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

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

BACHELORTHESIS

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 20, 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	Hiercharical Navigable Small World
OPTICS	Ordering Points To Identify the Clustering Structure
AE	Autoencoder
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
KL	Karhonen-Loéve
SVD	singular value decomposition
PDF	Portable Document Format

Das ist die Einleitung. “Dies ist ein Zitat” [7]. 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`.

```
1 class Foo {
2     String bar;
3 }
```

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 [15], 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/ pre-

processing, 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

[14]

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

[9]

2.2.1 Cosine Similarity

2.2.2 Soft Cosine Similarity

2.2.3 euclidian distance

2.3 Embeddings

[12] [11]

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

2.3.1 Doc2Vec

[11] two flavor of doc2vec: PV-DM and PV-DBOW (https://thinkinfi.com/simple-doc2vec-explained/) [13]

2.3.2 TF-IDF

[17] Test test test

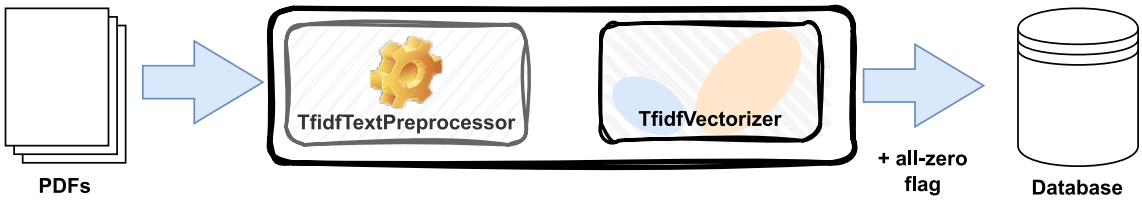


Figure 2.1: TFIDF Preprocessing

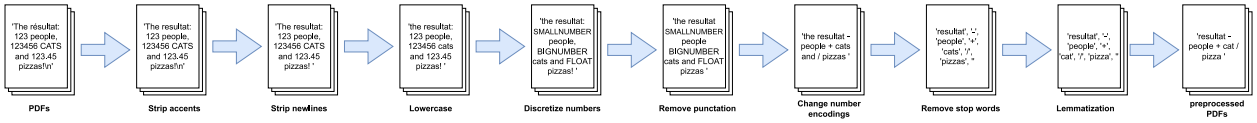


Figure 2.2: TFIDF Preprocessing

2.3.3 Universal sentence encoder

Universal Sentence Encoder (USE) [3]

2.3.4 InferSent

[4]

2.3.5 Hugging face's sentence Transformers

[18]

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 the size of the database when querying db, top image results looked similar, which is how the idea of this section arose

2.5.1 Compression of data

AE

eigenface

According to Turk and Pentland, the idea of Eigenfaces is inspired by information theory. Opposed to former approaches in the domain of face recognition which relied on the classification of images based on a set of predefined facial features, such as distance between

eyes, Eigenfaces does not use predefined features [30]. More specifically, the goal of this approach is to represent images using a smaller set of image features, which best describes and distinguishes between the images [30, 32]. Similar pictures, i.e. of the same person, should lie on a manifold in the lower-dimensional feature space [20]. These features do not necessarily correspond to human facial features [30]. The decomposition of input images not only reduces the complexity but also facilitates modeling probability density of a face image [20].

The input greyscale images are two-dimensional arrays of numbers: $\mathbf{x} = \{x_i, i \in \mathbf{S}\}$, $\mathbf{S}$ being a square lattice [35, 30]. The images are reshaped to an one-dimensional array $\mathbf{x} = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$, where $n = \|\mathbf{S}\|$ and $\mathbb{R}^n$ is the n-dimensional euclidean space [35]. The background is removed in some literature to omit values outside the face area [30]. The original images' dimension is 512x512 [30]/ 64x64 [5], whereas the projected images' dimension is 16x16 [30]/ 250 [5].

The next step is to find an alternative lower-dimensional representation of the images, which preserves most of the information of the original image. The lower-dimensional representation ought to provide the possibility to distinguish between the images [30]. In mathematical terms, this decomposition can be expressed as $\mathbf{x} = \sum_{i=1}^{n} \hat{x}_i \mathbf{e}_i$, $\hat{x}_i$ being inner product of $\mathbf{x}$ and $\mathbf{e}_i$, $\mathbf{e}$ being an orthogonal basis [35]. If all basis vectors are used, the original image can be reconstructed using a linear combination of the basis vectors [30, 5]. The number of basis vectors is limited by the minimum of the training set size N [30] and the number of pixels n [5]. In order to compress the input from a $n-$ to a $m-$, given $m \ll n$, dimensional space, only the first m basis vectors are used. m is chosen such that $\hat{x}_i$ is small for $i \geq m$ [35]. The compressed version of the image is denoted $\mathbf{x} \simeq \hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, ..., \hat{x}_m]^T$. In other words: The compressed image is a vector of the first m weights of the linear combination of weight and basis vectors which can be used to transform the image back to the original space [30]. The decomposition is a projection of the face images onto a feature space or so-called face space spanned by the first m basis vectors [30].

