UNIKASSEL VERSITÄT

Intelligent Embedded Systems

Identification of Key Information with Topic Analysis on Large Unstructured Text Data

BACHELOR THESIS

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. Bernhard Sick
Examining board 2:	Prof. Dr. Gerd Stumme
Supervisor:	Dr. Christian Gruhl
Date:	21. November 2023

Abstract

The goal of this thesis is to investigate the applicability of computational means to the exploration of large unstructured text corpora. Finding relevant documents and interconnections between documents becomes significantly more difficult due to the sheer amount of documents available. Institutes, such as the German tax offices, have access to leak data, for instance, the *Panama Papers* or the *Bahamas 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 inherent key topics. Hence, computational means, such as text mining or topic analysis, may help to overcome this obstacle. This thesis proposes an approach to finding relevant documents which share common topics from a large unstructured text corpus. The approach bundles different methods, such as textual embeddings, transformation of images and clustering techniques. As a result of this work, a web interface that enables the comparison of the methods examined via queries for similar documents to a database is provided.

Zusammenfassung

Das Auffinden relevanter Dokumente und von Zusammenhängen zwischen Dokumenten wird durch die enorme Menge an verfügbaren Dokumenten erheblich erschwert. Institutionen, wie z.B. deutsche Finanzämter, haben Zugang zu Datenleaks, wie etwa den *Panama Papern* oder dem *Bahamas-Leak*, die große Mengen an Dokumenten und wertvollen Informationen enthalten, die es zu extrahieren gilt. Diese Institute, Unternehmen und Einzelpersonen verfügen jedoch nicht über ausreichende Ressourcen, um einzelne Dokumente zu durchsuchen, ein bestimmtes Dokument zu finden oder inhärente Themen zu identifizieren. Daher können computergestützte Verfahren wie *Text Mining* oder *Topic analysis* sie dabei unterstützen. In dieser Arbeit wird ein Ansatz vorgestellt, der in einem großen unstrukturierten Textkorpus relevante Dokumente mit gemeinsamen Themen findet. Dieser Ansatz bündelt verschiedene Methoden, wie z.B. textuelle Embeddings, Transformation von Bildern und Clustering-Techniken. Als Ergebnis der in dieser Arbeit untersuchten Methoden wird eine Weboberfläche bereitgestellt, die Abfragen nach ähnlichen Dokumenten zum Vergleich der verschiedenen Methoden ermöglicht.

Contents

List of abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
AE	Autoencoder
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bi-directional Long Short-Term Memory
BoW	Bag of Words
CBOW	Continuous-Bag-of-Words
CSS	Cascading Style Sheet
CSV	Comma Separated Values
Doc2Vec	Document to Vector
DAN	Deep Averaging Network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DNN	Deep Neural Network
GB	Gigabyte
GloVe	Global Vectors
HDBSCAN	Hierarchical DBSCAN
HNSW	Hierarchical Navigable Small World
IDF	Inverse Document Frequency
IES	Intelligent Embedded Systems
IR	Information Retrieval
JSON	JavaScript Object Notation
KL	Karhonen-Loéve
kNN	k-Nearest Neighbour
LDA	Latent Dirichlet Allocation
LSTM	Long Short-Term Memory
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Network
NoSQL	Not only SQL
OPTICS	Ordering Points To Identify Clustering Structure
PCA	Principal Component Analysis
PKL	Pickle
PV-DBOW	Distributed Bag of Words
PVDM	Paragraph Vector Distributed Memory
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RSME	Root Mean Square Error

SBERT	Sentence-BERT
SNLI	Stanford Natural Language Inference
SQL	Structured Query Language
SVD	Singular Value Decomposition
SVM	Support Vector Machine
Top2Vec	Topic to Vector
TF-IDF	Term Frequency - Inverse Document Frequency
TF	Term Frequency
UI	User Interface
UMAP	Uniform Manifold Approximation and Projection
URL	Uniform Resource Locator
USE	Universal Sentence Encoder
VSM	Vector Space Model
Word2Vec	Word to Vector

1 Introduction

This thesis addresses the task of analysing large unstructured text corpora. Their exploration is challenging due to the heterogeneous nature of the data, i.e. different formats and layouts, and the amount of information. It is not possible to find a group of semantically similar documents by traversing corpora manually. Therefore, this thesis explores different approaches to support the exploration of large unstructured text corpora by computational means with the goal of grouping documents according to their similarity.

The dataset inspected in this thesis is the so-called *Bahamas leak*. It is a collection of roughly 38 Gigabyte (GB) of fiscal documents, which were leaked in 2016 [49]. The documents are unstructured, i.e. they are of different types, content and layout. They are relevant in the context of tax fraud since they contain information about offshore companies and their owners. Tax offices examine this dataset to identify tax evasion. However, it has proven to be challenging to identify relevant documents and their interconnections due to the amount of documents contained in the leak.

1.1 Motivation

On a broader scope, this thesis aims to provide computational means to facilitate the work with large unstructured text data for humans. The goal is to actively use Machine Learning (ML) techniques to analyze a large text corpus and to reduce the amount of manual human work.

In the context of tax fraud, large unstructured text data, such as the Bahamas leak is examined by tax offices. However, tax offices do not have sufficient resources to examine all documents to find an object of interest, for instance, an invoice. Hence, ML techniques ought to facilitate the work of investigators by reducing manual labor. These methods should propose visually, i.e. of the same document type, or semantically, i.e. from the same company, similar documents to the investigator.

1.2 Research Questions

In order to support the exploration of large unstructured text data, this thesis aims to provide computational means to facilitate the work with large corpora. In this work, different methods to derive semantic and visual information from unstructured text data

are applied. These techniques ought to be compared and evaluated. In the following, the research questions addressed are defined:

RQ1. **Is it possible to use a visual representation to find similar documents in the corpus?**
Assuming that it is valuable to explore documents of similar type, for instance, invoices, simultaneously, the system should be able to find similar documents with respect to their visual appearance. It remains to be seen whether encodings of the visual appearance of a document are sufficient to find similar documents.

RQ2. **Do different embedding methods produce similar results?**
The task at hand defines a result as a set of response documents similar to a query document. Hence, one has to compare response sets of different methods. The similarity between to response documents can be evaluated with respect to the content or the visual appearance of the documents.

RQ3. **How are the results of the system presented to experts?**
This question aims to find a suitable way to present the results of the system to the user in an intuitive manner.

RQ4. **How can the performance of the system be evaluated?**
Since the dataset is not labeled, the performance of the system cannot be evaluated with respect to a ground truth. Hence, other means of evaluation have to be found. These techniques could include time measurements or qualitative analysis of the query responses.

1.3 Own approach

This thesis proposes an approach to group documents based on their appearance or semantic similarity, which is defined via different embedding strategies, i.e. methods to derive embeddings from texts. Embeddings are numerical representations of words, sentences or texts. They enable the comparison of heterogeneous data via cosine similarity, i.e. the angle between embedding vectors, whereas visual information is clustered using different approaches including OPTICS (cf. Subsection 3.6.2) beforehand. The resulting groups of documents are visualized using word clouds.

This work's goals include the implementation of a User Interface (UI) for the techniques examined. However, this UI is not supposed to be an operational application for end users from the tax office but serves the purpose of displaying the techniques examined. It should assist the natural human approach to exploration: A human finds a document of interest, for instance, by keyword search, and thus, wants to find similar documents. The tool should support keyword search, a detailed inspection of a document of interest and the exploration of similar documents.

1.4 Structure of the Thesis

The thesis is structured as follows.

Chapter 1

Firstly, the problem of working with large unstructured text corpora is introduced. Secondly, the dataset used in this thesis is described. Moreover, the goal of this thesis, as well as the target audience of the problem investigated is stated. Afterwards, the motivation and research questions are presented. The chapter concludes with an outlook on the techniques used and an overview of the thesis.

Chapter 2

This chapter covers related work where similar approaches are presented. Moreover, the chapter introduces the literature that serves as a basis for this thesis.

Chapter 3

The theoretical foundations of the techniques applied in this thesis are outlined in Chapter 3. The techniques can be divided into preprocessing (cf. Section 3.1), semantic embeddings (cf. Section 3.2), similarity measurements (cf. Section 3.3), topic analysis (cf. Section 3.4), compression of data (cf. Section 3.5), clustering algorithms (cf. Section 3.6) and software frameworks (cf. Section 3.7).

