UNI KASSEL VERSITÄT Department of Electrical Engineering and Computer Science
Chair Intelligent Embedded Systems

Identification of key information with topic analysis on large unstructured text data

B A C H E L O R T H E S I S

Department of Electrical Engineering and Computer Science
University of Kassel

Author Name:	Klara Maximiliane Gutekunst
Address:	*** REMOVED ***
	34125 Kassel
Matriculation number:	*** REMOVED ***
E-Mail:	klara.gutekunst@student.uni-kassel.de
Department:	Chair Intelligent Embedded Systems
Examining board 1:	Prof. Dr. rer. nat. Bernhard Sick
Examining board 2:	Prof. Dr. Gerd Stumme
Supervisor:	Dr. Christian Gruhl
Date:	September 17, 2023

Abstract

Finding relevant documents and connections between multiple ones becomes significantly more difficult due to the sheer amount of documents available. Institutes, such as the (German) tax offices have access to leak data, e.g., the Bahama leak, containing huge amounts of documents and valuable information yet to be extracted. However, these institutes, companies and individuals do not have sufficient resources to explore individual documents in order to find a specific one or to identify the key topics of them. Hence, computational means, such as text mining, may facilitate the situation. This thesis proposes an approach to find relevant documents and identify topics from a large text corpus.

Contents

Abkürzungsverzeichnis

RQ	Research Question
LDA	Latent Dirichlet Allocation
TF-IDF	Term Frequency - Inverse Document Frequency
BERTopic	BERT Topic Model
Doc2Vec	Document to Vector
GloVe	Global Vectors
USE	Universal Sentence Encoder
PCA	Principal Component Analysis
kNN	k-nearest neighbor
API	Application Programming Interface
JSON	JavaScript Object Notation
PKL	Pickle
HNSW	Hierch""arical Navigable Small World
PDF	todo

Das ist die Einleitung. “Dies ist ein Zitat” [5]. Das ist eine Fußnote[1].

Abbildung 1 zeigt das Logo der Uni Kassel.

Figure 1: Das Logo der Uni Kassel

Listing 0.1 implementiert eine Klasse in `java`.

```java
class Foo {
    String bar;
}
```

Listing 0.1: Eine einfache Klasse

Tabelle 1 enthält die Daten für die Auswertung.

Table 1: Einfache Daten

Nr.	Punkte	Aufgaben	Bewertet
1	30	40	26
2	44	75	43
3	22	23	14
4	47	46	32
5	45	63	42
6	58	71	54
7	54	80	54
8	51	60	44
9	35	48	35
10	25	38	25
11	37	48	37
Gesamt	448	592	406

[1]Ich putz hier nur.

1 Introduction

According to [13], the Bahamas leak is roughly 38 GB collection of documents, which were leaked from in 2016. The data is used by (German) tax offices to identify tax evasion. However, it has proven to be challenging to identify the relevant documents and connections between documents due to the amount of documents in the leak.

Therefore, the goal of this thesis is to suggest approaches to support the investigators of the tax offices. Text exploration methods include topic modelling.

The topics to be identified can be groups of words which appear more often than the average or groups of similar documents. Hence, a topic is not always the defined topic in terms of content, but sometimes a statistical phenomenon. Since different methods define different topics, as they work and define the meaning of 'topic' differently, their results are compared and evaluated on the dataset.

Besides literature research, application and evaluation of the methods identified, certain preprocessing methods have proven to be eminent to successful work with unstructured text data. These methods include chunking/ tokenization (separating texts into equally sized segments), lemmatization (e.g., faster to fast), conversion to small letters and stop-word-lists.

1.1 Motivation/ Objective

Assumption: similarities between documents (in terms of appearance and content wise) On a broader scope this thesis aims to provide computational means to facilitate the work with large unstructured text data for individuals. In the following, certain goals are defined, which are to be achieved in this thesis.

Motivation/ problem: actively use machine learning techniques to analyse large text corpus and thus, reduce the amount of manual (human) work. This includes analysis in terms of textual (content) and visual (appearance/ layout) information, like a human would do. The goal is to identify similarities between documents and group (cluster) them together - topic of the cluster do not have to be labeled specificially. This serves as a first step/ preprocessing, e.g., a human finds a document of interest (for instance from random sampling) and wants to find similar documents to it.

Usability. The methods should be bundeled in an application, which is easy to use and does not require any programming skills.

Semantic similarity. The documents grouped together should be semantically similar.