In the context of Eigenfaces one basis used for decomposition is the Karhonen-Loéve (KL) basis, i.e. Principal Component Analysis (PCA) [35, 30]. According to Zhang et al., the KL representation is optimal in the sense that it minimizes the mean squared error between the original image and the compressed image calculated using $m < n$ orthogonal vectors. The KL basis is a vector of the eigenvectors of covariance matrix $\mathbf{C} = E\left[\mathbf{x}\mathbf{x}^T\right]$ of the input images $\mathbf{x}$ [35]. Since these eigenvectors can have facial features, they are called *eigenfaces*. To determine the number of Eigenfaces m used to compress the input images, the cumulative explained variance of the first $i \leq n$ eigenvectors (sorted by eigenvalues λ_i) is calculated [35, 5, 19]. The eigenvalues λ_i can be interpreted as the amount of variance explained by the corresponding eigenvector $\mathbf{e}_i$, which is equivalent to information or entropy. The user can choose how much variance, i.e. information, should be preserved,

by choosing m such that the explained variance is greater than the chosen threshold. Sudiana et al. use a threshold of 90%.

In order to reduce calculation complexity, C is approximated. There are different approaches to the approximation. Turk and Pentland [30] denote the approximation of C as $\frac{1}{N}\sum_{k=1}^{N}\Phi_k\Phi_k^T$, N being the number of training images, $\Phi_k = \mathbf{x}_k - \psi$ being the difference of the k-th training image and the average image $\psi = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k$. Zhang et al. propose an approximation according to $\frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k\mathbf{x}_k^T$ and thus, $\mathbf{C} \simeq \frac{1}{N}\mathbf{X}\mathbf{X}^T$, with $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^n$ [35]. The section below refers to the approach from Zhang et al..

Finding the eigenvectors of $\mathbf{X}\mathbf{X}^T$ is still computationally expensive, since $\mathbf{X}\mathbf{X}^T$ is a n by n matrix. The key idea to solve this problem is singular value decomposition (SVD) [35]. According to Zhang et al., the eigenvectors of $\mathbf{X}\mathbf{X}^T$ can be calculated by using the eigenvectors of $\mathbf{X}^T\mathbf{X}$. Hence, the problem is reduced to a N by N matrix, which is computationally less expensive to solve, since $N \ll n$. The eigenvalues $\mathbf{e}_i \in \mathbb{R}^n$ of $\mathbf{X}\mathbf{X}^T$ can be derived from the eigenvectors $\mathbf{v}_i \in \mathbb{R}^N$ of $\mathbf{X}^T\mathbf{X}$ by $\mathbf{e}_i = \frac{1}{\sqrt{\lambda_i}}\mathbf{X}\mathbf{v}_i$ as discussed in more detail in [35].

In the literature, face images are classified by comparing their position in the face space with those of already known faces [30].

According to [30], this approach performs well on datasets with little variation in pose, lighting and facial expression. However, Zhang et al. state, that the performance deteriorates if the variations increase since the changes introduce a bias that makes the distance function used to make classifications a no longer reliable measure.

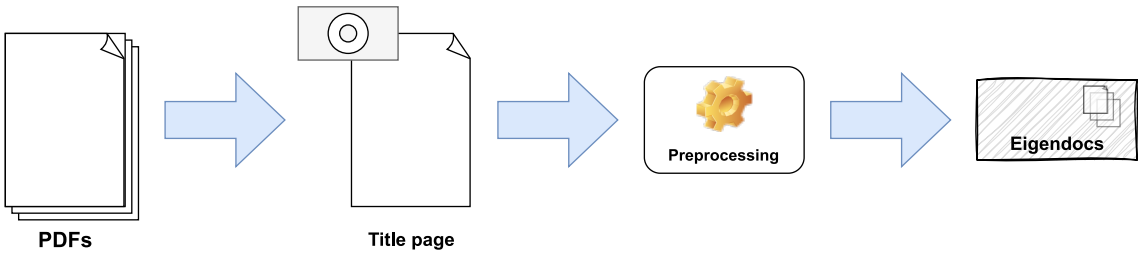


Figure 2.3: From Portable Document Formats (PDFs) to Eigendocs. Firstly, the first page of a document is converted to an image. Then the image is preprocessed: It is placed on a white canvas, to ensure all images have the same dimensions. Moreover, it is converted to greyscale. Afterwards, the 2d image is reshaped to a 1d array. Lastly, the image is compressed using Eigenfaces.