Chapter 4

This chapter describes the implementation of the methods. The implementation is based on the theoretical foundations presented in Chapter 3. On a more granular level, this chapter covers the offline preprocessing (cf. Section 4.1), the implementation of the UI (cf. Section 4.2) and the trade-off between memory and query time in Section 4.3.

Chapter 5

The evaluation of the methods is presented in this chapter. It gives a reason why certain parameter choices were made with respect to established parameter estimation approaches. Moreover, it compares the different methods with regard to their query responses and the bundle of methods constructed in the course of this thesis to an existing baseline topic analysis approach.

Chapter 6

This chapter concludes this thesis. The insights acquired by exploring different techniques with the goal of the exploration of large unstructured text data are presented and the research questions are revised in Section 6.1. In Section 6.2 the scientific contributions are highlighted.

Chapter 7

The last chapter gives an outlook on future work. It also includes a discussion of the limitations of this thesis.

2 Related work

This chapter examines and summarises different literature about topic analysis of text corpora and related fields. It presents a selection of textual embedding methods, visual information encodings, dimensionality reduction methods, similar data corpora, similarity measures and clustering methods. Moreover, a few topic analysis libraries are presented.

The domain of Information Retrieval (IR) works on large datasets. Hence, multiple scientific papers working with large corpora, such as [44], have been published. Research on (large) fiscal datasets includes ML tasks such as anomaly detection to identify credit card fraud [75, 48, 47, 39, 60, 27].

ML techniques usually require numerical data as input. In order to utilize ML techniques, textual data is often represented as real-valued vectors. Depending on the approach, vectors either represent single words, sentences or whole documents. The models used in this work are briefly introduced in the following. More detailed information can be found in Section 3.2.

The TF-IDF model is a widely used model for text representation. Even though Zhang et al. discuss TF-IDF's drawbacks [77] the model is incorporated in this work due to its simplicity.

Mikolov et al. discuss the well-established Word2Vec models CBOW and Skip-gram. The authors found that these Word2Vec models produce high-quality word embeddings on large datasets [44]. These Word2Vec models form the basis of so-called Doc2Vec models which embed whole documents. The PVDM model extends the CBOW model to work on a set of documents or paragraphs instead of words [78] and is used in this work.

A more complex model is the SBERT model [58]. This model is an extension of the BERT model which set state-of-the-art results in many Natural Language Processing (NLP) tasks. Reimers and Gurevych show that BERT is not suitable for certain similarity measures, such as cosine similarity. Moreover, they argue that SBERT overcomes BERT's shortcomings. Since the SBERT model is able to produce document embeddings, it is used in this work.

The InferSent model is a sentence embedding model [14]. Conneau et al. state that it outperforms models trained in an unsupervised fashion. They train it on a labeled dataset and optimize the model's architecture. Since the model is pretrained and open-source, it is used in this work.

Another embedding model of interest is the USE model [9]. Cer et al. propose two model architectures which respectively are either superior with regard to accuracy or resource

consumption. They claim that their model surpasses word-level embedding transfer learning on several NLP tasks. Due to the fact that the pretrained models are open-source, the one that consumes fewer resources is used in this work.

This thesis aims to encode visual information as low-dimensional real-valued vectors. Since the domain of face recognition deals with the task of deriving meaningful information from high dimensional data, the Eigenfaces approach is adapted to document images in this work. The task of finding similar images of faces is transferred to finding similar document images. Eigenfaces projects face images into a lower-dimensional feature space which best encodes the variation among the faces [67]. Since 1991 this technique has been covered in a lot of papers [67, 76, 71, 65, 16, 64].

Anowar et al. propose a survey on different dimensionality reduction techniques including PCA, LDA and SVD [6]. They conceptually categorize and compare the techniques. The authors conduct experiments on different datasets to compare the techniques' performance on classification tasks. They find that the classification accuracy obtained on the reduced version of the datasets is superior to the accuracy achieved on the original datasets. Their work serves as a theoretical foundation for Section 3.5.

Another dimensionality reduction technique is an AE [46, 40]. The papers provide a theoretical foundation for Subsection 3.5.1. An AE learns a meaningful low-dimensional representation of the input. This representation is used as a compressed version of certain embeddings in this work.

To determine the similarity between two objects, one has to define a metric. Prevalent metrics in the domain of comparing objects in a Vector Space Model (VSM) include (soft) cosine similarity outlined by Sidorov et al., as well as by Charlet and Damnati [62, 11], the Manhattan and the Euclidean norm [37]. Sidorov et al. propose the calculation of the soft similarity. Charlet and Damnati state that the soft cosine similarity is superior to the cosine similarity since it takes into account the relations between words. When comparing different norms in the context of IR from images Khosla et al. find that the Manhattan norm has a better precision than the Euclidean norm. The similarity metric used in this work is the cosine similarity.

The visual information of the document images shall be used to cluster them. Ankerst et al. introduce the clustering algorithm OPTICS which seems to be suitable for the task at hand since it does not return an explicit clustering but a clustering structure [5]. Moreover, Ankerst et al. state that OPTICS is a method for database mining. Other researchers, for instance, Kanagala and Krishnaiah, compare related clustering algorithms including K-Means, DBSCAN and OPTICS. They state that OPTICS overcomes DBSCAN's difficulties and K-Means limitations [35]. Patwary et al., Deng et al. and Agrawal et al. propose OPTICS extensions for spatially and temporally evolving data or a parallel version [54, 17, 2].

The methods explored in this thesis ought to be bundled into a tool. Some researchers have already developed complete topic analysis libraries whose functionalities can be compared to the tool developed in this thesis. They merge a selection of the techniques stated above into a well-reasoned composite. BERTopic is a library that merges SBERT embeddings with UMAP dimension reduction, HDBSCAN clustering and the application of TF-IDF on the clusters [26].

Other well-established topic analysis approaches consist of LDA. Wang and Qian propose a technique that first applies LDA to reduce the data's dimensionality and thereafter classifies the result with a Support Vector Machine (SVM) [72]. Similarly, Chen et al. use a kNN algorithm instead of a SVM on the textual subspace generated by LDA [12]. Another technique proposed is LDA2VEC, which is subject to Churchill and Singh's work [13]. Chaney and Blei's paper introduces an open-source library for topic model visualization, exemplary showcased on a Wikipedia dataset [10].

Angelov and Niu and Dai claim that the Top2Vec model not only overcomes LDA's shortcomings [4, 52] but is also developed for topic analysis on a large collection of documents [4]. The Top2Vec library serves as a baseline model for the application developed in this work.

In contrast to the assumption of this thesis that the prevalent topics are static, in reality, topics may be dynamic or change over time. Not only Alghamdi and Alfalqi, but also Vayansky and Kumar have published surveys on topic analysis techniques, which take into account factors such as time [3, 69].

Some of the techniques that were briefly introduced in this chapter partake in the application developed in this thesis. They are described in more detail in Chapter 3.

3 Fundamentals

The following chapter outlines the theoretical principles of the methods used in this work. First, the preprocessing of the data is described in Section 3.1. Then, a variety of ways to generate numerical representations of textual data is outlined in Section 3.2. Afterwards, the different similarity measurements are introduced in Section 3.3. A selection of conventional topic analysis approaches is outlined in Section 3.4. Subsequently, two data compression techniques are presented in Section 3.5. Then, Section 3.6 presents multiple clustering methods. Finally, the libraries used to implement the web application and the database are introduced in Section 3.7.

3.1 Preprocessing

Similar to other ML domains, NLP requires preprocessing of the data. Usually, textual data contains irrelevant information and noise. Examples of noise include so-called stop words, such as "the" or "and". However, irrelevant information can be task-specific. In some cases, numerical data may be regarded to be irrelevant and should be omitted. Preprocessing improves the performance and the results [56]. The next sections describe the non-trivial preprocessing steps applied in this work.

