for Solr query (and index) processing are explained, including the parameters that are set for the live system.

3.1 Transparenzportal

The Transparenzportal Hamburg (TP) is the result of the Transparenzgesetz Hamburg (2012), a law that obliges the town administration of Hamburg to make administrative data accessible to the public. Apart from this, it also contains the data that has previously been made public in the Open Data Portal Hamburg. On the first of October 2014, the Transparenzportal went live and has since then collected more than 100,000 documents [1].The displayed data includes, for instance, geography data, building permits, and contracts. All data that falls within the scope of the disclosure obligation has to be stored for 10 years[2].

The data can be accessed through a search which is realized using an expanded version of The Comprehensive Knowledge Archive Network[3] (CKAN). CKAN is a data management system written in Python with an Apache Solr[4] index and a Postgres database. The Solr index makes it possible to search the data and additionally offers filter possibilities that are the basis for the expanded search. It provides harvesting possibilities to receive metadata from e.g. ALLRIS[5] (a citizen information system) and also provides the Transparenzportal Application programming interface (API). To be able to perform a full-text search over the documents, the Apache Tika™ [6]toolkit is used. It extracts the text of everydocument to render it indexable. The Transparenzportal website itself is implemented in the design of hamburg.de and therefore does not use the CKAN interface.[7]

3.2 Solr

Solr (pronounced "solar") is an enterprise open-source search server, similar to Elasticsearch[8], that uses Apache Lucene. Created in 2004 by Yonik Seeley, Solr is used as a search engine by a variety of companies, for example Netflix [30], page 4.

1. http://transparenz.hamburg.de/transparenzgesetz-hamburg/11931448/transparenzportal/ (last accessed 02-08-2019)
2. https://www.hamburg.de/bkm/transparenzportal/ (last accessed 20-07-2019)
3. https://ckan.org/ (last accessed 02-05-2019)
4. https://lucene.apache.org/solr/ (last accessed 03-04-2019)
5. https://www.cc-egov.de/de-de/produkte/allris (last accessed 04-04-2019)
6. https://tika.apache.org/ (last accessed 01-03-2019)
7. https://media.ccc.de/v/31c3_-_6582_-_de_-_saal_2_-_201412281600_-_das_transparenzportal_hamburg_-_lothar_hotz (last accessed 24-04-2019)
8. https://www.elastic.co/de/products/elasticsearch (last accessed 02-03-2019)

Solr creates an inverted index that is used to search through documents. Tokenizers and filters are applied to the index and the search query to expand the query and therefore generate better results. In the case of the Transparenzportal, a tokenizer is applied to break the input query up into tokens. Afterwards, a number of filters is applied to manipulate the query tokens, e.g. on that adds synonyms to the query. The following analyzers are used: Whitespace Tokenizer, Synonym Filter, Word Delimeter Filter, Lower Case Filter, Snowball Porter Filter and ASCII Folding Filter. The following subsection will explain these filters in detail.

3.2.1 Solr processing

The way data is processed in Solr has a huge impact on the chance of success for the search results.

In German, all nouns are capitalized. Consequently, if a German noun, *Bauplan* (building plan), for instance, is not converted to lowercase in the index, a search for *bauplan* might not detect the corresponding documents.

The data that is added to the Transparenzportal is mostly unstructured, for example pdfs containing protocols of board meetings. To be able to search through this data, Apache Tika is used to extract the content. The resulting text then has to be processed; this step is called analysis. During the analysis, a number of tokenizers and filters are applied. On each step of the way, a stream of tokens is produced and processed, with the end result being a number of terms that are then ready to be indexed. Solr creates a so-called inverted index this way, which maps terms to a list of documents in which they occur, comparable to the index in a book [27], page 47.

For the Transparenzportal, a bundle of successive steps are defined for different field types. The first step is a tokenizer that produces a stream of tokens. The end result of the last Token Filter are the terms that are indexed. Regarding the query input string, the following tokenizers and filters are applied at query time.

The Whitespace Tokenizer splits the text at the whitespace, the output is a token stream. Afterwards, the Synoym Filter takes action, with the help of a text file that contains all words that are synonymous with each other. The TP has not filled this text file with new synonyms yet but has taken over the synonyms from the Open Data Portal. Given the fact that the Whitespace Tokenizer leaves delimiters in place, the Word Delimiter Filter is applied afterwards. Taking the example *wi-fi*, it gets resolved to wi and fi. The Lower Case Filter then transforms the text to lower case. Afterwards, the Snowball Porter Filter, the stemming filter, is applied. Stemming means, that every word gets reduced to their word stem form, for example *spielerisch* (playfully) to *spiel* (play). At the end, the ASCII Folding Filter is applied, so that words with accented and not accented characters are valued as the same, e.g. Eugène and Eugene. The Unicode words get converted to ASCII.[9]

To illustrate the process, take for example the query *Olympische-Spiele Hamburg* (olympic games Hamburg) . As Table 3.1 shows, the query gets processed to *olymp spiel hamburg* at query time. Every row holds the output of the corresponding filter, every column denotes their position in den token stream.

9. http://www.pathbreak.com/blog/solr-text-field-types-analyzers-tokenizers-filters-explained(last accessed 2019-02-02)

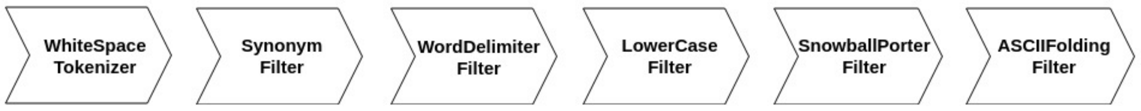


Figure 3.1: The sequency of applying tokenizers and filters to the input query. The filters are applied in the following order, after the Whitespace Tokenizer: Synoym Filter, Word Delimiter Filter, LowerCase Filter, Snowball Porter Filter and ASCII Folding Filter.

token position	1	2	3
White Space	Olympische-Spiele	Hamburg	
Word Delimiter	Olympische	Spiele OlympischeSpiele	Hamburg
Lower Case	olympische	spiele olympischespiele	hamburg
Snowball Porter	olymp	spiel olympischespiel	hamburg
ASCII Folding	olymp	spiel olympischespiel	hamburg

Table 3.1: This Table illustrates the tokenization and filter application on *Olympische-Spiele Hamburg* (olypic games Hamburg) at index time. The result is the query tokens *olymp, spiel olympischespiel* and *hamburg*.

A similar process has to be applied to the document texts at index time, see Figure 3.2. The biggest difference is, that no synonym filter is started, as it would bloat the index. Another small difference shows in the Word Delimiter Filter, where words (and numbers) are not catenated. This means that *wi-fi* would not only resolve in *wi* and *fi*, but also in in *wifi*. The same happens with numbers. This way, words that could be written with or without a delimiter get matched on the same word, for example the queries for *wifi*, *WiFi*, *wi-fi* and *wi+fi* would all match.[10]

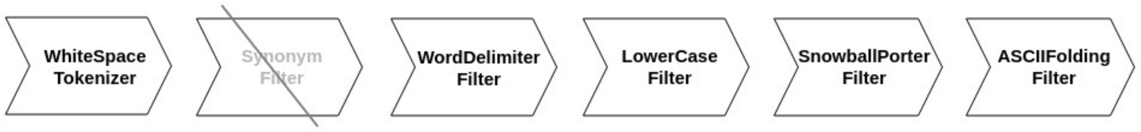


Figure 3.2: The sequency of applying tokenizers and filters to the index documents. The filters are applied in the following order, after the Whitespace Tokenizer: Word Delimiter Filter, LowerCase Filter, Snowball Porter Filter and ASCII Folding Filter.

Table 3.2 illustrates the the query "Olympische-Spiele Hamburg". As Table 3.1 shows, the query gets processed to "olymp spiel olypischespiel hamburg" at index time. The difference in processing regarding the Word Delimiter Filter will be adressed later in this thesis.

10. https://lucene.apache.org/core/4_7_0/analyzers-common/org/apache/lucene/analysis/miscellaneous/WordDelimiterFilter.html (last accessed 11-08-2019)

token position	1	2	3
WT	Olympische-Spiele	Hamburg	
SF	Olympische-Spiele	Hamburg	
WDF	Olympische	Spiele	Hamburg
LCF	olympische	spiele	hamburg
SPF	olymp	spiel	hamburg
AFF	olymp	spiel	hamburg

Table 3.2: A Table that illustrates the tokenization and filter application on "Olympische-Spiele Hamburg" at query time. The result is the query tokens *olymp, spiel, hamburg*.