In this work, the Eigenfaces approach is used to compress the images of the first page of documents. The idea is that documents not only hold textual information but also visual information, such as layout, company logo or signature. By mapping those images on a subspace, they ought to be grouped by visual similarity. The procedure of the eigenface adaption *eigendocs* is displayed in Figure 2.3.

2.5.2 Clustering

Clustering is used in a variety of domains to group data into meaningful subclasses, i.e. clusters [16, 6, 8]. 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 [16]. The similarity is measured by a distance function, which is dependent on the data type. Common distance functions are the Euclidean distance, the Manhattan distance and the Minkowski distance [8].

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

- **Hierarchical clustering**: Algorithms, that create spherical or convex-shaped clusters, possibly naturally occurring. A terminal condition has to be defined beforehand. Examples include CLINK, SLINK [6] and Ordering Points To Identify the Clustering Structure (OPTICS) [16].

- **Partitional based clustering**: Algorithms, that 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.

- **Density based clustering**: Density is defined as the number of objects within a certain distance of each other [8]. The resulting clusters can be of arbitrary shape and size. The algorithm usually chooses the optimal number of clusters given the input data. However, some algorithms are sensitive to input parameters, such as radius, minimum number of points and threshold. Popular examples are Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and OPTICS.

- **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 [6].

Multiple approaches below use the term ε-neighbourhood, which is defined as the set of all objects within a certain distance ε of a given object [16]. In other words: $N_\varepsilon(x) = \{y \in X | dist(x, y) \leq \varepsilon, y \neq x\}$.

KMeans

The goal of KMeans is to partition the data into $k \in \mathbb{N}$ clusters, k is given apriori [8]. First, k centroids, i.e. cluster center, are randomly initialized. Then, the objects are assigned to

the closest centroid. Afterwards, the centroids are updated by calculating the mean of the assigned objects. The process is repeated until the terminating condition, for instance, no more change in the clusters, is met [8]. By iteratively reassinging the objects to the closest centroid and updating the centroids, the algorithm minimizes the within-cluster sum of squared errors E, i.e. the sum of squared distances between objects in a cluster and their centroid μ_i, calculated in Equation 2.1 from [8], where C_i is the i-th cluster.

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{2.1}$$

Kanagala and Krishnaiah claim, that KMeans will not identify outliers.

DBSCAN

The clusters identified by DBSCAN have a high density and are separated by low-density regions [8]. In order to create clusters of minimum size and density, DBSCAN distinguishes between three types of objects [8]:

- **Core objects**: An object x with at least $minPts$ objects in its ε-neighbourhood $N_\varepsilon(x)$. $N_\varepsilon(x)$ contains all objects within radius ε of x, ε being the so-called generating distance [16]. In other words: The neighbourhood of x has to exceed a certain threshold for x to be considered a core object, i.e. $|N_\varepsilon(x)| \geq minPts$ is true.

- **Border objects**: An object with less than $minPts$ objects in its ε-neighbourhood, which is in the ε-neighbourhood of a core object.

- **Noise objects**: An object, which is neither a core object nor a border object.

Kanagala and Krishnaiah define $y \in X$ as directly density reachable from $x \in X$, if y is in the ε-neighbourhood of core object x [8]. Moreover, a point $y \in X$ is density reachable from $x \in X$, if there is a chain of objects $x_1, ..., x_n$ with $x_1 = x$ and $x_n = y$, which are directly density reachable from each other as displayed in Figure 2.4 [8].

The points $x \in X$ and $y \in X$ are said to be density connected, if there is an object o, from which both x and y are density reachable [8]. Density connectivity is visualized in Figure 2.5.

The DBSCAN algorithm starts by labeling all objects as core, border or noise points. Then, it eliminates noise points and links all core points, which are within each other's neighbourhood [8]. Groups of connected core points form a cluster [8]. At the end every

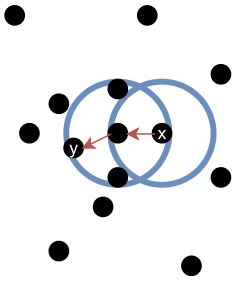


Figure 2.4: Density reachability cf. [2]. The point $y \in X$ is density reachable from $x \in X$, since there is a chain of directly density reachable objects x, o, y.