Topic identification. The topics identified should be meaningful to the task at hand.

Offline Calculation. The database should be calculated offline, so that the queries can be executed with little latency.

1.2 Related work

1.3 Research Questions

The following research questions build the guideline for this thesis.

1.3.1 Research Question (RQ)1: Effect of different preprocessing pipelines on performance?

In terms of RQ1, one could compare different types of stemmers (i.e. algorithmic vs. dictionary-based).

1.3.2 RQ2: Effect of different similarity measurement types on performance?

In terms of RQ2, one could compare different types of similarity measurement types (i.e. cosine similarity vs. soft cosine similarity).

1.3.3 RQ3: Which type of database is best suited for this task?

In terms of RQ3, one could compare different types of databases (i.e. object-orientated, relational, document).

1.3.4 RQ4: Effect of different embeddings on performance?

In terms of RQ4, one could compare different types of embeddings (i.e. Doc2Vec, Bag-of-words, LDA, BERTopic).

1.4 Structure of the Thesis

The rest of this thesis is structured as follows. Chapter 2 provides background information on the topic of this thesis. Chapter 3 describes the implementation of the methods. Chapter 4 evaluates the methods. Chapter 5 discusses the results. Chapter 6 concludes this thesis and Chapter 7 gives an outlook on future work.

2 Methodology

[12]

Basic concepts, methods used, etc.

2.1 Preprocessing

2.1.1 Tokenization/ Chunking

2.1.2 Lemmatization

Type of Stemmers. Porter, Snowball, Lancaster, etc. Pre-trained/defined dense vector dictionaries (Word2Vec, Global Vectors (GloVe), FastText, etc.)

2.1.3 Stop-Word-Removal

2.1.4 Lower case

2.2 Similarity Measurement

[7]

2.2.1 Cosine Similarity

2.2.2 Soft Cosine Similarity

2.2.3 euclidian distance

2.3 Embeddings

[10] [9]

Skizze von Pipeline für jedes Embedding, welche zeigt, wie die Daten vorverarbeitet (stemming etc.) werden/ was das Model selber macht.

2.3.1 Doc2Vec

[9] two flavor of doc2vec: PV-DM and PV-DBOW (https://thinkinfi.com/simple-doc2vec-explained/) [11]

2.3.2 TF-IDF

Test test test

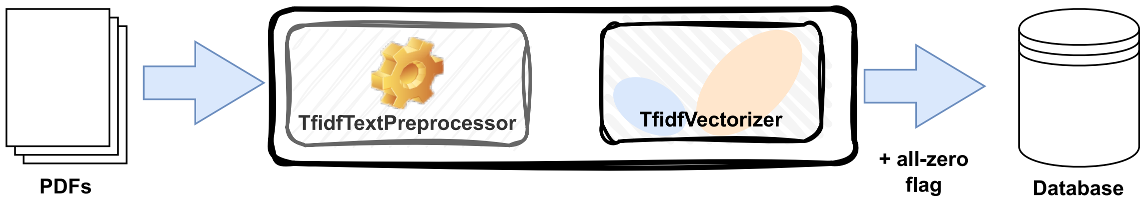

Figure 2.1: TFIDF Preprocessing

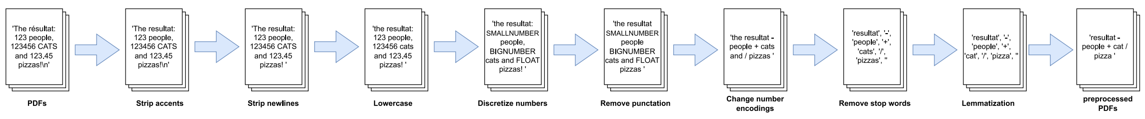
'The résultat: 123 people, 123456 CATS and 123.45 pizzas!\n'
PDFs
'The resultat: 123 people, 123456 CATS and 123.45 pizzas!\n'
Strip accents
'The resultat: 123 people, 123456 CATS and 123.45 pizzas! '
Strip newlines
'the resultat: 123 people, 123456 cats and 123.45 pizzas! '
Lowercase
'the resultat: SMALLNUMBER people, BIGNUMBER cats and FLOAT pizzas! '
Discretize numbers
'the resultat SMALLNUMBER people BIGNUMBER cats and FLOAT pizzas '
Remove punctation
'the resultat - people + cats and / pizzas '
Change number encodings
'resultat', '-', 'people', '+', 'cats', '/', 'pizzas', ''
Remove stop words
'resultat', '-', 'people', '+', 'cat', '/', 'pizza', ''
Lemmatization
'resultat - people + cat / pizza '
preprocessed PDFs