3.2.2 Solr parameters

Solr offers a variety of parameters to adjust the search results. The following section describes the most important parameters set by the TP, to understand how the search results come about. Listing 3.1 shows an excerpt of query parameters concerning the search are set on the Transparenzportal.

```
'defType': 'dismax',
'qf': 'name^4 title^4 tags^2 groups^2 text',
'mm': '100%',
'sort': 'score desc,title_sort asc'
```

Listing 3.1: An excerpt of the query parameters set on the TP. The query parser *DisMax*, the query fields name^4, title^4, tags^2, groups^2 and text with boost factors (^), the minimum must match factor of 100% and the setting to sort by score descending and title_sort ascending

When beginning to use Solr, it has to be decided which query parser to use. The QueryParser transforms the query in a way that Solr Lucene can understand it, which can lead to quite different search results depending on the choice [27], page 131. The TP uses the *DisMax* Query parser. *DisMax* stands for "disjunction maximum" [27], page 140. Compared to the Standard QueryParser it shows very few syntax errors and is suitable for simple user enquiries.[11] It also comes with the addition of a weighting function, called Query Function (qf) parameter. Through this function, specific fields can be given a boost factor to increase the score of the corresponding documents. The weighting given by the TP can be seen in Listing 3.1. Title and name are basically the same, in that the name is written in lowercase letters and connected with hyphens, e.g *title=Einredeverzichtserklärung Elbphilharmonie* and *name=einredeverzichtserklaerung-elbphilharmonie*. The tags are added to each document at index time to aid the search. A document concerning yourth welfare does, for that purpose, contain the tag *Jugendhilfe in Hamburg* (youth welfare). Groups are assigned at the same time, they represent the categories that can be filtered for on the TP website. Both are taken on during the import of documents into the TP. Tags and groups will not be subject of change in this thesis.

11. https://doc.lucidworks.com/fusion-server/4.1/solr-reference-guide/7.2.1/the-dismax-query-parser.html (last accessed 2019-02-02

Generally, the default handling of boolean queries is the locical OR concatenation. However, with the *DisMax* query parser, this is converted to a logical AND by applying the mm parameter with 100%. The mm clause stands for "minimum must match" and restricts the number of queries that must match in the result document. Having set the parameter to 100% means that for example the search for *2017 Bebauung Wandsbek-Gartenstadt* would not return results if not the exact three words appear in a document [27], pages 141-142.

Every document gets a score assigned, measured by how good they match the given query. Consequently, the display of the search results is sorted by score descending and title ascending. The TP uses the Default Scoring of Solr. The scoring is mainly influenced by the term frequency and the inverse document frequency. The term frequency weights a document more if the term appears more often in it. The inverse document frequency, however, weighs a document more if it appears in fewer documents [11], pages 65-67.

4 Methods

The previous chapter describes the technologies behind the TP and specifically, how Solr is used for the document search. With that, the context of this thesis is set. This chapter further introduces language technologies and principles of information retrieval. These are used to complement and improve the search results of the TP. First, Levenshtein distance and decompounding are explained. Then, the ASV toolbox with some of its tools is introduced. Last, a measurement technique, the Normalized Discounted Cumulative Gain (nDCG), is explained which is the chosen evaluation measure.

4.1 Language Processing Techniques

In this section, the Levenshtein distance and decompounding are explained in detail.

4.1.1 Levenshtein distance

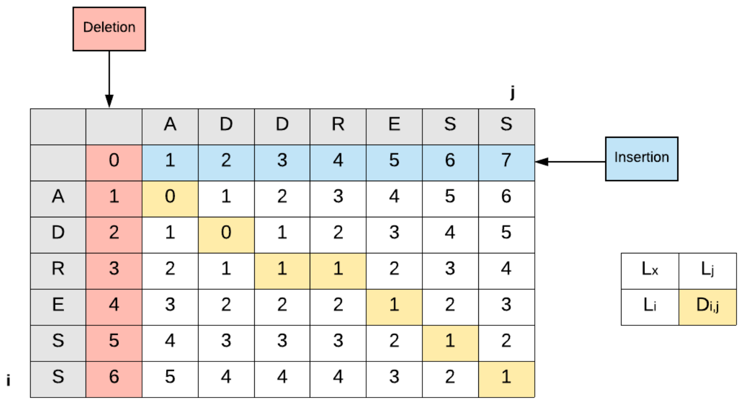
The Levenshtein distance (LD), is a string-edit distance metric that determines the minimal distance between two strings. It originates from Vladimir Iosifovich [17]. The distance is defined as the number of edit operations necessary to turn one string into another. The three available edit operations are insertion, deletion, and substitution. Transpositions, meaning the swapping of adjacent characters, are not included in this algorithm. For every executed operation, the value one is added to the LD. Weighted versions of this algorithm may add different values to the operations [23].

In dynamic programming, the distance is computed by calculating a matrix that holds the edit distances between between prefixes of growing length. An example for the edit distance of *address* and *adress* is shown in 4.1. The first row can be seen as only using insertion, the first column as only using deletion operations [24]. The computation can be reduced to the following equation:[1]

$$\begin{aligned} Lev_{ij} &= \min(L_i + 1, L_j + 1, L_s + C_{\text{substitution}}) \\ C_{\text{substitution}} &= \begin{cases} 1, if S_1[i] \neq S_2[j] \\ 0, else \end{cases} \end{aligned} \tag{4.1}$$

The Equation 4.1 shows that the calculation for the LD at the position i,j in the matrix results from the minimum of the minimum of the LD around it. The substitution value depends on whether the two characters in question are the same (+ 0) and no operation is necessary, or different (+ 1). A visualization of this is depicted in Figure 4.1. Creating the word *address* adds up the operations *insert* and the deletion of the word *adress* adds up the operations *deletions*. The calculation for the LD at the first position (A-A) results from calculating min = (2,2,9) = 0.

1. http://ntz-develop.blogspot.com/2011/03/fuzzy-string-search.html (last accessed 06-08-2019)



		A	D	D	R	E	S	S
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
D	2	1	0	1	2	3	4	5
R	3	2	1	1	1	2	3	4
E	4	3	2	2	2	1	2	3
S	5	4	3	3	3	2	1	2
S	6	5	4	4	4	3	2	1

Figure 4.1: A Levenshtein distance matrix comparing the words *address* and adress. The result is a distance of one.

Different use cases for the LD have been for instance plagiarism detection [31], measuring dialect pronounciation differences [12] or adapting the LD for error-correcting barcodes for multiplexed DNA sequencing [4].

String matching algorithms are applied to words of certain string lengths. Not every word is applicable for every algorithm. For example, a string problem that could arise with strings that are too short is a false positive matching. Short strings, e.g. *kale*, can arise issues with false positive matching. Applying the LD to this string would generate the results *tale* and *sale* which are in close string distance to the original string, but are far away content-wise. To avoid these false positives as much as possible, this thesis is going to adopt the Levenshtein tolerance distance proposed by Meyer and Lisbach in [18], page 94, see Table 4.1.

query token length	tolerated Levenshtein distance
< 5	0
5-8	1
9-12	2
> 12	3

Table 4.1: This table is adapted from [18], page 94. The table compares the length of a query token and the resulting tolerated Levenshtein distance. The three conversion operations are applied to all search query tokens with more than five characters. Every query with a length of five to eight characters is matched with a distance of one, every word with a distance of nine to twelve is matched with a distance of two and every string-length higher than twelve is matched with a distance of three.

4.1.2 Decompounding

Decompounding (also known as compound decomposition or compound splitting) denotes the method of dividing compound words into their respecting constituents. A compound is a complex word that is composed of other words. German and Scandinavian languages are some of the languages that often use compounds [25]. They can either take shape in one word, for example, *birthday* or be hyphenated, for example *so-called* [8], page 158. Compounding in German is a very complex and diverse matter, as combinations of different word forms are possible. An example for a noun-noun compound is for example*behindertenbeförderung* (transport for disabled persons). A verb-noun compound, for example, is *Suchmaschine* (search engine) [15].

Dictionaries do not necessarily need to contain a compound to make it valid for usage in the German language. The advertisement industry for example tends to create new compounds such as *Frischekick* (translates directly to "fresh kick") [19]. As [32] describe it, a semantic relation is sufficient in German to create a whole new noun.

In general, compound splitting does not work with compositional compounds, which means compounds which cannot be explained by the meaning of their compound words. For instance *blackmail* can be split into *black* and *mail*, however a query expansion with *black* and *mail* would not return satisfactory results. A. Schiller describes this as over-segmentation [25]. Monz et al. also report this disadvantage and point out the slight topic drift that arises with query expansion. This difficulty is accepted in this thesis.