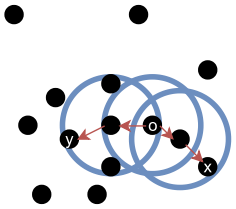


Figure 2.5: Density connectivity cf. [2]. The objects x and y are density connected since there is an object o, from which both x and y are density reachable.

border point is assigned to a cluster [8]. The non-core point cluster assigning is non-deterministic [16]. This algorithm creates clusters as a maximal set of density-connected points [8].

According to Kanagala and Krishnaiah, DBSCAN can identify outliers or noise. However, the algorithm is sensitive to the input parameters $minPts$ and ε and has difficulties distinguishing closely located clusters [8]. Moreover, if one wants to obtain hierarchical clustering, one has to run the algorithm multiple times with different ε, which is expensive in terms of memory usage [16].

OPTICS

OPTICS does not return an explicit clustering, but rather a density-based clustering structure of the data, which is equivalent to clustering results of a broad range of parameters [2]. The idea of Ankerst et al.'s approach is that real-world datasets cannot be described by a single global density, since they often consist of different local densities, as displayed in Figure 2.6.

Opposed to DBSCAN, OPTICS is able to detect clusters of varying densities [6]. OPTICS produces an order of the elements according to the distance to the already added elements [6, 16]: The first element added to the order list is arbitrary. ε defines the neighbourhood radius, i.e. the maximum distance between two elements, which are still considered to be in the same neighbourhood [8]. The order list is iteratively expanded by adding the element of the ε-neighbourhood to the order list, which has the smallest distance to any of

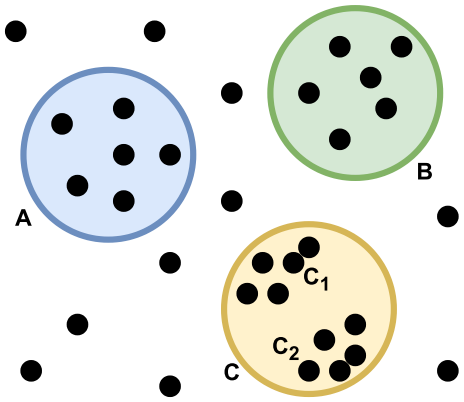


Figure 2.6: Clusters of different densities cf. [2]. Since C_1 and C_2 have different densities than A and B, a clustering algorithm using one global density parameter would detect the clusters A, B and C, rather than A, B, C_1 and C_2 .

the elements already in the order list. Hence, clusters with higher density, i.e. lower ε, are added first (prioritized) [8, 2]. When there are no more elements in the ε-neighbourhood to add, the process is repeated for the other clusters. The non-core point cluster assigning is non-deterministic [16].

$$RD(y) = \begin{cases} \text{NULL} & \text{if } |N_\varepsilon(x)| < minPts \\ max(core\_dist(x), dist(x,y)) & \text{otherwise} \end{cases} \tag{2.2}$$

OPTICS saves the reachability distance $RD(y)$, as calculated in Equation 2.2 from [16], with core distance core_dist being the minimal distance ε^{min} such that $|N_{\varepsilon^{min}}(x)| \geq minPts$ (the distance to the $minPts^{th}$ point in N_ε) or NULL else, of each element to its predecessor in the order list and thus, a representation of the density necessary to keep two consecutive objects in the same cluster [16]. If $\varepsilon < RD(y)$, then y is not density reachable from any of its predecessors and thus, one can determine whether two points are in the same cluster for given information saved by OPTICS [16, 2]. If the core distance of an element is not NULL, i.e. it is a core object, and it is not density reachable from its predecessors, it is the start of a new cluster [2]. Otherwise, the element is a noise point [2]. According to Patwary et al., the algorithm builds a spanning tree, which enables obtaining the clusters for a given ε by returning the connected components of the spanning tree after omitting all edges with $\varepsilon < RD(y)$ [16]. The relationship between ε, cluster density and nested density-based clusters is displayed in Figure 2.7.

Hence, this procedure enables the extraction of clusters for arbitrary $0 \leq \varepsilon_i \leq \varepsilon$ [8, 2]. According to Patwary et al.'s work, even though the clustering algorithm is expensive the extraction only needs linear time. According to [2], the algorithm yields good results if the input parameters $minPts$ and ε are "large enough" and thus, the algorithm is rather insensitive to the input parameters.