3.1.1 Tokenization

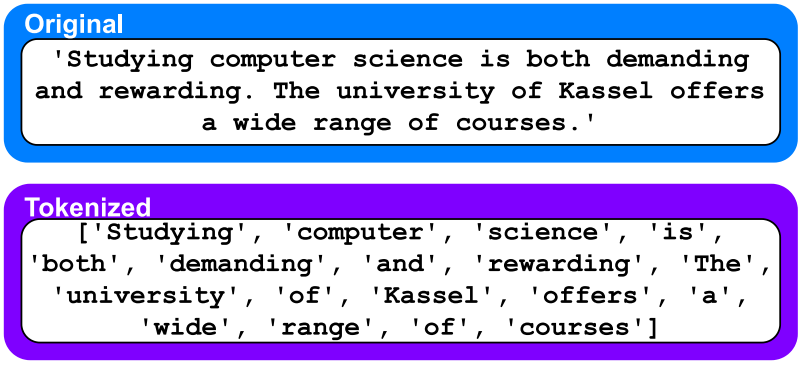


Figure 3.1: Tokenization visualized using an example text.

Tokenization is the process of splitting a text into smaller pieces, so-called *tokens*. Tokens can be words and punctuation marks [8]. The definition of a token depends on the application. For instance, certain tokenization implementations may identify tokens as subsequent series of non-whitespace characters omitting all numbers and punctuation marks [63].

3.1.2 Stop-Word-Removal

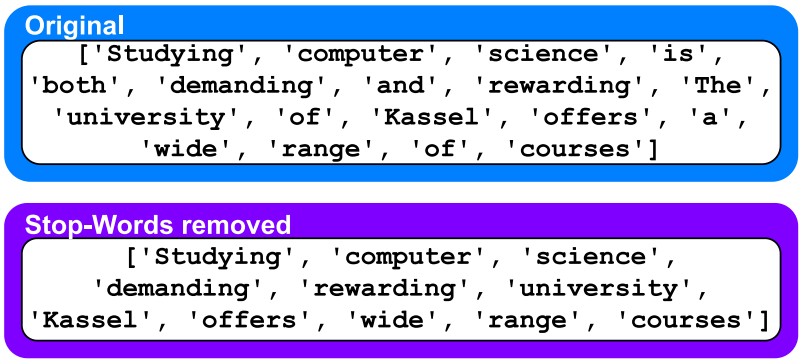


Figure 3.2: Stop-Word-Removal visualized using an example text.

Omitting words that are not relevant to the context of the text is called *stop-word-removal*. Stop words not only depend on the domain but also the language [63].

3.1.3 Stemming

In order to avoid language inflections, i.e. treating words with similar meanings differently, stemming is applied [56]. According to Bird et al., *stemming* is the process of stripping off any affixes, i.e. prefixes and suffixes [63], from a word and returning the stem. Different types of stemmers are better suited for certain applications than others. Hence, the choice of the stemmer depends on the application.

3.1.4 Lemmatization

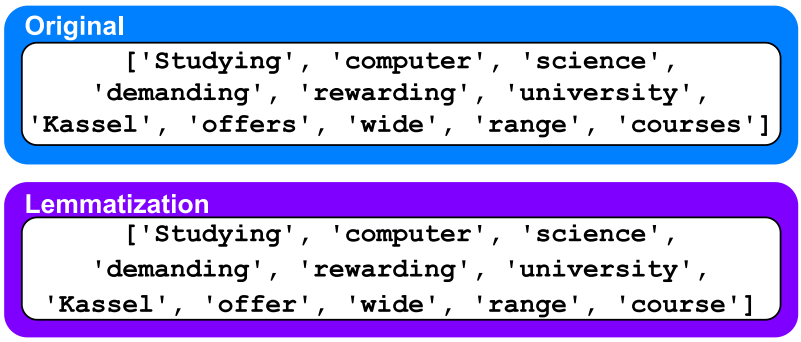


Figure 3.3: Lemmatization visualized using an example text.

Stemming and lemmatization are used to reduce the vocabulary size [56]. Opposed to stemming, lemmatization returns only stems that are considered valid words [8]. Some implementations of lemmatizers validate stems with regard to a set of valid words, i.e. *lemmas*, stored in a dictionary. Lemmatizers are usually slower than stemmers [8].

The *WordNetLemmatizer* from the `nltk` package[1] requires a vocabulary. According to Radu et al., it is frequently used for lemmatization of English texts [56].

3.2 Embeddings

Usually, ML techniques require textual inputs to be converted to embeddings [43]. Embeddings are numerical representations of words, sentences or texts. They can be used to present the textual data as real-valued vectors in a VSM. A simple example of a VSM in the NLP context is shown in Figure 3.4. A VSM is a N-dimensional space [62]. VSMs are commonly used due to their conceptual simplicity and because spatial proximity correlates with semantic proximity [77, 9, 58, 4]. Representations in a VSM can improve the performance in NLP tasks [45]. The following section outlines the fundamentals of a selection of embeddings.

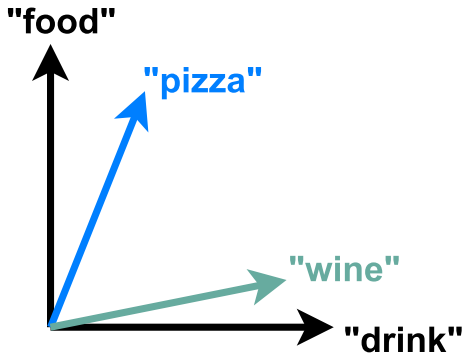


Figure 3.4: A simple VSM. The words are represented as vectors in a two-dimensional space. Since *wine* is semantically more similar to *drink* than to *food*, the vectors are closer together.

3.2.1 Neural Networks

A Neural Network (NN) is a ML model which consists of multiple layers of nodes. A node, or so-called neuron, takes an input vector and produces an output vector. The output is derived from the calculation of a weighted sum of the inputs and an activation function [32]. The architecture of a NN is shown in Figure 3.5. The first and last layers are called input and output layers, respectively. The layers between the input and output layers are called hidden layers. If a NN has more than one hidden layer, it is called a Deep Neural Network (DNN) and working with DNNs is considered deep learning. To propagate the input through the network the layers are connected. In a *feed-forward* NN, the information flows from the input layer to the output layer [31].

NNs are trained using the backpropagation algorithm which reduces the error between the predicted and the actual output iteratively. While data is propagated in a forward direction

[1] https://www.nltk.org/_modules/nltk/stem/wordnet.html (last accessed: 12/11/2023)

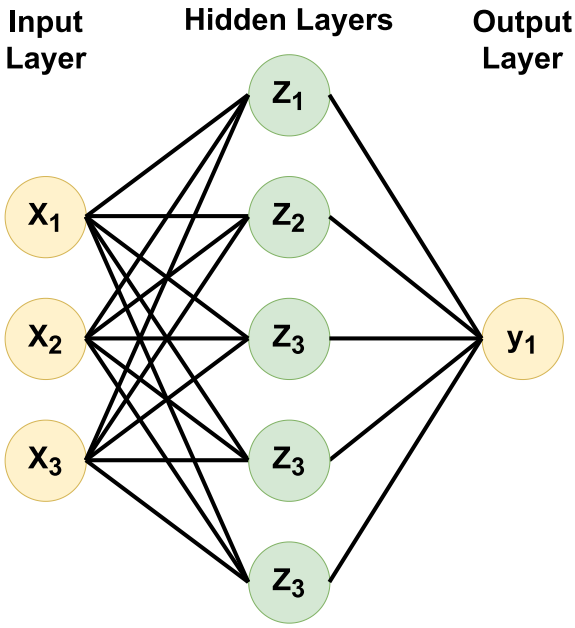


Figure 3.5: Architecture of a NN. The input layer is the first layer of the network. It receives the input data $\mathbf{x}$. The output layer is the last layer of the network and returns $\mathbf{y}$. Between the input and output layers, there are one or more hidden layers.

through the network, the error is propagated in a backward direction. The weights of the layers are adjusted according to the error [32].

3.2.2 Term Frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency (TF-IDF) provides a numerical representation of a word in a document. Let a corpus of documents be denoted $D = \{d_1, d_2, ..., d_M\}$, M being the total number of documents in the corpus. Let a sequence of terms $w_j \in V$ be denoted a document $d_i = \{w_1, w_2, ...\}$, V being the vocabulary, i.e. set of distinct words [56].