Figure 2.2: TFIDF Preprocessing

2.3.3 Universal sentence encoder

Universal Sentence Encoder (USE) [2]

2.3.4 InferSent

[3]

2.3.5 Hugging face's sentence Transformers

[15]

2.4 Topic Modelling

2.4.1 BERTopic

2.4.2 LDA

2.4.3 Word Clouds

frequency of words in a document

2.5 Appearance of documents

documents saved as images in .png format, bad quality to minimize size of database when querying db, top image results looked similar, which is how idea of this section arose

2.5.1 Compression of data

AE

eigenface

like pca, but for images

2.5.2 Clustering

Clustering is used in a variety of domains to group data into meaningful subsclasses, i.e. clusters [14, 4, 6]. According to Patwary et al., common domains include anomaly/ outlier detection, noise filtering, document clustering and image segmentation. The goal is to find clusters, which have a low inter-class similarity and a high intra-class similarity [14].

There are multiple clustering techniques, which can be divided into four categories [1]:

1. **Hierarchical clustering**: Algorithms, which create spherical or conex shaped clusters, possibly naturally occurring. A terminal condition has to be defined beforehand. Examples include CLINK and SLINK [4].

2. **Partitional based clustering**: Algorithms, which partition the data into k clusters, whereas k is given apriori. Clusters are shaped in a spherical manner, are similar in size and not necessarily naturally occurring. KMeans is a popular example of a partitional based clustering algorithm.

3. **Density based clustering**: Resulting clusters can be of arbitrary shape and size. The number of clusters does not have to be defined apriori. However, the algorithms are sensitive to input parameters, such as radius, minimum number of points and threshold. Popular examples are DBSCAN and OPTICS.

4. **Grid based clustering**: Similar to density based clustering, but according to Agrawal et al. better than density based clustering. Examples include flexible grid-based clustering [4].

KMeans

DBSCAN

HDBSCAN*

OPTICS

Variational Bayesian estimation of a Gaussian mixture

Annoy

3 Implementation

3.1 Slurm

Slurm is an open-source management tool for Linux clusters [25]. It allocates recources, i.e. compute nodes, and provides the means to start, execute and monitor jobs [25, 27].

The so-called slurm deamons control nodes, partitions, jobs and job steps [25]. According to TODO, a partition is a group of nodes and a job is the allocation of ressources, i.e. compute nodes, to an user for a limited time period. A basic visualization of the architecture is given in Figure 3.1.

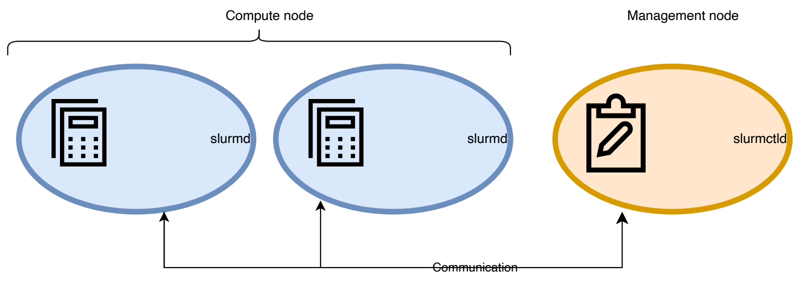

Figure 3.1: Slurm architecture. The management node has a `slurmctld` deamon, while every compute node has a `slurmd` deamon. The nodes communicate. The user can use certain commmands, for instance `srun` and `squeue`, anywhere on the cluster.

3.2 Database Elasticsearch

Elasticsearch is a widely used non-relational database, which was designed to store and perform full-text search on large corpus of unstructured data [26]. This open-source distributed document-driven database system is build in Java and is based on the Apache Lucene (Java) library for high-speed full-text search [26, 28]. According to [28], Elasticsearch provides Wikipedia's full-text search and suggestions as well as Github's code search and Stack Overflow's geolocation queries and related questions. Needless to say, Elasticsearch is prone to handle Big Data and enables near real-time search by index refreshing periods of one second.