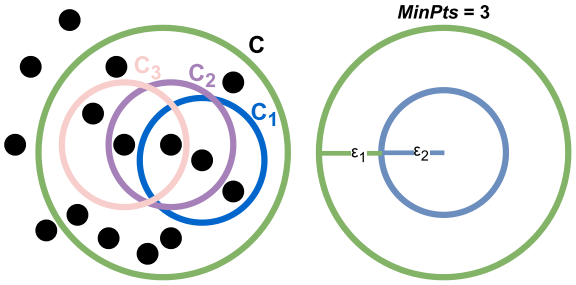


Figure 2.7: The relationship between ε, cluster density and nested density-based clusters cf. [2]. For a constant $minPts$, clusters with higher density such as C_1, C_2 and C_3, i.e. a low ε_2 value, are completely contained in lower density clusters such as C given $\varepsilon_1 > \varepsilon_2$. This idea forms the basis of OPTICS of expanding clusters iteratively and thus, enables the detection of clusters for a broad range of neighbourhood radii $0 \leq \varepsilon_i \leq \varepsilon$.

The smaller ε is chosen, the more objects will be identified as noise and thus, the algorithm will not identify clusters with low density, since some objects only become core objects for a larger ε [2]. According to Ankerst et al., the optimal value for ε creates one cluster for most of the objects with respect to a constant $minPts$, since information about all density-based clusters for $\varepsilon_i < \varepsilon$ is preserved. A heuristic for choosing ε based on the expected k-nearest neighbour distance is presented in [2].

High values for $minPts$ smoothen the reachability curve, even though the overall shape stays roughly the same [2]. According to Ankerst et al., the optimal value for $minPts$ is between 10 and 20.

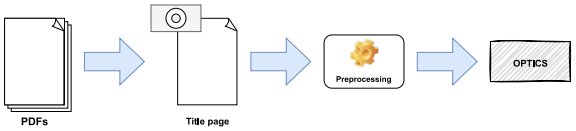


Figure 2.8: First, the first page of each document is converted to an image. Then the image is preprocessed: There are two different preprocessing approaches were used: The first approach resizes the image to a 32x32 format when reading it initially. The second approach uses the Eigendocs approach (i.e. an adaption of Eigenfaces) to reduce the image to a 2x2 format. In both cases, the compressed image is converted to greyscale.

Similar to the approach from [2], OPTICS was used to cluster the images of the first page of documents in this work. The procedure is displayed in Figure 2.8. There were two different preprocessing approaches:

1. The images were preprocessed to 32x32 greyscale pixels as visualized in Figure 2.9.

2. The technique Eigendocs from subsubsection 2.5.1 was used to compress the images to 2x2 greyscale images as displayed in the fourth row of Figure 2.10.

The reachability distance ordered by OPTICS is displayed in Figure 2.11. The resulting clusters are displayed in Figure 2.12. Example instances of both clusters (cluster and noise) are displayed in Figure 2.13 and Figure 2.14. TODO: compare preprocessing results

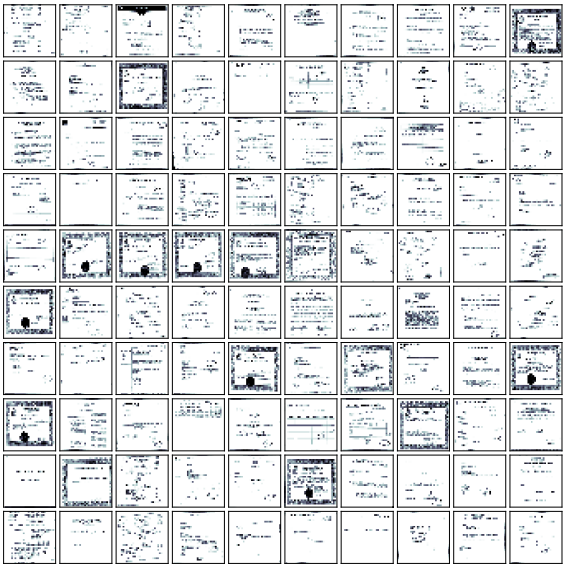
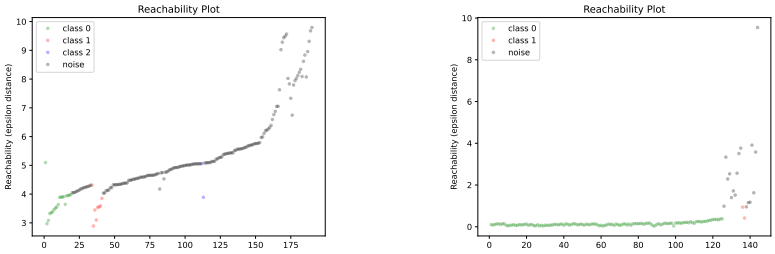


Figure 2.9: The first 100 preprocessed documents of the dataset. They were preprocessed in order to have the same characteristics as the images used in [2]. The images were preprocessed as discussed in item 1 to 32x32 greyscale pixels, which drastically reduced the quality of the images.