The TF-IDF model considers the frequency $f_{w_j,d}$ of a word w_j in a document d and the frequency of a word in the whole corpus. The frequency $f_{w_j,d}$ is defined in Equation 3.1, w'_j being the number of occurrences of w_j in d.

$$f_{w_j,d} = \frac{w'_j}{\sum_{k \in d} w'_k} \tag{3.1}$$

$$TFIDF(w_j, d, D) = TF(w_j, d) \cdot IDF(w_j, D) \tag{3.2}$$

$$TF(w_j, d) = f_{w_j,d} \tag{3.3}$$

$$IDF(w_j, D) = \log_2 \frac{M}{M_j} \tag{3.4}$$

TF-IDF is calculated using Equation 3.2 from [56]. Each entry of a TF-IDF embedding vector represents the TF-IDF value of a word in a document. Hence, the embedding vector is of the same length as the vocabulary of the corpus. The Term Frequency (TF) is determined utilizing Equation 3.3, whereas the Inverse Document Frequency (IDF) is computed by Equation 3.4, M_j being the number of documents the term w_j appears in.

IDF measures the importance of a term w_j in the corpus of documents D under the assumption that a term's importance to the data corpus is inversely proportional to its occurrence frequency [77]. In other words: Terms which appear in many documents are not as important and thus, weighted less than document-specific terms. The calculation of TF and IDF is visualized exemplary in Figure 3.6.

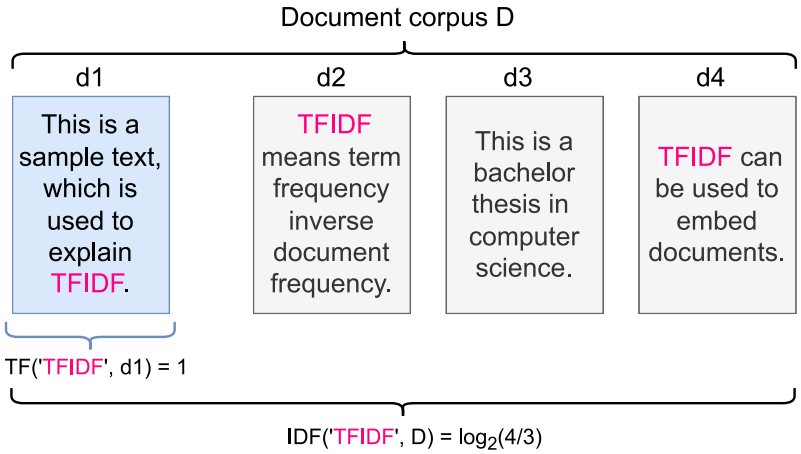


Figure 3.6: Exemplary calculation of TF and IDF for a document corpus D: TF only considers the documents of interest while IDF incorporates the importance of the word with respect to D.

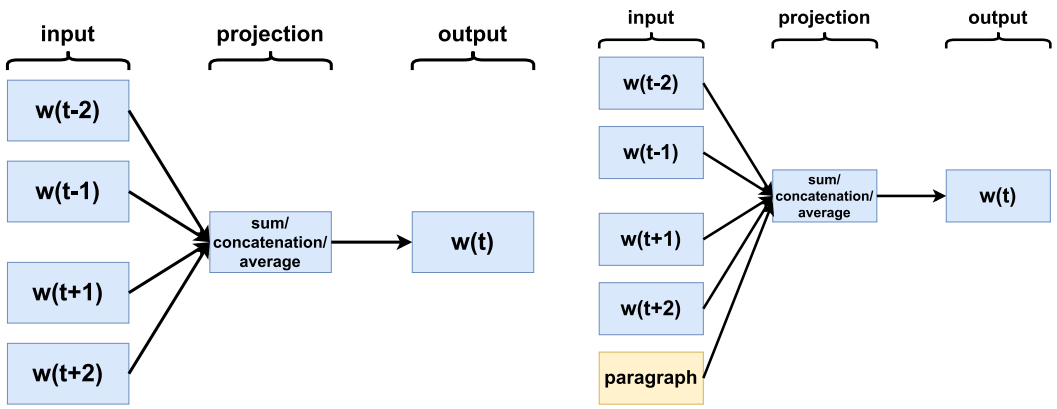
TF-IDF has several drawbacks [56, 77]:

- TF-IDF does not consider semantic similarities between words.
- TF-IDF does not take into account the order of words in a document.
- TF-IDF often produces high dimensional representations which have to be postprocessed to reduce their dimensionality, e.g., by using Principal Component Analysis (PCA).

3.2.3 Document to Vector

Another term used for Document to Vector (Doc2Vec) is *Paragraph Vector* [56, 43]. Doc2Vec addresses TF-IDF's drawbacks by encoding texts as $N-$dimensional vectors learnt using the words' context [56]. $N \in \mathbb{N}$ can be chosen arbitrarily. It preserves semantic similarities between words and encodes linguistic regularities and patterns [45]. The model handles inputs of different lengths, i.e. inputs can be sentences, paragraphs or documents.

Doc2Vec is an adaption of the Word to Vector (Word2Vec) model, which maps words into a VSM [56]. Both approaches assume that words appearing in similar contexts are



(a) CBOW architecture cf. [44]. (b) PVDM architecture cf. [43].

Figure 3.7: Both approaches predict the centre word **w(t)** using the context. PVDM is an adaption of CBOW to work on a set of documents or paragraphs instead of words.

semantically similar. Hence, words which often appear in the same context produce similar embeddings.

The Doc2Vec embedding is obtained using a shallow NN, i.e. the NN has only one hidden layer. The embeddings are created by the hidden layer. There are two Doc2Vec approaches to designing the architecture of the NN:

- Paragraph Vector Distributed Memory (PVDM):
  Predicts a word given a context [43, 44].
- Distributed Bag of Words (PV-DBOW):
  Predicts the context given a word [38, 45, 43].

The PVDM algorithm considers words within a sliding window and their document the context of a centre word [43]. The document vector is added to incorporate the document's topic and thus, acts like a memory [43, 4]. PVDM encodes the context words into vectors via the Word2Vec Continuous-Bag-of-Words (CBOW) model [55]. Each document is mapped to a vector using an additional document-to-vector matrix. The vectors can be concatenated, averaged or summed up [43]. The resulting vector is the prediction of the central word. The CBOW and the PVDM approach are displayed in Figure 3.7.

3.2.4 Universal Sentence Encoder

Cer et al. have published their Universal Sentence Encoder (USE) model on TensorFlow Hub. They propose two architectures, one based on a Transformer and one based on a Deep Averaging Network (DAN) [9]. Both models' input is a lowercase tokenized string. Their output is a 512-dimensional vector.

The transformer model is more accurate and more complex than the DAN model [9]. The transformer's (self) attention is used to compute context-aware word embeddings, which consider both the word order and their semantic identity. Since a sequence of

word embeddings of a sentence produces embeddings of different dimensions, the approach postprocesses the word embeddings. A sentence vector is obtained by computing the element-wise sum of the word embeddings and normalizing the result by dividing by the square root of the sentence length.

The DAN model receives real-valued embeddings of words and bi-grams as input. A bi-gram is a tupel of two subsequent words in a text [8], for instance, *(red, wine), (wine, tastes), (tastes, good)*. The embeddings can be obtained from the text strings using models such as the Bag of Words (BoW) model [21]. They are averaged and subsequently passed to a feedforward DNN [9]. The architecture of the DAN model is depicted in Figure 3.8.