The various compound kinds also differ in whether they contain an interfix or not. An interfix is a phoneme that appears within a compound word as a connector, for instance for grammatical reasons. Some possible German interfixes are -s-, -(e)n-, -es-, -e-, -er-, -ens- [16], page 86. For example, *Präsidentschaftswahl* (presidential election) contains an interfix "s", *Präsidenschaft + s + wahl*. This has to be kept in mind when dealing with compound splitting [26].

Research on decompounding in compounding languages has discovered the positive impact that decompounding can have in IR. Braschler et al. report that decompounding results in higher text retrieval improvement than stemming [3]. A study by Chen et al. found an improvement of German monolingual retrieval by 11.47% with decompounding [8].

4.2 ASV Toolbox

The ASV Toolbox is a collection of natural language processing tools created by Chris Biemann, Uwe Quasthoff, Gerhard Heyer and Florian Holz [2]. ASV is an acronym for *Automatische Sprachverarbeitung* in German (Natural Language Processing). It contains twelve independent tools that are also available as single modules. It is written in Java and all tools can be assessed via a GUI version or incorporated into a project with Java Archives (.jar files). In particular, the tools Levenshtein and Baseforms are relevant for this thesis.

4.2.1 Levenshtein Tool

The Levenshtein Tool was created by Stephan Schubert. It contains the option to return a list of alternative words given a word, the required distance and the corresponding Directed Acyclic Word Graph (DAWG). The DAWG can either be chosen from a selection of 15 pre-trained DAWG for a number of languages, including German or English, or a self-created graph can be added from file. Additionally, the tool offers the possibility to create a custom DAWG.

For our implementation, we make use of a custom created DAWG. A DAWG is a smallest suffice automaton of words, also known as an Acyclic Deterministic Finite Automata (ADFDA) [7], page 87. It is, in effect, a finite state automaton that accepts words as input [10], page 3. The construction of said DAWG is performed by using the ASV Toolbox. The training of the DAWG requires a word list, with each word in its own row. Given this DAWG and a weight, the ASV Toolbox returns a list with all possible alternative words, sorted in alphabetically order ascending.

4.2.2 Baseform Tool

Regarding decompounding, this thesis uses the *Baseform Tool* of the ASV Toolbox by Thomas Eckart and Dirk Goldhahn. It is designed to split noun compounds at offers four different options: *compound noun decomposition*, *compound noun training*, *baseform training* and *baseform reduction*. The *compound noun decomposition* is going to be used in this thesis. It can either be used with an own classification tree, or thr pre-trained German tree. In this case, the German tree is going to be used for time reasons.

The tool is implemented by using a Compact Patricia Trie (CPT). PATRICIA is an acronym for *practical algorithm to retrieve information coded in alphanumeric* [22]. A Trie is a data structure for storing strings, where each prefix is a node. A compact version of this tree is a Patricia Trie, where nodes without a branch are combined [**rais1993limiting**].

To split a word into its constituents, CPT classifiers recursively traverses the word from both ends. They split the words until no segmentation is possible anymore. The interfixes mentioned in Chapter4 are getting pruned and the constituent words reduced to their base form by applying a part-of-speech-independent base form reducer[35].

4.3 nDCG

The Normalized Discounted Cumulative Gain (nDCG) [13] is a measurement to asses the quality of search results. It possesses a few advantages in comparison with other ranking measurements. First, it allows a ranking grade that expands beyond binary [34]. Especially in the context of information retrieval, this characteristic is of importance. While documents might be graded as relevant or irrelevant, many documents fall into the category of „somewhat relevant“ or „somewhat irrelevant“, which needs to be considered when evaluating an IR system.

Second, the nDCG integrates the position of each document in the final score. The DCG, which is part of the nDCG calculation, is based on the assumption that the higher the rank of a document is, the higher up it should appear in the result list[34]. The „discounted gain“ denotes the decreasing weight of the position of each document [13]. These discounted gains are eventually cumulated.

The DCG is calculated in the following way:

$$\mathrm{DCG}_p = \sum_{i=1}^{p} \frac{relevance_i}{\log_2(i+1)} \tag{4.2}$$

The nDCG is generated by normalizing the DCG. The DCG is first calculated and then divided by an ideal DCG, the iDCG. The iDCG is calculated by assuming an ideal ranking. For this thesis, the ideal is adopted that a result list with only *good* documents is ideal. Without this normalization, the DCG would just a number that is difficult to compare with other results. The nDCG results in a number between zero and one, with one signifying ideal results and zero signifying that no relevant results were returned. Anything in between is reflected on the score between zero and one.

The nDCG is calculated in the following way:

$$\text{nDCG}_p = \frac{DCG_p}{iDCG_p} \tag{4.3}$$

To illustrate the calculation of Equation 4.3, Table 4.2 shows an example calculation. Every returned document for the fictional query *airport security* is graded by relevance on a scale of three to zero, where zero means *nonexistent* and three means *good result*. The score for each result is calculated an the adds up to the DCG. Lastly, it gets normalized by dividing through the ideal DCG.

rank	query result	relevance	score
1	airport security system	3	9.97
2	airport cleaning service	1	2.1
3	airports do not have enough security	3	4.98
4	-	0	0.00
5	-	0	0.00

DCG	17.05
iDCG	29.38
nDCG	0.58

Table 4.2: The table contains a column for the rank of the document (1-5), the query result in form of the document title, the graded relevance of the document and the calculated relevance score. Three documents with the relevance of tree, one and three are returned. The last two query results are nonexistent and are graded a zero. They result in a combined DCG score of 17.05. Divided by the iDCG of 29.38, the nDCG results in 0.58. The iDCG is calculated by assuming that the best possible results would be five documents with the relevance of 3.

5 Experimental Design

The previous chapter described methods with whom the document search of the Transparenzportal (TP) may be improved. Following on that, this chapter dives into how exactly these methods can be applied to the TP search. Furthermore, the evaluation of the impact of those methods on the quality of search results is designed. Therfor, the collection and processing of test data from actual query logs is described. Then, the prototype for evaluating the search result changes is shown.

5.1 Procedure

The general idea behind the experiment setup is as follows. For every language processing method (including a control experiment), every query result gets manually evaluated regarding the fit to the query. The results are individually evaluated on a scale of good, okay, bad and nonexistent. To guarantee a homogenous evaluation, all tests are carried out by one person, the author. This could lead to a confirmation bias, which cannot be avoided given the test setup. The expanded queries are given query weights. This should ensure that documents that match the original query get ranked at the top, by assigning them a higher weight than the others.

For the Levenshtein distance (LD), the Cartesian product is calculated and added to the query with a logical OR. Every query that is not the original query gets assigned a weight of 0.5, as in[8], page 157. For instance, the query *Immobilienmarktbericht* (property market report) is expanded in the following way:

$(immobilienmarktbericht)^{100.0}\ OR\ (immobilienmarktbericht)^{0.5}\ OR\ (immobilienmarktbereicht)^{0.5}$
$OR\ (immobilienmarktberichts)^{0.5}\ OR\ (immobilienmarktberichte)^{0.5}\ OR\ (immobilenmarktberichte)^{0.5}$
$OR\ (immobilienmarktberichten)^{0.5}\ OR\ (immobilienmarktberichtes)^{0.5}$

The LD gets only applied at query time. The aim is to improve the query input of the user to match documents in the database. Before creating these alternatives and adding them to the query, the question remains whether the Levenshtein distance should be applied before the Solr tokenizers and filters processing steps described in Chapter 3, or afterward. The application of the Levenshtein distance would be considerably less effective if applied after stemming. This is the reason why the LD gets applied to the original query, and not the stemmed version. The alternatives are calculated and added to the query before handing it over to Solr.

The addition of the alternative queries to the original query has to be handled carefully since the query should not become too complex. Given the case of a one-word query, the alternatives are just added to the original query as a logical OR. For queries with more than one word, the Cartesian product of all words generated by LD is used as the query. It returns query combinations of all possible search words, including the original query. To ensure that the documents matching the original query are ranked higher, the original query gets assigned a weight of 100. All other queries get the same weight of 0.5. Combinations of queries as in the following example arise:

(georeferenzierte AND rasterdaten)$^{100.0}$
OR (georeferenzierte AND rasterdaten)$^{0.5}$ *OR (georeferenzierte AND rasterkarten)*$^{0.5}$

OR (georeferenzierte AND laserdaten)$^{0.5}$ *OR (georeferenzierte AND wasserdaten)*$^{0.5}$
OR (georeferenzierte AND halterdaten)$^{0.5}$.