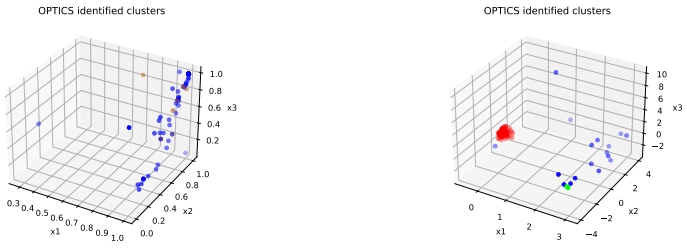
Figure 2.10: The first 10 preprocessed documents of the dataset. The original images are displayed in the first row. The second row shows the reconstructed images using the compressed images from the fourth row. The third row shows the reconstruction error, i.e. the difference between the reconstructed and the original image. The fourth row shows them in their compressed 2x2 greyscale form as discussed in item 2. The last row presents the greyscale values of the compressed image as a line.



(a) The reachability plot of the documents preprocessed according to item 1.

(b) The reachability plot of the documents preprocessed according to item 2.

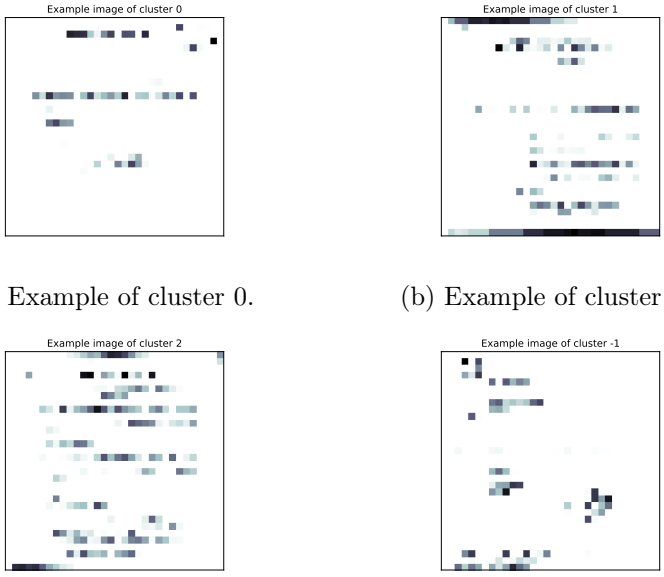
Figure 2.11: The plot was created using the OPTICS algorithm from the Python library scikit-learn. The plot shows the reachability distance of each document to its predecessor in the order list. The reachability distance is the minimum distance necessary to keep two consecutive objects in the same cluster. The plot shows that the documents are divided into a cluster and a noise region.



(a) The clusters identified by OPTICS of the documents preprocessed according to item 1.

(b) The clusters identified by OPTICS of the documents preprocessed according to item 2.

Figure 2.12: The clusters were extracted from the respective reachability plot in Figure 2.11. The blue points are noise points, whereas any other colour denotes a cluster.



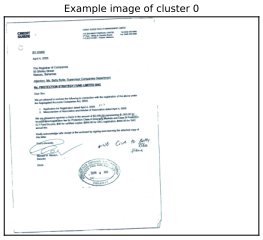
(a) Example of cluster 0.

(b) Example of cluster 1.

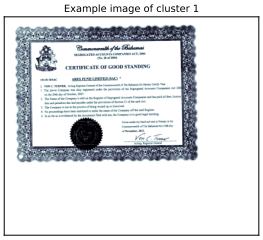
(c) Example of cluster 2.

(d) Example of noise.

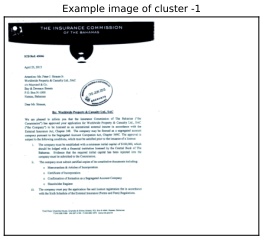
Figure 2.13: The clusters were identified by OPTICS of the documents preprocessed according to item 1.



(a) Example of cluster 0.



(b) Example of cluster 1.



(c) Example of noise.

Figure 2.14: The clusters were identified by OPTICS of the documents preprocessed according to item 2.

3 Implementation

3.1 Slurm

Slurm is an open-source management tool for Linux clusters [29]. It allocates resources, i.e. compute nodes, and provides the means to start, execute and monitor jobs [29, 33].

The so-called slurm daemons control nodes, partitions, jobs and job steps [29]. According to TODO, a partition is a group of nodes and a job is the allocation of resources, i.e. compute nodes, to a user for a limited period of time. A basic visualization of the architecture is given in Figure 3.1.