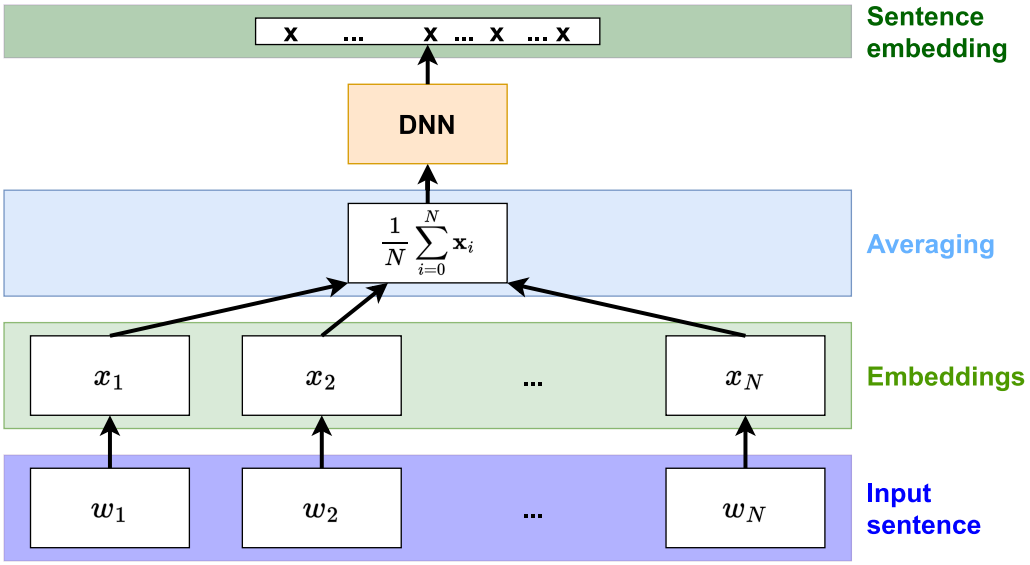


Figure 3.8: Architecture of the DAN model used for USE based on the textual description from [14]. The input words and bi-grams $(w_1, w_2, ..., w_N)$ are embedded. The embeddings are averaged and subsequently passed to a feedforward DNN, which produces a 512-dimensional sentence embedding.

The models are trained on both unsupervised training data, e.g., Wikipedia, and a supervised training dataset, i.e. Stanford Natural Language Inference (SNLI) [9, 58]. The unsupervised training task is to predict the context given an input, i.e. Skip-Gram like tasks. The supervised training task is classification [9].

3.2.5 InferSent

InferSent is a sentence embedding method trained in a supervised manner on the SNLI dataset [14, 58]. The trained model is transferable to other tasks. Conneau et al. compare multiple architectures in their work. The Bi-directional Long Short-Term Memory (BiLSTM) architecture with max pooling which was found to be the best option for the sentence encoder is depicted in Figure 3.9 [14].

A Long Short-Term Memory (LSTM) is a Recurrent Neural Network (RNN) that is capable of learning long-term dependencies. RNNs have closed loops, i.e. feedback connections

between the nodes [59]. In other words, a LSTM is able to remember information as a so-called *state*. Certain LSTM mechanisms control whether the current state is deleted, whether new data is saved and to what degree the current state contributes to the current input processed in the node. Hence, LSTM nodes are not only influenced by former outputs but also by their state. Since the LSTM computes different numbers of hidden vectors h_t depending on the length of a sentence, a max pooling layer is applied to the hidden vectors which selects the maximum value for a patch of the hidden vectors.

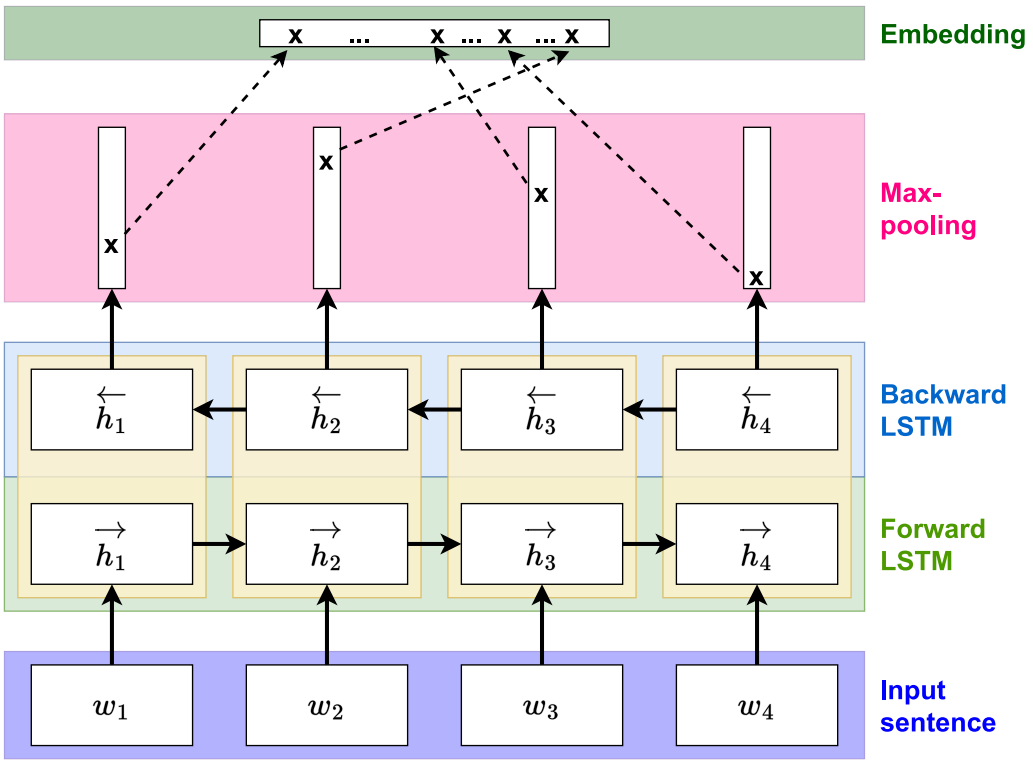


Figure 3.9: Architecture of the BiLSTM model with max pooling used for InferSent cf. [14]. The input sentence $(w_1, w_2, ..., w_T)$ is read from both directions by a forward and a backward LSTM producing $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ respectively. After concatenating $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ to h_t, max pooling is applied. The output is a fixed-sized embedding.

According to Reimers and Gurevych, InferSent consists of a single BiLSTM layer [58]. Given a sentence $(w_1, w_2, ..., w_T)$ of T words, the BiLSTM architecture computes the hidden representations h_t for each word w_t. The hidden representation h_t is the concatenation of the forward and backward hidden vectors $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$. $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ are produced by a forward and backward LSTM respectively. Hence, the sentence is read from both directions and thus, considers past and future context.

3.2.6 Sentence-BERT

Sentence-BERT (SBERT) is an enhancement of Bidirectional Encoder Representations from Transformers (BERT). The applicability of BERT is limited because it does not produce independent embeddings for single sentences [58]. Moreover, Reimers and Gurevych

found that common similarity measurements, for instance, the ones discussed in Section 3.3, do not perform well on sentence embeddings produced by BERT.

BERT is a pre-trained transformer network which predicts a target value based on two input sentences for sentence classification or sentence-pair regression tasks [58]. The BERT base model applies multi-head attention over 12 transformer layers, whereas the large model applies multi-head attention over 24 transformer layers. The attention mechanism enables access to all hidden states as opposed to only the last hidden state [34]. It derives its output vector as a dynamic weighted sum of the hidden states. The final label is derived from a regression function, which receives the output of the 12^{th} or 24^{th} layer, respectively.

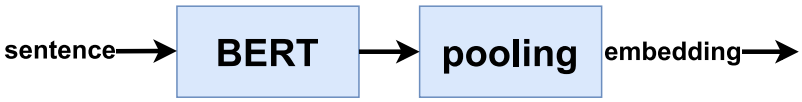


Figure 3.10: Architecture of SBERT cf. [58]. BERT is extended by a pooling layer. The input is a sentence and the output is a fixed-sized embedding.

SBERT provides fixed-sized embeddings for single sentences [58]. It differs from BERT in terms of architecture, since it adds a pooling layer after the BERT model. Reimers and Gurevych compare different pooling strategies, such as using the output of the `CLS` (i.e. first) token, mean pooling and max pooling. The architecture of a single SBERT network is depicted in Figure 3.10. In order to work with multiple input sentences at the same time, siamese and triplet network architectures, i.e. multiple BERT networks with tied weights, are constructed. To perform classification or inference tasks, layers are added on top of the SBERT network. SBERT is trained on the SNLI dataset [58, 28].

According to Reimers and Gurevych, SBERT outperforms InferSent and USE on Semantic Textual Similarity tasks and on SentEval, which is an evaluation toolkit for sentence embeddings [58].

3.3 Similarity measurement

Embeddings not only facilitate human interpretability of relationships between texts, but they also enable the use of metrics, i.e. similarity measures, to quantify the similarity between texts [63, 37].