Regarding decompounding, an easy approach would be to add the decompounded parts to the query. However, given the Solr "minimum must match" requirement of 100%, just adding the compound parts to the query is not effective. When searching for *immobilienmarktbericht* (property market report), the query would not benefit from *immobilie* (property) , *markt* (market) and *bericht* (report), as it would look like this: *Immobilienmarktbericht* AND *immobilie* AND *markt* AND *bericht*, which would be entirely in contrast to our goal. Also, forming the Cartesian product the same way as with the Levenshtein query expansion would be equally counterproductive. Instead, subsets with the decompounded parts are formed. Additional weights are applied to be able to rank queries that contain more of the decompounding parts higher than the ones with less decompounding parts. It is assumed, that the former would be more relevant for the query. As a heuristic, a query got assigned a weight corresponding to the number of query tokens in it. For example, a query with three query tokens gets a weight of three. Regarding the query *Immobilienmarktbericht*, the expanded query then looks like this:

$(immobilienmarktbericht)^{100.0}$ *OR* $(immobilie)^{1.0}$ *OR* $(markt)^{1.0}$ *OR* $(immobilie\ AND\ markt)^{2.0}$ *OR* $(bericht)^{1.0}$ *OR* $(immobilie\ AND\ bericht)^{2.0}$ *OR* $(markt\ AND\ bericht)^{2.0}$ *OR* $(immobilie\ AND\ markt\ AND\ bericht)^{3.0}$

Before being able to start the experiments, an alteration of the Solr parameters has to be made. It was originally planned to use the exact Solr parameters from the Transparenzportal to be able to implement the methods as unproblematic as possible, given successful results.. However, the *DisMax* query parser does not offer an uncomplicated option to apply weights to query tokens. It is only possible to indicate whether a query token has to be included or whether it is optional for the search. A reasonable approach to tackle this issue was to use the *Extended DisMax* (eDisMax) query parser instead. As the name implies, the *eDisMax* query parser extends the *DisMax* query parser and therefore can be used interchangeably. The general procedure for evaluating the query results is listed below:

1. For every query, the resulting documents are shown in the web application.
2. Every document gets evaluated on a scale ranging from good to bad. If a document has already been evaluated, the response is retained.
3. After every query is done, the nDGC for every query is calculated.

The evaluation of information retrieval systems is driven by assessing information retrieval effectiveness. Manning et al. emphasize that the relevance of a result should be rated regarding the expressed information need, and not the query [20], page 152. Hence, the results of the experiments are evaluated as *good* if the document matches the information need. For example, if the query *immobilienmarktbericht* actually returns the property market report of the town Hamburg, it is evaluated as a *good* result. If it is just mentioned in a document, the content of the information given about it determines wether an evaluation is going to be *okay* or *bad*. A document that only contains a query word in the Appendix is clearly *bad*. Considered for the evaluation were the title of the document, the fulltext, and the original document. Unfortunately, a lot of URLs for the original documents were outdated and therefore not accessible. If none of the former sources explain the match, the tags and groups are consulted.

The training word list used in this case is derived from the TP full texts, so the sum of the texts of all documents contained in the TP. When creating this list, two different factors have to be taken into consideration. First, most user queries on the TP contain only nouns or combinations of nouns. Second, the whole point of creating a custom DAWG is to be able to catch those words that are peculiar for the TP. Otherwise, using a pre-trained DAWG for German would suffice. An example of those nouns is be *Baumkataster* (tree register). A simple regular expression applied to the full text corpus extrudes all nouns. A part-of-speech-tagger would resultd in a similar result, a regex is used i this case as a simpler measure. To verify the correctness of this custom DAWG, the ASV toolbox GUI is used, a screenshot of the tool is shown in Figure 5.2, using the misspelled input *baumkatster* (misspelled "tree register"). With a distance of one the spelling *baumkataster* is proposed, which is correct.

5.2 Test Set

This section describes the creation of the test set used in the experiments. The test set should contain a selection of queries with few search results, to be able to assess the improvements of the language processing techniques. As the first step, the logs of the TP are analyzed to extract all queries. Afterward, an experiment test collection is created based on the original queries in the query logs.

5.2.1 Query Log Analysis

All TP logs were collected from October 2014 to August 2018. This analysis is the first step to identify user queries that produce less than five results, and which are, therefore, applicable for the following experiments.

5.2.1.1 Collected Data

From 2014 on, user queries were recorded in the following form:

```
0.0.0.0 - - [04/Jun/2018:11:47:25 +0200]
"GET /?q=S-Bahn+Verkehrsvertrag
&sort=score+desc%2Ctitle_string+asc
&esq_not_all_versions=true HTTP/1.1" 200 15463
"http://transparenz.hamburg.de/"
"Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko"
```

As shown in the listing above, the recording starts with an Internet Protocol (IP) address and the time of access. The IP address, however, cannot be assigned to specific users, because no unique IP addresses are stored. Instead, there is a range of IP addresses that signifie whether the query has been placed from inside the cities administration network, and the IP address 0.0.0.0 that signifies access from the outside. In addition, the logs contain the query itself, the kind of sorting was applied, whether the user chose to search within the newest TP data set and other information, including which web browser was used.

All queries are analyzed collectively to prevent the domination of time-dependent query terms. Even though the queries from 2014 might differ greatly from the ones from 2018, it does not diminish their value for the evaluations in this thesis.

5.2.1.2 Analysis

The following analysis is based on the analysis steps proposed by Anne Chardonnens and Simon Hengchen [6]. In their paper, they differentiate between five analysis steps for large log files: collecting, parsing, grouping, cleaning, and clustering, as described in Chapter 2.

5.2.1.3 Collecting

The recorded Solr logs are not limited to user query logs, but also contain error logs and recordings regarding feed requests. As a result, the whole log set contains 14 GB of data.

5.2.1.4 Parsing

Since the collection of data is not needed in its entirety, a parsing step needs to be applied. First, all logs that did not contain queries (GET /?q= and GET dataset?q) are removed. This includes the feeds requests, as they would skew the distribution significantly. Second, only the IP address, the timestamp and the query tokens of each query are kept. Also, it is not differentiated between the individual IP address origins. In preparation for the next steps, the timestamps are cut off at the minute mark. This step only keeps logs that contain queries, hereinafter called "query logs".

5.2.1.5 Grouping

In the next step, search queries that should not be counted more than once are grouped. A python script was applied, making use of the "group by" function of the python package Panda[1] [6]. The aim of this step is to take out requests that otherwise unnecessarily inflate the query count of specific queries. This serves the purpose to avoid counting the same input of one user multiple times, as it might be the case with a frustrated user. Also, this way the heavy influence of bots on the query count can be diminished, including production tests from the TP developers themselves. For example, one of the most used queries when testing the TP is *Kindergarten*. It was used 165,339 times out of 330,808 in 2017.To be able to identify those queries, a heuristic is applied: search quries recorded within one minute are treated as identical . Silverstein et al. use a cutoff of five minutes, however in this case one minute seemed justified after examining the query logs [29].

Before going through the cleaning process, only the actual queries and the corresponding timestamps are kept. In addition, the value one was added to every query, representing the query count. An excerpt of what the data locked like at this point is shown in Table 5.1, the plus signs and %C3%BC encoding show that a cleaning step is necessary. The query *stoferkamp+* shows

1. https://pandas.pydata.org/ (last accessed 06-06-2019)

that the user added an unnecessary whitespace, probably out of accident. Overall, the search data was reduced to 796.249 queries (33.5 MB).

timestamp	query	count
2014-10-12 06:34:00	stoferkamp+	1
2014-10-12 06:35:00	Stoferkamp	1
2014-10-12 06:35:00	Baugenehmigung+Eimsb%C3%BCttel	1

Table 5.1: This table contains the result of the third step of the query log analysis, the grouping. An excerpt of the query log data is shown with three queries, their timestamps and their query count, sorted ascending by timestamp. The queries are *stoferkamp+* (count 1), *Stoferkamp* (count 1) and *Baugenehmigung+Eimsb%C3%BCttel* (count 1).