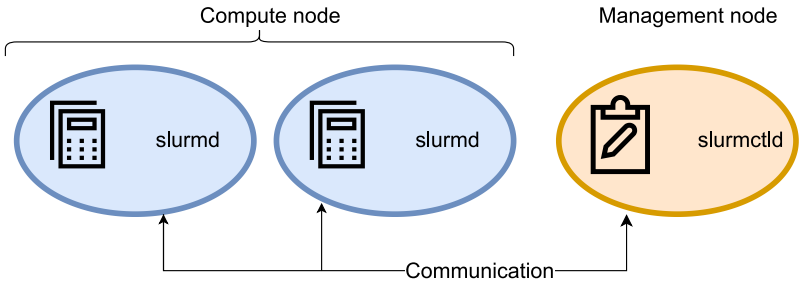


Figure 3.1: Slurm architecture. The management node has a `slurmctld` daemon, while every compute node has a `slurmd` daemon. The nodes communicate. The user can use certain commands, for instance `srun` and `squeue`, anywhere on the cluster.

3.2 Database Elasticsearch

Elasticsearch is a widely used non-relational database, which was designed to store and perform full-text search on a large corpus of unstructured data [31]. This open-source distributed document-driven database system is built in Java and is based on the Apache Lucene (Java) library for high-speed full-text search [31, 34]. According to Zamfir et al., 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. It enables near real-time search by index refreshing periods of one second. Needless to say, Elasticsearch is qualified to handle Big Data.

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

field name	field description
_id	Unique identifier of document `i`. The identifier is generated by the sha256 hash algorithm from hashlib.
doc2vec	55 dimensional doc2vec embedding of `i`.
sim_docs_tfidf	sim_docs_tfidf embedding + all-zero flag of `i`. The all-zero flag is one if the TF-IDF embedding consists of only zeros, zero else.
google_univ_sent_encoding	512 dimensional google_univ_sent_encoding embedding of `i`.
huggingface_sent_transformer	384 dimensional huggingface_sent_transformer embedding of `i`.
inferSent_AE	inferSent_AE embedding of `i`. Since the pretrained infersent model embedding's dimension is 4096, the encoder of a trained Autoencoder (AE) is added to reduce the dimension to 2048.
pca_image	Two dimensional 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	Base64 encoded image of first page of `i`.

Elasticsearch's entries, i.e. documents, are stored in logical units, so-called indices. As stated by Zamfir et al. and Voit et al., the indices are structured similarly to Apache Lucene's inverted index format. An index can be spread into multiple nodes. A node is single running instance of Elasticsearch [34]. An index is divided into one or more shards, which can be stored on different servers and enable parallelization [34].

Elasticsearch indices' entries are documents, which are saved in a JavaScript Object Notation (JSON) format [31]. 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 [34].

Replicas are copies of shards, which create redundancy and thus, ensure availability [34].

The database is filled once with data from a large unstructured corpus of PDF files. After the initialization of the database, it is used for queries. Therefore, the workflow is completely offline.

was kann Elasticsearch vs. was will ich? PDFs werden nicht in DB gemacht 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.

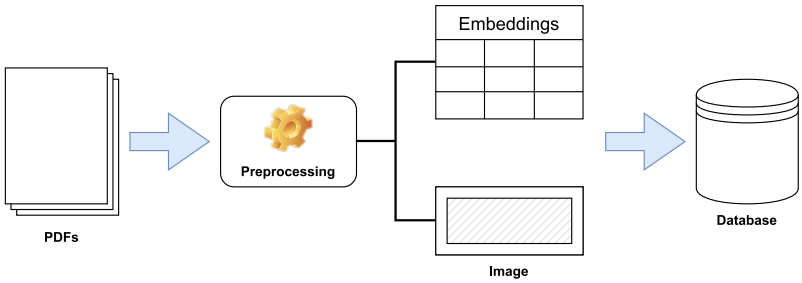


Figure 3.2: PDFs to Database. First, the data is preprocessed: The first page of a PDF file is converted to an image and the complete text is extracted. The images are stored in the database as well as the text and different embeddings of the text.

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 parameters `_source_excludes` or `_source_includes` may be used to exclude or include specific fields of the document in the response [22].

The keyword used when performing a full-text search 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 [28]. The default preprocessing steps of the so-called default analyzer include tokenization and lowercasing [28]. Omitting stop words is disabled by default, but it is possible to provide custom stop words or use the English stop word list [28]. It is possible to create custom tokenizers, which split the query value into tokens of a certain maximum length. In this work, the default analyzer 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-term synonym expansion. When the user queries a specific phrase Elasticsearch expands the query to include synonyms of the query terms [27]. The maximum number of expansion terms is set to 50 by default but can be configured by the user [26]. By default, the multi-terms synonym expansion option is enabled [26].