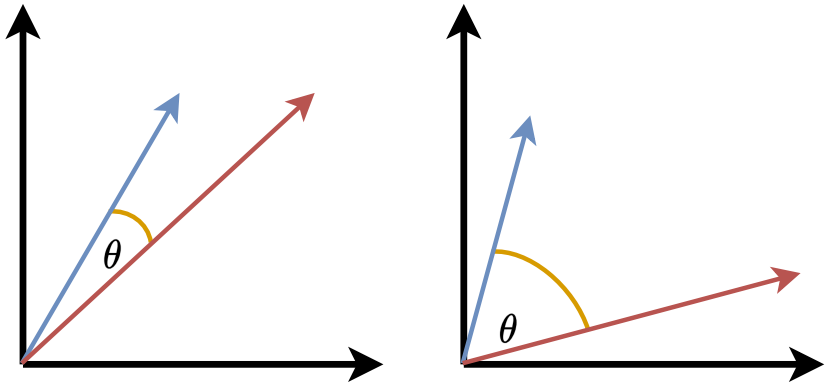
There are several similarity measures, such as the dot product quantifying the number of shared tokens of two texts, the (soft) cosine similarity [62, 11], which is the normalized dot product and calculates the angle between two vectors, and many more [63, 37, 58]. The following section outlines a selection of similarity measures.

3.3.1 Euclidian distance

The *euclidian distance* is a distance measure. In order to measure the distance between two vectors in a N-dimensional space, the root of the sum of squared distances between the respective values of every dimension is calculated. The Euclidean (L2) norm between two vectors a, b is defined in Equation 3.5 [37]. The distance is zero if the vectors are identical, i.e. $a = b$. The more a and b differ, the greater is the distance $d_E(a, b)$ between them.

$$d_E(a, b) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2} \tag{3.5}$$

3.3.2 Cosine Similarity



(a) Similar vectors a, b. (b) Dissimilar vectors a, b.

Figure 3.11: Cosine Similarity between two vectors considers the angle between them.

The similarity between two texts is measured by the cosine of the angle between their respective real-valued vectors. The cosine similarity is defined in Equation 3.6 [62]. For positive vectors, for instance, produced by TF-IDF, it is a value between 0 and 1. If the angle is close to zero degrees, the cosine similarity is close to 1 and the vectors are similar. If the angle is close to 90 degrees, the cosine similarity is close to 0 and the vectors are dissimilar. Both a similar and a dissimilar pair of vectors are depicted in Figure 3.11.

$$cosine(a, b) = \frac{a \cdot b}{\|a\| \times \|b\|} = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{3.6}$$

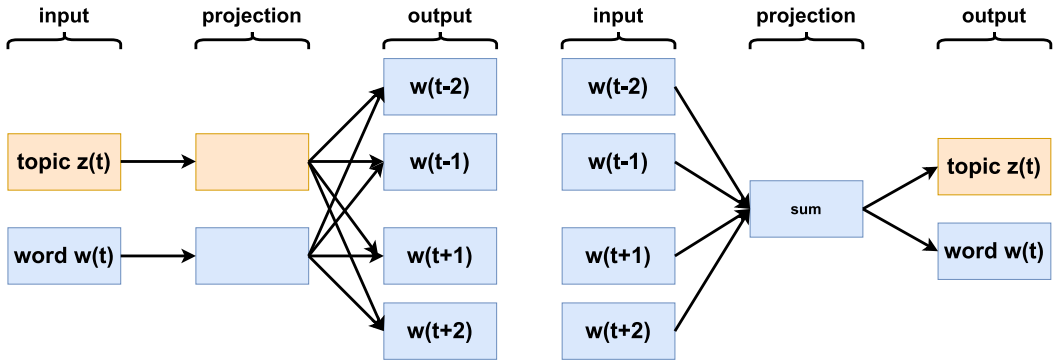
The formula from Equation 3.6 assumes that the vectors, which span the VSM are orthogonal and thus, completely independent. However, in practical applications, the index terms which span the VSM are often semantically dependent.

3.4 Topic analysis

Since more and more textual data emerges, methods to analyze and extract information from texts become more important. One of these methods is topic analysis. A topic can be defined as a cluster of words that occur frequently or are semantically similar to each other. A document can be represented by one or more topics [3].

3.4.1 Topic to Vector

The approach Topic to Vector (Top2Vec) addresses several problems of state-of-the-art topic analysis approaches, such as Latent Dirichlet Allocation (LDA) [4]. Top2Vec does not require the user to specify the number of topics k, i.e. it does not discretize the topic space into k topics, and it does not require stop word removal or lemmatization. It considers the semantic meaning of words. Top2Vec only associates one topic with a document.



(a) Skip-gram architecture cf. [52]. (b) CBOW architecture cf. [52].

Figure 3.12: Both learning architectures of Top2Vec. $w(t-2), w(t-1), w(t+1), w(t+2)$ are the context words of the centre word $w(t)$ of topic $z(t)$.

Top2Vec is based on Word2Vec and Doc2Vec. The documents are embedded using the Doc2Vec model PV-DBOW. The two learning architectures CBOW and Skip-gram from Word2Vec are adapted to train the model as depicted in Figure 3.12 [52]. The Skip-Gram learning task is to predict the context a word came from [4, 52]. Top2Vec embeds words, documents and topics in the same feature space. The similarity between embeddings can be measured using the cosine similarity function [52].

Angelov regards each point in the VSM as a topic, described by its nearest words. The author states that topics are continuous and can be described by different sets of words [4]. Hence, topic analysis can be defined as the task of finding sets of informative words that describe the topic of a document. Documents in dense areas of the topic space are considered to be about the same topic. The density-based clustering algorithm Hierarchical DBSCAN (HDBSCAN) is used to find these dense areas. Since HDBSCAN has difficulties finding dense clusters in high-dimensional data, the dimensionality reduction method Uniform Manifold Approximation and Projection (UMAP) is applied [4]. The steps of the topic analysis procedure Top2Vec are depicted in Figure 3.13.

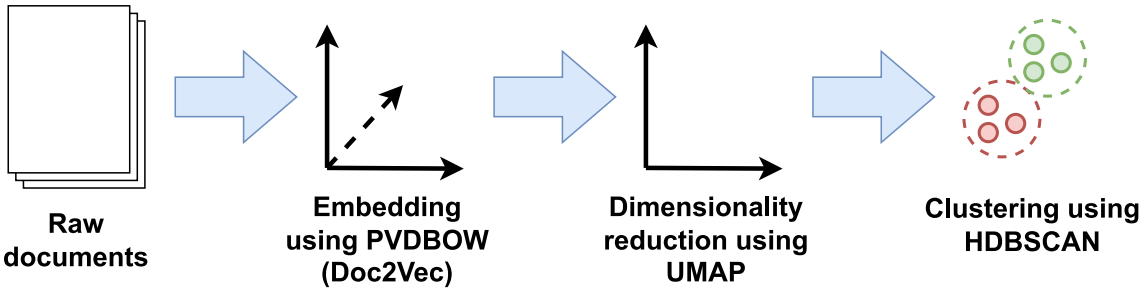


Figure 3.13: Procedure of topic analysis using Top2Vec.

A topic vector is denoted as the centroid or average of the document vectors that belong to a certain topic. The number of topics is derived from the number of dense areas. It is possible to merge topics to hierarchically reduce the number of topics to any number smaller than the number of topics initially found.

3.4.2 Word clouds

A word cloud is a technique to visualize the most predominant words in a text [36]. The size of a word correlates to its frequency or importance in the text. However, a word does not have to be meaningful to appear large. A word cloud does not provide information about the meaning or context of words and thus, one has to be careful when interpreting the results.

3.5 Compression of data

According to Radu et al., a decomposition of data preserves the inner structure in inherent clusters. When data analysis techniques are applied to reasonably low-dimensional data, the results usually improve. Moreover, compressed data is less memory-consuming and often less difficult to interpret by humans since there are more methods to visualize low-dimensional data. In the following, two approaches to reduce the dimensionality of data are presented.

3.5.1 Autoencoder

The idea of this approach is to find a meaningful low-dimensional version of the input. The high-dimensional data is encoded into a low-dimensional representation using the encoder of an undercomplete Autoencoder (AE) [46]. Hence, the output of the latent space corresponds to the input's embedding. The low-dimensional representation can be decoded into an approximation of the high-dimensional original using the decoder of the AE.