5.2.1.6 Cleaning

Due to URL encoding, the remaining query logs had to go through a cleaning process. It is, for instance, not possible to use a whitespace directly in an URL. Instead, whitespaces are encoded with a plus sign (+) or with %20[2]. Without this process, the queries *stoferkamp+* and *Stoferkamp* would not be counted as the same queries when applying the clustering. Besides, all data was converted to lowercase, as shown in Table 5.2.

timestamp	query	normalized query	count
2014-10-12 06:34:00	stoferkamp+	stoferkamp	1
2014-10-12 06:35:00	Stoferkamp	stoferkamp	1
2014-10-12 06:35:00	Baugenehmigung+Eims%C3%BCttel	baugenehmigung eimsbüttel	1

Table 5.2: This table contains the result of the fourth step of the query log analysis, the cleaning. An excerpt of the query log data is shown with three queries. Each query contains the timestamp, the normalized query and the query count, sorted ascending by timestamp. The queries are *stoferkamp+*, normalized: *stoferkamp* (count 1), *Stoferkamp+*, normalized: *stoferkamp* (count 1) and *baugenehmigung+eimsb%C3%BCttel*, normalized: *baugenehmigung eimsbüttel* (count 1).

5.2.1.7 Clustering

In the last step, all normalized queries that are equal are clustered as one and, in the same step, the query counts are summed up, see Table 5.3. Overall, 74,174 queries are left (2.1 MB). The result obtained from this preliminary analysis of the log files is a file that contains all queries ever made on the TP plus the number of search requests for each query. The top three queries were *flüchtlinge unterkunft* (refugees accommodation), textitzuwendungen (contributions) and textitolympische spiele (olympic games). When looking at the queries, it becomes apparent that most of them are not very specific. In the next section it is described how search words with less than five results are extracted from this list.

2. https://tools.ietf.org/html/rfc3986 (last accessed 01-05-2019)

normalized query	count
stoferkamp	2
baugenehmigung eimsbüttel	1

Table 5.3: This table contains the result of the fifth step of the query log analysis, the clustering. An excerpt of the query log data is shown with three normalized queries and their query count, sorted ascending by timestamp. The log data that has been cleaned to remove url-encoding, transformed to lowercase and clustered to combine equal queries.The queries are *stoferkamp* (count 2) and *baugenehmigung eimsbüttel* (count 1).

5.2.2 Experiment Test Collection

By the end of the query log analysis, a list consisting of all queries and their corresponding query count is created. For an excerpt from this list, please see the Appendix. It contains 74,175 unique queries. Given that this thesis concentrates on queries that produce less than 5 results, a further processing step has to be applied. As a result, the derived query list is reduced to 23,239 search terms. However, it still holds several queries that are just numbers (2,384 in total), therefore another optimization step has to take place to remove them. The reason for removing the numbers is that there is no indication of what the user was originally looking for, and they are therefore irrelevant for the following experiments. In the end, the derived query list consists of 20,782 search terms, which is about 28% of the initial result list.

After narrowing the query list down to all relevant queries, the next step is to take this as the basis to create a smaller set for the experiments. Given that the results of the queries are to be manually evaluated depending on their fit regarding the queries, taking all 20,000+ queries into account would not be feasible. On the one hand, the intended list has to be reasonably large to get adequate results. On the other hand, it has to be possible to test this list by one person within a reasonable time frame. Manning et al. provide an overview of evaluation methods in IR stating that 50 information needs are a sufficient minimum [20], page 152. Starting from this, a set of approximately 100 queries is aimed to be extracted from the original list.

This test set has to also exclusively contain queries with at least roughly predictable search results. A query such as *projektgrundsätze* (project principles) does not convey the kind of documents that should be returned. That is one of the reasons why, despite being the simplest way, just taking the top 100 queries of the original list is not attainable. Regarding this experiment list, a few restrictions have to be made. In summation, three categories were used to classify the different kinds of queries. These categories are not exclusive, which means a query only gets sorted in its most meaningful category, even though more than one would fit.

Firstly, not all search terms indicate the intention of the user well enough and therefore have to be excluded. For example, a search for *hamburg grün* (hamburg green) could signify that the user expects results concerning the party *Die Grünen* (The Greens), as well as green areas and parks. All search terms that are too obscure to work with, as they do not belong to the search scope of the TP, for example, *kruzifixkiller* (crucifix killer) , are also sorted into this category. The second category is called "proper nouns / locations". Proper nouns and locations, for instance *roggenkamp*, are quite impossible to optimize (if not misspelled), and therefore are irrelevant for this thesis. The last category *intention clear* includes the remaining terms that are understandable in their search intention, e.g. *Grundstücksmarktbericht* (property market report).

number	category
21 (10%)	intention unclear
98 (44%)	proper nouns / locations
101 (46%)	intention clear

Table 5.4: Sorting the 220 most searched queries from the small set into three different groups: 21 (10%) intention unclear, 98 (44%) proper nouns/ locations and 101 (46%) intention clear

Before the categorization process can take place, the initial query list is sorted by query count in descending order. Then, the first 220 terms of the small result list are manually examined and sorted into the three categories, taking into account the search results and the comprehensibility of the search result itself. This smaller query list, like the original list, is sorted in descending order by query count. A different approach would be to select these terms randomly from the initial result list, however taking the queries that occur most often makes the improvements more meaningful. The results are presented in Table 5.4. All queries that contain spelling mistakes are deemed comprehensive and sorted in the category "intention clear". A frequent example of a spelling mistake is, for example, the noun *elbphilarmonie*. The Elbphilharmonie[3] is a very expensive landmark of Hamburg and therefore of great public interest - but prone to spelling errors, nonetheless. In summation, 7% of all queries contain spelling mistakes.

Consequently, the experiment list includes all terms with clear intention, in the sum 101 terms. Proper nouns and locations are not added to this set, because they are too specific. For example, the search for "intermisvertrag" cannot be improved, as it is a proper noun that needs to appear in the resulting documents.

5.3 Prototype and Web Application

To implement the modification of the queries according to the chosen techniques, a prototype was implemented in Java. In addition, a simple web application was created to evaluate the result.

The prototype is realized as a Java project that builds a connection to a Solr slave. Solr can be organized in a master-slave architecture, where a Solr slave is a copy of the original index. This requires a master server that contains the indexed documents and one or more slaves that copy this index. The slave created for this thesis was created in July 2018 and therefore contains a snapshot of the TP content at this time. The TP is continually extended as more and more documents are added to the index. The slave nonetheless represents an almost identical copy of the TP Index at the time of the experiment: July 2019.

Figure 5.1 offers an overview over the general architecture.

The prototype in the center of the figure makes search requests to Solr with a distinct query per request. It receives a list in return with all search results regarding this query, ranked by score. The prototype connects to a mysql database and offers a Representational State Transfer (REST) API for the web application.

3. https://www.elbphilharmonie.de/en/

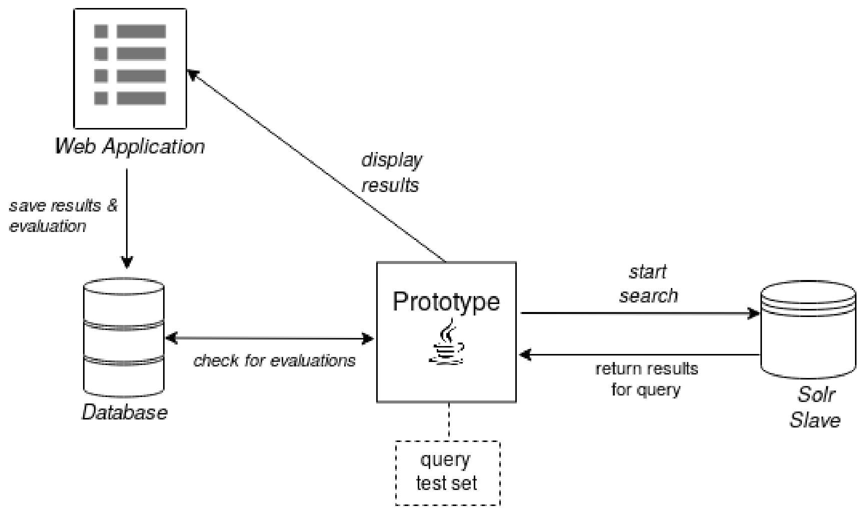
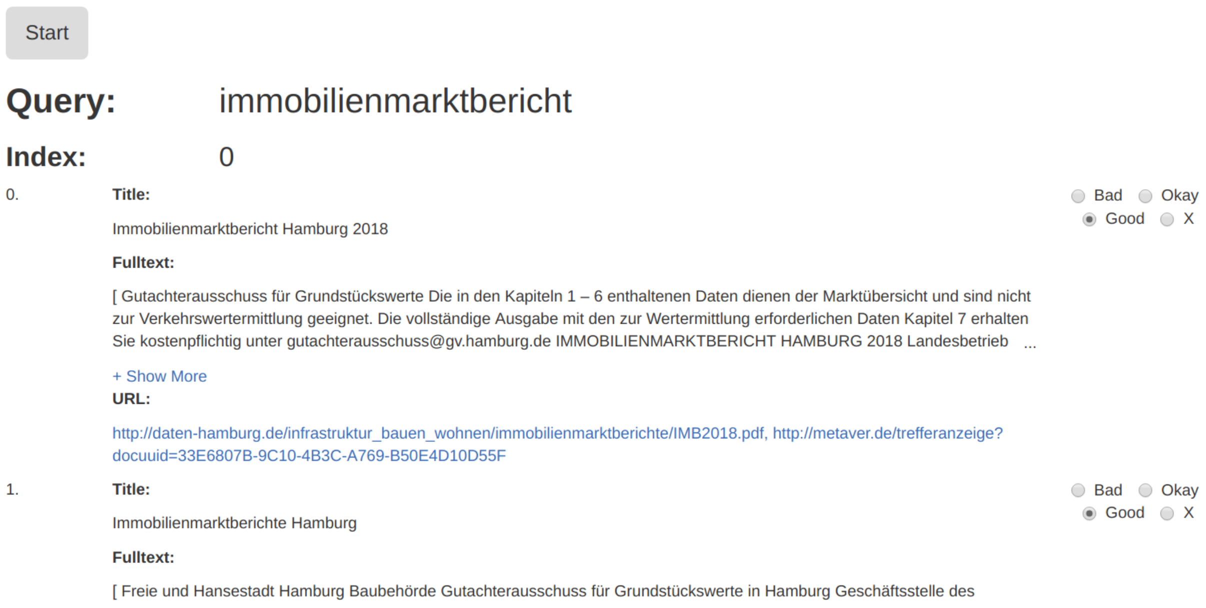