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 that have the word *Bahamas*. By default, this option is not enabled but can be enabled and configured individually by the user [26]. 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 [21]. 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 [21].

Another search option of Elasticsearch is the k-nearest neighbor (kNN) search. The return value of a kNN search is the `k` nearest neighbors in terms of a certain distance function of a query vector [10]. 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 [24], the kNN either returns the exact brute-force nearest neighbors or approximate nearest neighbors calculated by the Hiercharical Navigable Small World (HNSW) algorithm [10, 24]. 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 [10]. The term `navigable` refers to the graphs used, which are graphs with (poly-)logarithmic scaling of links traversed during greedy traversal concerning the network size [10]. 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 [10]. 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 [10].

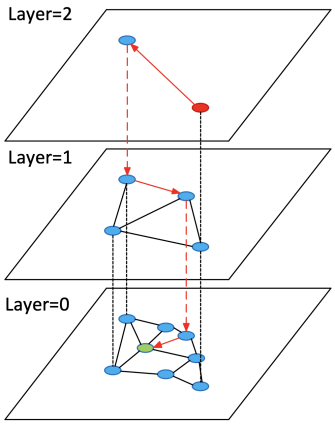


Figure 3.3: Structure of HNSW layer from [10]. The search starts on the uppermost layer, i.e. the layer containing the longest links, greedily traversing the layer until reaching the local minimum. The local minimum is used as the starting point at the next lower layer and the process is repeated until 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 [24]. The similarity measure used in this work is the cosine similarity, which calculates the `_score` of a document according to Equation 3.1 from [23], where `query` is the query vector and `vector` is the vector representation of the document in the database. Since

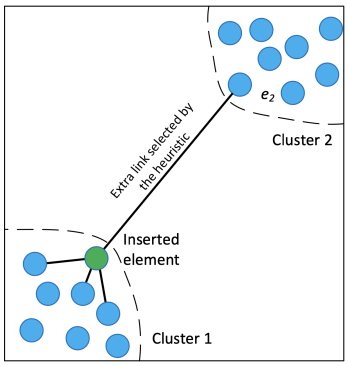


Figure 3.4: Neighbour selection heuristic of HNSW from [10]. The heuristic creates diverse links, i.e. links between close elements (e.g., green circle and elements in cluster 1) and between isolated clusters (e.g., green circle and e_2) to ensure global connectivity.

cosine is not defined on vectors with zero magnitude, embeddings that can return all zero vector representations, such as sim_docs_tfidf, are enhanced with an all-zero flag in this work.

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

Elasticsearch's kNN implementation not only allows literal matching on search terms but also semantic search [24]. 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 [25]. 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 that contains an embedding is used.

3.3 User Interface

3.3.1 Backend

Flask

3.3.2 Frontend

angular

3.4 Trade-off between memory and query time

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] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, jun 1999. ISSN 0163-5808. doi: 10.1145/304181.304187. URL `https://doi.org/10.1145/304181.304187`.

[3] 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.

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

[5] Laura Dayton, Dante Rousseve, Neil Sehgal, and Sindura Sriram. Csci 1430 final project report: Methods of facial recognition, 2020.

[6] 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`.

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

[8] 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.

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

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

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

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

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

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

[15] 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].

[16] 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`.

[17] Robert-George Radu, Iulia-Maria Rădulescu, Ciprian-Octavian Truică, Elena-Simona Apostol, and Mariana Mocanu. Clustering documents using the document to vector model for dimensionality reduction. In *2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pages 1–6, 2020. doi: 10.1109/AQTR49680.2020.9129967.

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

[19] Dodi Sudiana, Mia Rizkinia, and Fahri Alamsyah. Performance evaluation of machine learning classifiers for face recognition. In *2021 17th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*, pages 71–75, 2021. doi: 10.1109/QIR54354.2021.9716171.

[20] Yichuan Tang and Xuan Choo. Intrinsic divergence for facial recognition, 2008.

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

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

[23] 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].

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

[25] 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].

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

[27] 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].

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

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

[30] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991.

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

[32] Chang Wang and Sridhar Mahadevan. Multiscale manifold learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27:912–918, Jun. 2013. doi: 10.1609/aaai.v27i1.8633. URL https://ojs.aaai.org/index.php/AAAI/article/view/8633.

[33] 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.

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

[35] Jun Zhang, Yong Yan, and M. Lades. Face recognition: eigenface, elastic matching, and neural nets. *Proceedings of the IEEE*, 85(9):1423–1435, 1997. doi: 10.1109/5.628712.

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 20, 2023

Klara Maximiliane Gutekunst