Table 3.1: Fields in Elasticsearch database in index *Bahamas*

field name	**field description**
_id	unique identifier of document `i`
doc2vec	doc2vec embedding of `i`
sim_docs_tfidf	sim_docs_tfidf embedding + all-zero flag of `i`
google_univ_sent_encoding	google_univ_sent_encoding embedding of `i`
huggingface_sent_transformer	huggingface_sent_transformer embedding of `i`
inferSent_AE	inferSent_AE embedding of `i`
pca_image	two dimensional Principal Component Analysis (PCA) version of first page image of `i`
pca_kmeans_cluster	Cluster of `i` identified by KMeans on PCA version of image
text	text of `i`
path	path on local maschine to `i`
image	image of first page of `i`

Elasticsearch's entries, i.e. documents are stored in logical units, so called indices. As stated in Zamfir et al. and Voit et al.'s work, the indices are structured similar to Apache Lucene's inverted index format. An index can be spread into multiple nodes. A node is single running instance of Elasticsearch [28]. An index is divided into one or more shards, which can be stored on different servers and enable parallelization [28].

Elasticsearch indices' entries are documents, which are saved in a JavaScript Object Notation (JSON) fromat [26]. A document's fields and field types are defined by the user when initializing the database index. By default, every field of a document is indexed and searchable [28].

Replicas are copies of shards, which create redundancy and thus, ensure availability [28].

The database is filled once with data from a large unstructured corpus of todos (PDFs). After the initialization of the database, it is used for queries. Therefore, the procedure is carries out in a completely offline fashion.

The index *Bahamas* stores different embeddings of the text layer information and metadata of the documents. As depicted in Figure 3.2, not only textual information is stored in the database, but also the images of the first page of the PDFs. The structure of the index is presented in Table 3.1.

By specifying the unique `_id` of a document and the database `index`, it is possible to retrieve a specific document from the database using the `GET Application Programming Interface (API)`. The query is real-time by default. The parameter `_source_excludes` or `_source_includes` may be used to exclude or include specific fields of the document in the response [17].

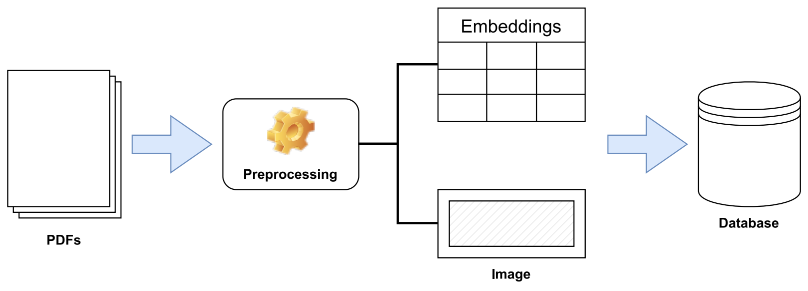

Figure 3.2: PDFs to Database. The first page of the PDFs are converted to images and the complete text is extracted. The images are stored in the database as well as the text and different embeddings of the text.

The keyword used when performing full-text search in this case is `match`. To query for a specific value, one has to specify the `<field>` of interest and the query value.

Elasticsearch preprocesses the query value before starting the search [24]. The default preprocessing steps of the so-called default analyser include tokenization and lowercasing [22]. Omitting stop words is disabled by default, but it is possible to provide custom stop words or use the English stop word list [22]. It is possible to create custom tokenizers, which split the query value into tokens of a certain maximum length. In this work, the default analyser is used for the full-text search, since for instance configuring a maximum token length did not seem necessary or likely to improve the results.

Another useful feature of Elasticsearch is the multi-terms synonym expansion. When the user queries a specific phrase Elasticsearch expands the query to include synonyms of the query terms [23]. The maximum number of expansion terms is set to 50 by default, but can be configured by the user [21]. By default, the multi-terms synonym expansion option is enabled [21].

Elasticsearch also provides the option to perform fuzzy matching instead of exact search. By enabling the fuzzy matching option, a Elasticsearch query consisting of for instance, *Bahama* returns documents which have the word *Bahamas*. By default this option is not enabled, but can be enabled and configured individually by the user [21]. In this work, the fuzzy matching option is set to `AUTO`, which means in terms of keyword or text fields that the allowed Levenshtein Edit Distance, i.e. number of characters changed to create an exact match between two terms, to be considered a match, is correlated to the length of the term [16]. By default, terms of length up to two characters must match exactly, terms of length three to five characters must have an edit distance of one and terms of length six or more characters must have an edit distance of two [16].