Figure 5.1: This graph depicts the way the experiments are carried out, from a technical point of view. The prototype holds the query test list and makes search requests to Solr. Solr returns a list of documents that are then shown on the web application, together with any already existing evaluations. The following evaluation of these documents are stored in a database, and the process is repeated with the next query from the test list.

Every experiment follows the same scheme. The prototype contains the test list with all queries, as described in section 5.3 In the first step, the web application starts the experiment. A simple "start" button triggers the handling of the query term on top of the search list. Afterward, each query token gets modified respectively. After modifying the query tokens, they are forwarded to Solr. This means that the modified query then has to go through the filter and tokenizer process. The display of the resulting documents is realized through a web application, a screenshot is shown in Figure 5.2. Every document can be evaluated as good, okay, bad or nonexistent. Before showing the results, the prototype checks with the database whether an evaluation already exists for the query - document combination. If yes, the evaluation gets pre-filled. In the last step, the query, the resulting documents, and the evaluations get stored in the database.


Start

Query: immobilienmarktbericht

Index: 0

0. **Title:** Immobilienmarktbericht Hamburg 2018 — Bad Okay [✓ Good] X

Fulltext:

[Gutachterausschuss für Grundstückswerte Die in den Kapiteln 1 – 6 enthaltenen Daten dienen der Marktübersicht und sind nicht zur Verkehrswertermittlung geeignet. Die vollständige Ausgabe mit den zur Wertermittlung erforderlichen Daten Kapitel 7 erhalten Sie kostenpflichtig unter gutachterausschuss@gv.hamburg.de IMMOBILIENMARKTBERICHT HAMBURG 2018 Landesbetrieb ...

+ Show More

URL:

http://daten-hamburg.de/infrastruktur_bauen_wohnen/immobilienmarktberichte/IMB2018.pdf, http://metaver.de/trefferanzeige?docuuid=33E6807B-9C10-4B3C-A769-B50E4D10D55F

1. **Title:** Immobilienmarktberichte Hamburg — Bad Okay [✓ Good] X

Fulltext:

[Freie und Hansestadt Hamburg Baubehörde Gutachterausschuss für Grundstückswerte in Hamburg Geschäftsstelle des

Figure 5.2: This figure shows a screenshot of the web application with the query *immobilienmarktbericht*, the position in the test list (0) and the first results. For every resulting document, the title, full text and URL are shown. The evaluation of each document is implemented with radio buttons (bad, okay, good, nonexistent.

6 Evaluation and Discussion

An experiment was conducted to determine whether the Levenshtein distance and decompounding can improve the search results of the Transparenzportal. Both techniques were applied separately to the same data set. During the experiments, it became apparent that specific characters could not be used in queries as they would always return zero results, due to the Solr version. After comparing with the Transparenzportal (TP) website itself, it became clear that the bug has been fixed there and the queries in question return good results. Therefore the two queries, that contain "&", were removed from the experiment list. As an additional complication, the query *fischerei biologisch Gutachten alster* (fishery biological report alster) produced so many queries, due to the Cartesian product, that it broke Solr. This query also was excluded from the test set.

This chapter is subdivided into three sections. First, the statistical analysis of the results with the aid of a t-test is presented. Second, the general evaluation of the data and the discovered findings is described. The third section provides an insight into a possible implementation in the TP.

6.1 Statistical analysis

To be able to compare the results statistically, a two-tailed paired t-test is applied. A t-test is a measure to compare the means of two dependent sets, for example, used in case-control studies. The control set in this case is the experiment on the test set without any modifications. The null hypothesis assumes that there is no difference between the mean of the control sample and the manipulated sample. The alternative hypothesis assumes that there is a difference between the true mean of the manipulated sample and the comparison value of the control sample. The significance level α is set to 0.05, which is most commonly used. The p-value is calculated as two-tailed, to not assume an effect in any direction.

Levenshtein: The mean nDCG for the control sample results in 0.2105, the mean for the Levenshtein sample is 0.2917. The degrees of freedom are set to 97. This results in a test statistic of -3.9375. The p-value is 0.0002 and the critical two-tail t-value is measured at 1.9847

The critical t-value is greater than the test statistic, and the p-value is less than α 0.05. Therefore the difference between the two sets is significant.

Decompounding: The mean nDCG for the control sample was measured to 0.2105, the mean for the decompounding sample is 0.4623. The degrees of freedom are set to 97. This results in the following values. This results in a test statistic of -16.1289. The p-value is 3.3138E-29 and the critical two-tail t-value is measured at 1.9847

The critical t-value is greater than the test statistic, and the p-value is less than α0.05. Therefore the difference between the two sets is significant. Both tests result in statistical significance. The null hypothesis is rejected. Therefore, it is assumed that both techniques improve search results of queries with originally five or fewer search results.

6.2 Data Evaluation

Both experiments reach statistical significance. However, some effects have been observed during the experiments that need to be included in the final evaluation.

First of all, the Levenshtein distance has proven to be great to compensate for ambiguities in the German language. For example, the nouns *Potential* and *Potenzial* denote the same thing (potential), both versions, with a "t" or "z", being formally correct. Even though none of the examined queries contain a street name, it is obvious that the Levenshtein distance would greatly improve findings for different streets with the endings "-straße" and "-strasse". This aspect was unfortunately not considered when creating the experiment list and all locations were removed, as explained in Chapter 5.2.2. Further experiments would certainly be able to confirm the benefit of the Levenshtein distance on street names. At the same time, decompounding would probably have a negative effect given that they are proper nouns. A compromise has to be found.

As an unexpected but unsurprising side effect, quite a large amount of LD alternative words were stemmed to the original query. An example of this is given by the query *Immobilienmarktbericht*. The following alternatives were found: *immobilienmarktbereicht, immobilienmarktberichts, immobilienmarktberichte, immobilenmarktberichte, immobilienmarktberichten, immobilienmarktberichtes*. However, only *immobilienmarktbereicht* (property market report, with a spelling mistake) was not stemmed and therefore is the same as *immobilienmarktbericht*. Nevertheless, this does not have an impact on the usability of the LD.

As expected, the LD works great on queries with spelling mistakes. Especially the two queries with a spelling mistake in *Elbphilharmonie* result in an nDCG of 1.0.

Compared to the LD, the decompounding algorithm results in a higher recall. Still, many of the resulting documents are not relevant for the query. This results from the fact that many queries contain the words *Bericht* (report), *Vertrag* (contract), *Verzeichnis* (register), *Gutachten* (reports) and other, which are featured in a lot of documents. As a result, many documents concerning completely different topics are returned. This difficulty arises from the overall nature of the user queries. Nearly half of the analyzed queries, see Subsection **??**, are proper nouns. Taking the proper noun *interimsvertrag* (Interim Agreement), for example, even though a search with decompounding returns a lot of documents for the query *vertrag* (contract), the user might get confused by the profound topic drift.