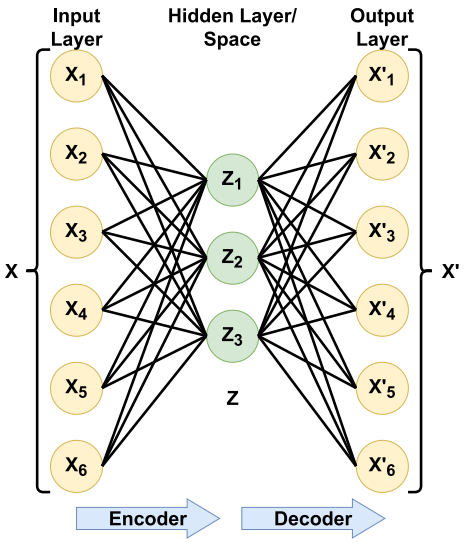


Figure 3.14: Structure of an AE cf. [46]. The six-dimensional input is encoded into a three-dimensional representation. This encoding is decoded into a six-dimensional approximation of the original input.

An undercomplete AE is a feed-forward NN, which consists of an encoder and a decoder. NNs are discussed in Subsection 3.2.1. It learns efficient (non-correlated) encodings of the input data [46]. It is *undercomplete* because the dimensionality of the hidden layer, or so-called hidden space, is lower than the dimensionality of the input layer [31]. The input and output layers have the same dimensionality.

The network employs backpropagation to update the parameters of the network during training. The AE's goal is to approximate the identity function $f_\theta(X) = X$ (trivial solution eliminated) for input X and function parameters to be learned θ [31].

3.5.2 Eigenfaces

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 [67]. The goal of this approach is to represent images using a smaller set of image features, i.e. compression to a lower-dimensional feature space, such that it is possible to distinguish between the images [67, 71]. These features do not necessarily correspond to human facial features [67]. Similar pictures, i.e. of the same person, should lie on a manifold in the lower-dimensional feature space [65]. The decomposition of input images not only reduces the complexity but also facilitates modeling probability density of a face image [65].

The greyscale input images are two-dimensional arrays of numbers: $\mathbf{x} = \{x_i | i \in \mathbf{S}\}$, $\mathbf{S}$ being a square lattice [76, 67]. 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 [76]. Some authors remove the background to omit values outside the face area [67].

Turk and Pentland stress that the data should be normalized, i.e. centered, as computed in Equation 3.7. Φ_k is the difference of the k-th training image and the average image calculated using Equation 3.8, N being the number of training images.

$$\Phi_k = \mathbf{x}_k - \psi \tag{3.7}$$

$$\psi = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k \tag{3.8}$$

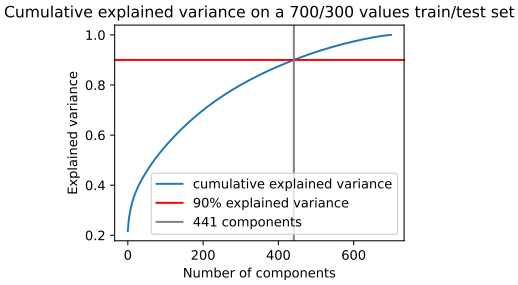
$$\mathbf{x} = \sum_{i=1}^{n}\hat{x}_i\mathbf{e}_i \tag{3.9}$$

The next step is to find an alternative lower-dimensional representation of the images, which preserves most of the information of the original image. In mathematical terms, this decomposition can be expressed using the formula in Equation 3.9, $\mathbf{e}$ being an orthogonal basis [76]. If all basis vectors are used, the original image can be reconstructed using a linear combination of the basis vectors [67, 16]. The number of basis vectors is limited by the minimum of the training set size N [67] and the number of pixels n [16]. In order to compress the input from a n to a m-dimensional space, given $m \ll n$, only the first m basis vectors are used. The parameter m and the basis $\mathbf{e}$ is chosen such that $\hat{x}_i$ is small for $i \geq m$ [76]. 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 used to transform the compressed image back to the original space. The weights denote the position of the projection of the face images in the feature space or so-called face space spanned by the first m basis vectors [67].

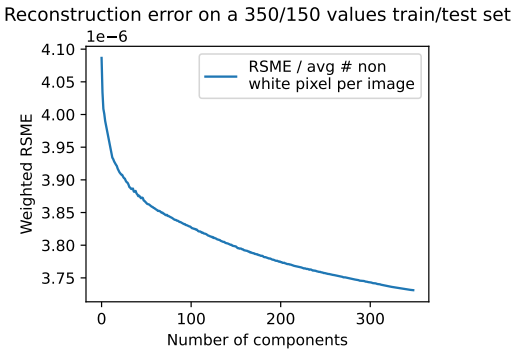
In the context of Eigenfaces, one basis used for decomposition is the Karhonen-Loéve (KL) basis, i.e. PCA [76, 67]. According to Zhang et al., the KL representation is optimal in the sense that it minimizes the Root Mean Square Error (RMSE) between the original image and the compressed image calculated using $m < n$ orthogonal vectors. The KL basis consists of the eigenvectors of covariance matrix $\mathbf{C} = E\left[\mathbf{x}\mathbf{x}^T\right]$ of the input images $\mathbf{x}$ [76]. Since these eigenvectors can have facial features, they are called *Eigenfaces*. There are two approaches in the literature to determine the number of Eigenfaces m used to compress the input images:

(a) The cumulative explained variance of the first $i \leq n$ eigenvectors (sorted by eigenvalues λ_i) is calculated [76, 16, 64]. 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 a chosen threshold. Sudiana et al. use a threshold of 90%. A plot displaying the cumulative explained variance and a threshold of 90% is shown in Figure 3.15 (a).

(b) The number of Eigenfaces m is chosen using the reconstruction error-complexity trade-off. The reconstruction error, i.e. the RMSE of the original image x and the inverse transformed image x', is calculated in Equation 3.10 for different values of m. The "elbow" point marks the point where the reconstruction error decreases only slightly for increasing m and thus, is an indicator for the optimal m. A visualization of this approach is shown in Figure 3.15 (b).



(a) The cumulative explained variance of the first $i \leq n$ eigenvectors (sorted by eigenvalues λ_i).



(b) The reconstruction error RMSE calculated for different values of m. The reconstruction error increases less rapidly after 10 to 20 components.

Figure 3.15: Two approaches to determine the number of Eigenfaces m used to compress the input images.

$$\text{RSME} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - x_i')^2}{N}} \tag{3.10}$$

$$\mathbf{C} \simeq \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k\mathbf{x}_k^T = \frac{1}{N}\mathbf{X}\mathbf{X}^T \tag{3.11}$$

$$\mathbf{e}_i = \frac{1}{\sqrt{\lambda_i}}\mathbf{X}\mathbf{v}_i \tag{3.12}$$

In order to reduce calculation complexity, C is approximated. Zhang et al. propose the approximation displayed in Equation 3.11, with $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^n$ [76].

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. According to Zhang et al., the eigenvectors of $\mathbf{X}\mathbf{X}^T$ can be calculated by using the eigenvectors of $\mathbf{X}^T\mathbf{X}$. The eigenvectors $\mathbf{e}_i \in \mathbb{R}^n$ of $\mathbf{X}\mathbf{X}^T$ can be derived from the eigenvectors $\mathbf{v}_i \in \mathbb{R}^N$ of $\mathbf{X}^T\mathbf{X}$ using Equation 3.12 as discussed in more detail in [76]. According to Anowar et al., the problem is reduced to a N by N matrix, which is computationally less expensive to solve assuming $N \ll n$. The eigenvectors can be calculated using Singular Value Decomposition (SVD) [76]. SVD is a method, which decomposes a matrix into the so-called left singular vector, the diagonal matrix and the right singular vector [6].

In the literature, face images are classified by comparing their position in the face space with those of already known faces [67]. According to [67], 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 and thus, the distance function used to make classifications is no longer a reliable measure.

3.6 Clustering

Clustering is used in a variety of domains to group data into meaningful subclasses [54, 17, 35]. According to Patwary et al. and Radu et al., common domains include anomaly detection, noise filtering, document clustering and image segmentation. The objective is to find clusters, which have a low inter-class similarity and a high intra-class similarity [54]. 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 [35].