Another search option of Elasticsearch is the k-nearest neighbor (kNN) search. The return value of a kNN search are the `k` nearest neighbors in terms of a certain distance function of a query vector [8]. According to Malkov and Yashunin, one of kNN search's use cases is semantic document retrieval, which makes it a good fit for this task. The query is a dense vector of the same dimension as the (dense) vectors stored in the database. According to [19], the kNN either returns the exact brute-force nearest neighbors or approximate nearest neighbors calculated by the Hiercharical Navigable Small World (HNSW) algorithm [8, 19]. In this work, the approximate nearest neighbors search is used, since it is faster and the results are good enough for the use case of this work. HNSW is a graph-based algorithm [8]. The term `navigable` refers to the graphs used, which are graphs with (poly-)logarithmic scaling of links traversed during greedy traversal with respect to the network size [8]. The idea of a `hiercharical` algorithm is to create a multilayer graph, grouping links according to their link length, as displayed in Figure 3.3. The search starts on the uppermost layer, i.e. the layer containing the longest links, greedily traversing the layer until reaching the local minimum. It uses this local minimum as the starting point at the next lower layer and the process is repeated until the lowest layer is reached [8]. The layers of the graph are built incrementally, and a neighbour selection heuristic, as depicted in Figure 3.4, not only creates links between close elements, but also between isolated clusters to ensure global connectivity [8].

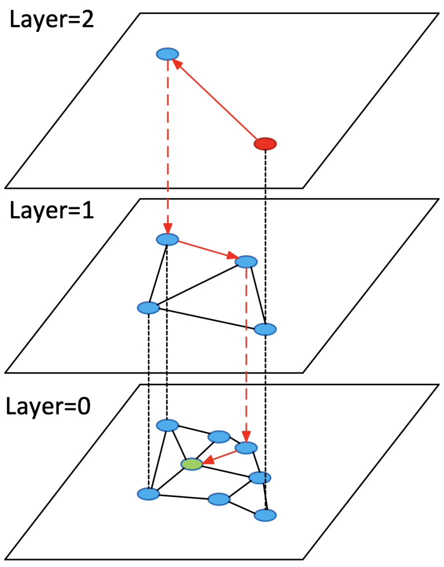

Figure 3.3: Structure of HNSW layer from [8]. The search starts on the uppermost layer, i.e. the layer containing the longest links, greedily traversing the layer until reaching the local minimum. The local minimum is used as the starting point at the next lower layer and the process is repeated until the lowest layer is reached.

In order to perform the kNN search on a `<field>` it has to be of type `dense_vector`, indexed and a `similarity` measure has to be defined when initializing the database [19]. The similarity measure used in this work is the cosine similarity, which calculates the `_score` of a document according to Equation 3.2, where `query` is the query vector and `vector` is the vector representation of the document in the database [18]. Since cosine is not defined on vectors with zero magnitude, embeddings which can possibly return all zero vector representations, such as sim_docs_tfidf, are enhanced with an all-zero flag in this work.

$$\frac{1 + \text{cosine}(\text{query}, \text{vector})}{2} \tag{3.1}$$

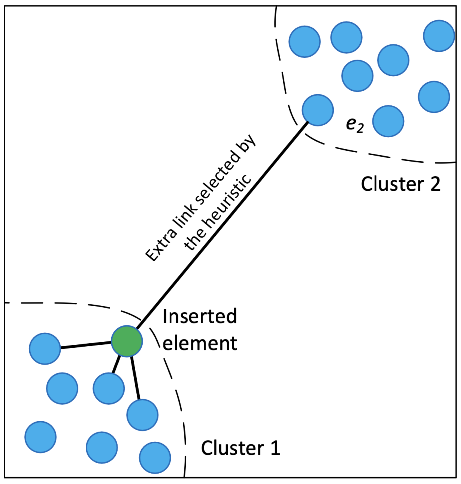

Figure 3.4: Neighbour selection heuristic of HNSW from [8]. The heuristic creates diverse links, i.e. links between close elements (e.g., green circle and elements in cluster 1) and between isolated clusters (e.g., green circle and e_2) to ensure global connectivity.

Elasticsearch's kNN implementation not only allows literal matching on search terms, but also semantic search [19]. Besides Elasticsearch, the elastic stack offers other tools, for instance Kibana, which provides a user interface to manage different models. After saving a model in Kibana, it is possible to create a text embedding ingest pipeline, which embeds new documents or reindexes existing documents [20]. However, in this work, Kibana is not used and the used models are saved on disk as Pickle (PKL) files. Therefore, instead of using the kNN query structure for semantic search on embeddings provided by Elasticsearch, the normal kNN search on a field which contains an embedding is used.