At the same time, decompounding works great for simple one-word queries such as *behindertenbeförderung* (transport for disabled persons). Through decompounding, the relevant document containing both *behinderten* (disabled persons) and *beförderung* (transport) is returned. This example shows how compounds are sometimes expected where only the constituents are used.

The decompounding approach of this thesis has also proven to be beneficial for queries that contain a specific year. A search for *immobilienmarktbericht 2017* (property market report 2017) and *2016 waffen- und munitionstransporte* (2016 arms and ammunition transports) is not able to find the intended document, because it does not exist. The original search fails at this point to return any other relevant documents. With decompounding, however, all other property market reports and arms and ammunition transports were found, due to the use of subsets.

The findings indicate that the Levenshtein distance and decompounding could improve the results for queries that otherwise only receive few results. When aiming to add these techniques to the Transparenzportal, a few limitations of the approaches in this thesis need to be considered.

First of all, as mentioned at the beginning of this chapter, the query *fischerei biologisch Gutachten alster* had to be excluded from the test set. The number of queries that were created by the Cartesian product exceeded the capacity of Solr. Hence, there either needs to be a cutoff at a specific amount of queries, or the LD should only be applied on maximum two-word-queries, or a combination of these approaches.

Second, given the fact that the TP contains some unusual nouns, it could be of interest to use the *compound noun training* tool from the Baseform Tool on a set of these nouns, to ensure that they are split correctly.

6.3 Application

The applied natural language processing has been proven useful in improving the recall of the Transparenzportal (TP). The experiments and the prototype were designed to easily test a small set of queries. This chapter discusses two possible implementations for the TP at query time on a larger scale, that can be added to the Transparenzportal: a filter and a Java solution. As the use of both decompounding and the Levenshtein distance together have not yet been evaluated, the following approaches are intended for either one or the other.

For the implementation, a number of criteria need to be considered. First, the solution should be low maintenance and easily added to the TP. It needs to be expandable and update itself when new documents are added. Second, a high degree of modularity is desired. The added elements might not prove to be as suitable for the live system as anticipated and should be easily switched off in the case of any problem. Third, the impact on load times and page speed needs to be taken into account.

The first approach considers the development of a custom Solr filter. This filter is responsible for applying the desired language processing technique. As stated in Chapter 3, the query has to undergo several steps bevor the actual search is carried out. Consequently, this filter should not be applied after the Snowball Porter Filter (stemming), but also not before the tokenization. As a compromise, the new filter could be situated between the Whitespace tokenizer and the Synonym Filter. This also prevents the modification of the synonyms, which is not desired.

As an alternative, a small service (for example written in Java) could be inserted between CKAN and Solr. Instead of directly forwarding the user queries to Solr, CKAN could forward the query to the service, which in turn produces alternative queries and hands them back. In the case of the Levenshtein distance, another function could be added to this service, which automatically updates the DAWG when new documents are added to the TP. This could be implemented, for example, with the aid of a part-of-speech tagger to extract the nouns from the full texts.

Both approaches could easily be added to the TP. However, the custom Solr filter might be more difficult to be kept up to date, as a new DAWG would need to be added every time new documents enter the TP. Regarding modularity, they do not differ. The page speed would in both cases be dictated by the loading time of the data structures. A query filter would have to load these every time a query is processed. In contrast, the service would only need to be started once and therefore only load the necessary data structures once. In conclusion, the addition of a service is the better solution given these two options.

7 Summary and Conclusion

This thesis has investigated the gain of adding the Levenshtein distance and decompounding to the TP. An experiment has confirmed with statistical significance that the language processing techniques improve the recall of resulting documents for queries that did not perform well before. It was discussed that a few adjustments would have to be made to the applied techniques before adding them to the TP. The corresponding application possibilities have been outlined and discussed.

Given the fact that the TP is the only source on the internet for most of these unique documents, the more results returned, the better. For this reason, the LD, as well as the decompounding technique, would be beneficial. They both tackle different problems: one works well with spelling mistakes, the other works well with compounds.

A difficulty arises from the fact that a parameter is set on the TP so that every single query token must occur in a resulting document. The LD cannot help with that, and the decompounding approach implemented in this thesis only fixes this as a by-product by using subsets. Changing the "minimum must match" setting to 90% could, therefore, be useful. In addition, the TP does not remove stopwords from a query. For example, the word *innerhalb* is unnecessary for most queries, because it does not convey an information need. The query *Bauvorhaben innerhalb Wandsbek* (construction project within Wandsbek) does not find any documents at the moment that contain the word*Bauvorhaben* and *Wandsbek*, if they do not also contain the word *innerhalb*. This could be avoided by adding a query processing step that removes all redundant words.

Furthermore, a small inconsistency in the Solr Filters became obvious during the experiments. The Word Delimiter Filter (WDT) does not concatenate query tokens at query time. This is a disadvantage that should be fixed. Given the query *meister-titel* (championship title), a search for *meister* (championship) and *titel* (title) would be started. However, the document that contains the correct spelling *meistertitel* would not be found. The settings for the WDT Filter should be the same for the query and for the index.

In conclusion, the presented findings suggest an improvement concerning the set of low performing queries. To calculate the impact these applied techniques could have on the whole TP, a second experiment should be carried out with a randomized test set that is created from all TP queries. This would ensure to discover possible unexpected side-effects that might not have been considered yet.

8 Outlook

The techniques applied in this thesis are only a small set of possible approaches. Taking the findings in Chapter 6 as a starting point, a few other improvement possibilities are presented in this chapter.

To be able to check whether the TP delivers the correct results, it could also be interesting to include a feedback option on the search result page. A form could be added that always includes the query itself and a comment section, to add why this particular document should have been found. This might be a less frustrating way for the user to convey an information need than to file an official request. This particular hurdle might be too high for most users to overcome, and they might just abandon their search. By adding a feedback form not only could it show which search queries lack results and are thought after. It could also signal the town of Hamburg, which areas are not yet covered enough by official documents.

Another field of study could be the nouns that are connected with *und* (and). In German, nouns that consist of nouns with the same last part can be combined. For example, *Wasserverbände und Bodenverbände* (Water Associations and Soil Associations) can be merged into *Wasser- und Bodenverbände* (Water and Soil Associations). Regardless of whether these are merged into a fixed term or not, a user might just search for one of them. *Bodenverbände* (Soil Associations) would return the correct results, however, if the user were to only search for *Wasserverbände* (Water Associations), they might get no results, because the word does not exist in the index. Therefore, these word structures could be subject to further improvement.

Another research topic could be the enrichment of the synonym file, which is not up to date at the moment. The synonyms could be found by examining which words often appear together in documents. Given the fact that the Levenshtein distance should not be applied to short strings, see Chapter 3, short queries could benefit from synonyms in particular.

Appendix

Query Log Analysis Results

nr	normalized query	query count
1	flüchtlinge unterkunft	85866
2	zuwendungen	64448
3	olympische spiele	61839
4	olympia	58771
5	olymp	55617
6	elbphilharmonie	45606
7	schuldnerberatung	32103
8	busbeschleunigung	27061
9	geodaten	25319
10	flüchtlinge	17970
11	jugendhilfe	13885
12	bundesrat	11523
13	munition	11152
14	justizbehörde	8445
15	neue liberale	7351
16	radioaktivität	6724
17	bunker	6475
18	sandbek-west	5927
19	ziegelteich	3895
20	universitätsbibliothek	3454
21	alkis	2545
22	staats- und universitätsbibliothek hamburg	2325
23	vergebene vob aufträge	1841
24	genehmigung_nach_hbauo	1734
25	gtfs	1630
26	biotopkataster	1514
27	dieter lenzen	1473
28	e-commerce	1466
29	baugenehmigung	1362
30	messergebnisse zur radioaktivität	1344
31	reinigungsplan	1238
32	haw hamburg	1161
33	g20	783

Table 8.1: Most searched queries 2014-2018 (after going through the cleaning, grouping and clustering process)

nr	normalized query	query count
1	verkehsunfälle	297
2	immobilienmarktbericht	237
3	orthophoto	225
4	düsterntwiete	117
5	disk	116
6	landesgrundbesitz	95
7	hempenkamp	92
8	landesgrundbesitzverzeichnis	90
9	ortho	79
10	alkis ausgewählte daten	73
11	bruchkanten	71
12	shapefile	68
13	2016 waffen- und munitionstransporte	65
14	bohrarchiv	59
15	waldjugend	59
16	straße op de elg	58
17	dimag	57
18	hauskoordinaten	52
19	dtk	52
20	baugenehmigung ohlwören	51
21	*	46
22	alkis verwaltungsgrenzen	45
23	orthofotos	44
24	projektgrundsätze	42
25	sponsoringbericht	42
26	citygml	38
27	virtuelle passpunkte	38
28	hausumringe	37
29	elbphilarmonie	37
30	lidar	35

Table 8.2: Most searched terms 2014-2018 with less than 5 results 2014-2018 (after going through the cleaning, grouping and clustering process).

Bibliography

[1] M-Dyaa Albakour, Udo Kruschwitz, Nikolaos Nanas, Yunhyong Kim, Dawei Song, Maria Fasli, and Anne De Roeck. *Autoeval: An evaluation methodology for evaluating query suggestions using query logs*. In: *European Conference on Information Retrieval*. Springer. Dublin, Ireland, 2011, pp. 605–610.

[2] Chris Biemann, Uwe Quasthoff, Gerhard Heyer, and Florian Holz. *ASV Toolbox: a Modular Collection of Language Exploration Tools*. In: *LREC*. Marrakech, Morocco, 2008, pp. 1760–1767.

[3] Martin Braschler and Bärbel Ripplinger. *How effective is stemming and decompounding for German text retrieval?* In: *Information Retrieval* 7.3-4 (2004), pp. 291–316.

[4] Tilo Buschmann and Leonid V Bystrykh. *Levenshtein error-correcting barcodes for multiplexed DNA sequencing*. In: *BMC bioinformatics* 14.1 (2013), p. 272.

[5] Kai-Uwe Carstensen, Christian Ebert, Cornelia Ebert, Susanne Jekat, Hagen Langer, and Ralf Klabunde. *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Springer-Verlag, 2009.

[6] Anne Chardonnens and Simon Hengchen. *Text Mining for User Query Analysis: A 5-Step Method for Cultural Heritage Institutions*. In: *Proceedings of the 15th International Symposium on Information Science (ISI 2017); Berlin, Germany, 13th—15th March 2017: Everything Changes, Everything Stays the Same? Understanding Information Spaces*. M. Gäde/V. Trkulja/V. Petras (Eds.) Berlin, Germany, 2017, pp. 177–189.

[7] Christian Charras and Thierry Lecroq. *Handbook of exact string matching algorithms*. Citeseer, 2004.

[8] Aitao Chen and Fredric C Gey. *Multilingual information retrieval using machine translation, relevance feedback and decompounding*. In: *Information Retrieval* 7.1-2 (2004), pp. 149–182.

[9] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*. Vol. 520. Addison-Wesley Reading, 2010.

[10] Antonio Ferrández, Jesús Peral, Higinio Mora, and David Gil. *Architecture for Efficient String Dictionaries in E-Learning*. In: *Multidisciplinary Digital Publishing Institute Proceedings*. Vol. 2. 19. article number 1251. Punta Cana, Dominican Republic, 2018.

[11] Trey Grainger and Timothy Potter. *Solr in action*. Manning Publications Co., 2014.

[12] Wilbert Jan Heeringa. *Measuring dialect pronunciation differences using Levenshtein distance*. PhD thesis. Citeseer, 2004.

[13] Kalervo Järvelin and Jaana Kekäläinen. *Cumulated gain-based evaluation of IR techniques*. In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[14] Thorsten Joachims. *Optimizing search engines using clickthrough data*. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. Edmonton, Canada, 2002, pp. 133–142.

[15] Jaap Kamps, Christof Monz, and Maarten De Rijke. *Combining evidence for cross-language information retrieval*. In: *Workshop of the Cross-Language Evaluation Forum for European Languages*. Springer. Rome, Italy, 2002, pp. 111–126.

[16] Andrea Krott, Gary Libben, Gonia Jarema, Wolfgang Dressler, Robert Schreuder, and Harald Baayen. *Probability in the grammar of German and Dutch: Interfixation in triconstituent compounds*. In: *Language and speech* 47.1 (2004), pp. 83–106.

[17] Vladimir I Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals*. In: *In: Doklady Akademii Nauk SSSR Soviet physics doklady 1965, p. 845-846 (Russian), English translation in: Soviet Physics Doklady, Vol. 10 location Soviet Physics Doklady*. Vol. 10. 8. 1966, pp. 707–710.

[18] Bertrand Lisbach and Victoria Meyer. *Linguistic identity matching*. Springer, 2013.

[19] Faulhaber Manja and Hilke Elsen. *Neologismen in der Kosmetikwerbung*. In: *Muttersprache: Vierteljahresschrift für deutsche Sprache* (2016), pp. 193–207.

[20] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Online edition. Cambridge University Press, 2009. ISBN: 978-1-78216-136-3.

[21] Mandar Mitra, Amit Singhal, and Chris Buckley. *Improving automatic query expansion*. In: *SIGIR*. Vol. 98. Melbourne, Australia, 1998, pp. 206–214.

[22] Donald R Morrison. *PATRICIA—practical algorithm to retrieve information coded in alphanumeric*. In: *Journal of the ACM (JACM)* 15.4 (1968), pp. 514–534.

[23] Eva Pettersson, Beáta Megyesi, and Joakim Nivre. *Normalisation of historical text using context-sensitive weighted Levenshtein distance and compound splitting*. In: *Proceedings of the 19th Nordic conference of computational linguistics (Nodalida 2013)*. Oslo,Norway, 2013, pp. 163–179.

[24] Shantanu Rane and Wei Sun. *Privacy preserving string comparisons based on Levenshtein distance*. In: *2010 IEEE International Workshop on Information Forensics and Security*. IEEE. Seattle, WA, USA, 2010, pp. 1–6.

[25] Anne Schiller. *German compound analysis with wfsc*. In: *International Workshop on Finite-State Methods and Natural Language Processing*. Springer. Helsinki, Finland, 2005, pp. 239–246.

[26] Barbara Schlücker. *Die deutsche Kompositionsfreudigkeit. Übersicht und Einführung*. In: *Das Deutsche als kompositionsfreudige Sprache. Strukturelle Eigenschaften und systembezogene Aspekte*. Vol. 46. Linguistik - Impulse & Tendenzen. Berlin, Boston: de Gruyter, 2012, pp. 1–25.

[27] Dikshant Shahi. *Apache Solr, A Practical Approach to Enterprise Search*. 1st. Springer Science+Business Media New York, 2015.

[28] Bashar Al-Shboul and Sung-Hyon Myaeng. *Analyzing topic drift in query expansion for Information Retrieval from a large-scale patent DataBase*. In: *2014 International Conference on Big Data and Smart Computing (BIGCOMP)*. IEEE. Bangkok, Thailand, 2014, pp. 177–182.

[29] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. *Analysis of a very large web search engine query log*. In: *ACm SIGIR Forum*. Vol. 33. 1. ACM. Salt Palace Convention Center, Salt Lake City, UT, 1999, pp. 6–12.

[30] David Smiley, Kranti Parisa, Eric Pugh, and Matt Mitchell. *Apache Solr Enterprise Search Server*. 3rd. Packt Publishing Ltd., 2015.

[31] Zhan Su, Byung-Ryul Ahn, Ki-Yol Eom, Min-Koo Kang, Jin-Pyung Kim, and Moon-Kyun Kim. *Plagiarism detection using the Levenshtein distance and Smith-Waterman algorithm*. In: *2008 3rd International Conference on Innovative Computing Information and Control*. IEEE. Dalian, China, 2008, pp. 569–569.

[32] Kyoko Sugisaki and Don Tuggener. *German compound splitting using the compound productivity of morphemes*. In: *14th Conference on Natural Language Processing-KONVENS 2018*. Austrian Academy of Sciences Press. Vienna, Austria, 2018, pp. 141–147.

[33] Ahmet Uyar. *Google stemming mechanisms*. In: *Journal of information science* 35.5 (2009), pp. 499–514.

[34] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. *A theoretical analysis of NDCG ranking measures*. In: *Proceedings of the 26th annual conference on learning theory (COLT 2013)*. Vol. 8. Princeton, NJ, USA, 2013, p. 6.

[35] Hans Friedrich Witschel and Chris Biemann. *Rigorous dimensionality reduction through linguistically motivated feature selection for text categorization*. In: *Proceedings of the 15th Nordic Conference of Computational Linguistics (NODALIDA 2005)*. Joensuu, Finland, 2006, pp. 210–217.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 14.08.2019

Katrin Caragiuli