3.3 User Interface

3.3.1 Backend

Flask

3.3.2 Frontend

angular

4 Evaluation

4.1 analysis/ comparison of models

difference query responses for different models? any images whoch produce unusuable results?

4.2 Evaluation of the performance

4.2.1 Fahnder clustern

4.2.2 Fahnder bewerten Resultate (image matrix)

4.3 Evaluation of the usability

4.3.1 Metrics

5 Results

Evaluate the results from the previous chapter.

5.1 Fulfilment of objective

5.2 Research results

5.2.1 RQ1: Question 1?

6 Conclusion

7 Outlook

7.1 Future Work

Bibliography

[1] K.P. Agrawal, Sanjay Garg, Shashikant Sharma, and Pinkal Patel. Development and validation of optics based spatio-temporal clustering technique. *Information Sciences*, 369:388–401, 2016. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2016.06.048. URL `https://www.sciencedirect.com/science/article/pii/S0020025516304765`.

[2] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[3] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data, 2018.

[4] Z. Deng, Y. Hu, M. Zhu, and et al. A scalable and fast optics for clustering trajectory big data. 18:549–562, 2014. doi: 10.1145/304181.304187. URL `https://doi.org/10.1007/s10586-014-0413-9`.

[5] Dragon. *The Dragon Book*. Acme Publishing, New York, 1st edition, 2012. This is a book.

[6] Hari Krishna Kanagala and V.V. Jaya Rama Krishnaiah. A comparative study of k-means, dbscan and optics. In *2016 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6, 2016. doi: 10.1109/ICCCI.2016.7479923.

[7] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances, 2014.

[8] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2018.

[9] Tomas Mikolov and Quoc Le. Distributed representations of sentences and documents, 2014.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[12] M. Mitra and B. B. Chaudhuri. Information retrieval from documents: A survey, 1999.

[13] Mauritius Much, Frederik Obermaier, Bastian Obermayer, and Vanessa Wormer. So funktioniert das system bahamas. URL `https://www.sueddeutsche.de/wirtschaft/bahamas-leaks-so-funktioniert-das-system-bahamas-1.3172913`. [Accessed 08.08.2023].

[14] Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok Choudhary. Scalable parallel optics data clustering using graph algorithmic techniques. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450323789. doi: 10.1145/2503210.2503255. URL `https://doi.org/10.1145/2503210.2503255`.

[15] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[16] TODO. Fuzziness, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#fuzziness`. [Accessed 15.09.2023].

[17] TODO. Get api, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-get.html`. [Accessed 15.09.2023].

[18] TODO. Dense vector field type, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/dense-vector.html#dense-vector-similarity`. [Accessed 15.09.2023].

[19] TODO. k-nearest neighbor search, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/knn-search.html`. [Accessed 15.09.2023].

[20] TODO. How to deploy a text embedding model and use it for semantic search, . URL `https://www.elastic.co/guide/en/machine-learning/8.10/ml-nlp-text-emb-vector-search-example.html`. [Accessed 15.09.2023].

[21] TODO. Match query, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html`. [Accessed 15.09.2023].

[22] TODO. Standard analyzer, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-standard-analyzer.html`. [Accessed 15.09.2023].

[23] TODO. Synonyms, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html#query-dsl-match-query-synonyms`. [Accessed 15.09.2023].

[24] TODO. Text analysis overview, . URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-overview.html`. [Accessed 15.09.2023].

[25] TODO. Quick start user guide, . URL `https://slurm.schedmd.com/quickstart.html`. [Accessed 16.09.2023].

[26] A. Voit, A. Stankus, S. Magomedov, and I. Ivanova. Big data processing for full-text search and visualization with elasticsearch, 2017.

[27] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39727-4.

[28] V. Zamfir, M. Carabas, C. Carabas, and N. Tapus. Systems monitoring and big data analysis using the elasticsearch system, 2019.

List of Figures

List of Tables

Listing-Verzeichnis

A Anhang

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit den nach der Prüfungsordnung der Universität Kassel zulässigen Hilfsmitteln angefertigt habe. Die verwendete Literatur ist im Literaturverzeichnis angegeben. Wörtlich oder sinngemäß übernommene Inhalte habe ich als solche kenntlich gemacht.

Kassel, September 17, 2023

Klara Maximiliane Gutekunst