There are multiple clustering techniques, which can be divided into four categories [2]:

- **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 [17] and Ordering Points To Identify Clustering Structure (OPTICS) [54].
- **Partitional based clustering**: Algorithms, that partition the data into k clusters, 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 [35]. 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 [17].

Multiple approaches listed below use the term *ε-neighbourhood*, which is defined as the set of all objects within a certain distance ε of a given object [54]. In other words: $N_\varepsilon(x) = \{y \in X | dist(x, y) \leq \varepsilon, y \neq x\}$, ε being the so-called generating distance.

3.6.1 DBSCAN

The clusters identified by DBSCAN have a high density and are separated by low-density regions [35]. In order to create clusters of minimum size and density, DBSCAN distinguishes between three types of objects [35]:

- **Core objects**: An object x with at least $minPts \in \mathbb{N}$ objects in its ε-neighbourhood $N_\varepsilon(x)$, i.e. $|N_\varepsilon(x)| \geq minPts$ is true [54].
- **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 [35]. 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 3.16 [35].

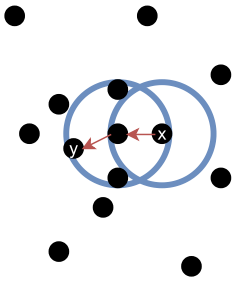


Figure 3.16: Density reachability cf. [5]. The object $y \in X$ is density reachable from $x \in X$, since it exists a chain of directly density reachable objects between x and y.

The objects $x \in X$ 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 [35]. Density connectivity is visualized in Figure 3.17.

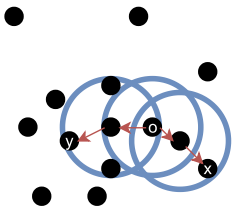


Figure 3.17: Density connectivity cf. [5]. The objects x and y are density connected since there is an object o, from which both x and y are density reachable.

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 [35]. Groups of connected core points form a cluster. In the end, every border point is assigned to a cluster. The non-core point cluster assigning is non-deterministic [54]. This algorithm creates clusters as a maximal set of density-connected points [35].

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 [35]. 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 [54]. According to Radu et al., DBSCAN is affected by the curse of dimensionality. Since DBSCAN relies on nearest neighbour queries and these become less meaningful in high dimensions, i.e. distances become difficult to interpret, the quality and accuracy of the results decline with increasing dimensionality [56]. Radu et al. found that their DBSCAN model assigns most objects noise when the dimensionality is sufficiently large.

3.6.2 OPTICS

OPTICS does not return an explicit clustering, but rather a density-based clustering structure of the data, which is equivalent to repetitive clustering for a broad range of parameters [5]. Ankerst et al. claim that real-world datasets cannot be described by a single global density, since they often consist of different local densities, as displayed in Figure 3.18.

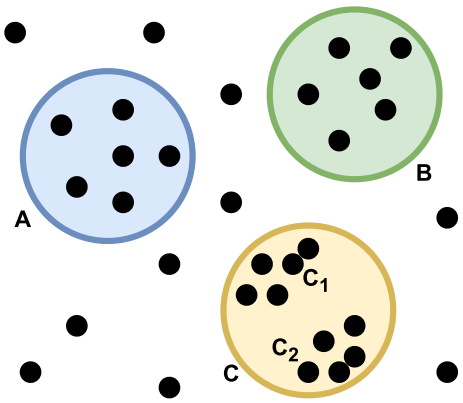


Figure 3.18: Clusters with different densities cf. [5]. Since C_1 and C_2 have different densities than A and B, a clustering algorithm using one global density parameter would detect the clusters A, B and C, rather than A, B, C_1 and C_2 .

Opposed to DBSCAN, OPTICS is able to detect clusters of varying densities [17]. OPTICS produces an order of the elements according to the distance to the already added elements [17, 54]: The first element added to the order list is arbitrary. 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 the elements already in the order list. Hence, clusters with higher density, i.e. lower ε, are added first (prioritized) [35, 5]. 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 [54].

$$RD(y) = \begin{cases} \text{NULL} & \text{if } |N_\varepsilon(x)| < minPts \\ max(core\_dist(x), dist(x,y)) & \text{otherwise} \end{cases} \tag{3.13}$$

OPTICS saves the reachability distance $RD(y)$, as calculated in Equation 3.13 from [54], with core distance *core_dist* being the minimal distance ε^{min} such that $|N_{\varepsilon^{min}}(x)| \geq minPts$ (i.e. the distance to the $minPts^{th}$ point in N_ε) or NULL else, of each element y to its predecessor x in the order list and thus, a representation of the density necessary to keep two consecutive objects x and y in the same cluster [54]. 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 using the information saved by OPTICS [54, 5]. 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 [5]. Otherwise, the element is a noise point. 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 $\varepsilon < RD(y)$ [54]. The relationship between ε, cluster density and nested density-based clusters is displayed in Figure 3.19.

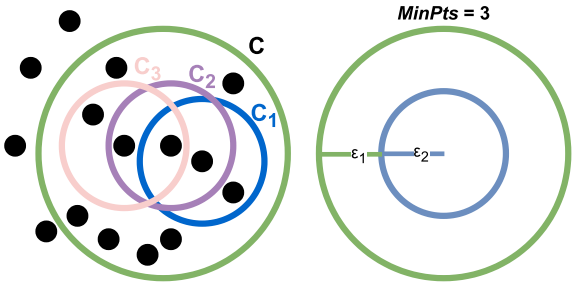


Figure 3.19: The relationship between ε, cluster density and nested density-based clusters cf. [5]. For a constant $minPts$, clusters with higher density such as C_1, C_2 and C_3, i.e. a low ε_2 value, are completely contained in lower density clusters such as C given $\varepsilon_1 > \varepsilon_2$. 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 $0 \leq \varepsilon_i \leq \varepsilon$.

This procedure enables the extraction of clusters for arbitrary $0 \leq \varepsilon_i \leq \varepsilon$ [35, 5]. According to Patwary et al.'s work, even though the clustering algorithm is expensive, the extraction only needs linear time. Ankerst et al. claim that 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.

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 [5]. 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. Ankerst et al. present a heuristic for choosing ε based on the expected k-nearest neighbour distance [5].

High values for $minPts$ smoothen the reachability curve, even though the overall shape stays roughly the same [5]. According to Ankerst et al., the optimal value for $minPts$ is between 10 and 20.

3.7 Software frameworks

The embeddings obtained by the methods described in Section 3.2 are stored in a Elasticsearch database. It is described in Subsection 3.7.1. The different methods explored in this work ought to be presented in a web application. This application should be used to compare the methods and to visualize the results. Subsection 3.7.2 and Subsection 3.7.3 respectively describe Flask and Angular, which are the frameworks and components used to implement the web interface.

3.7.1 Elasticsearch database

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 [70]. 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 [70, 74]. 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 short refreshing periods which make performed operations on the data quickly available for search.

Elasticsearch is a document store, which stores schemaless key-value pairs called documents [22]. The 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 a single running instance of Elasticsearch [74]. An index is divided into one or more shards, which can be stored on different servers and enable parallelization. Replicas are copies of shards, which create redundancy and thus, ensure availability.

The documents are saved in a JavaScript Object Notation (JSON) format [70]. 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 [74].

By specifying the unique `_id` of a document and the database `index`, it is possible to retrieve a specific document from the database using a `GET` endpoint of the HTTP API. The parameters `_source_excludes` or `_source_includes` can be used to define the structure of the response [19].

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 [19]. The default preprocessing steps of the so-called default analyzer include tokenization and lowercasing. Omitting stop words is disabled by default, but custom stop words can be provided by the

user or the English stop word list can be used. It is possible to create custom tokenizers, which split the query value into tokens of a certain maximum length.

Another useful feature of Elasticsearch is the multi-term synonym expansion where the user query is expanded to include synonyms of the query terms [19]. The maximum number of expansion terms is set to 50 by default but can be configured by the user. By default, the multi-term synonym expansion option is enabled.

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 contain the word *Bahamas*. By default, this option is not enabled but can be enabled and configured individually by the user [19].

Another search option of Elasticsearch is the k-Nearest Neighbour (kNN) search on real-valued vectors. The return value of a kNN search is the `k` nearest neighbours to the query vector in terms of a certain distance function [42]. In order to perform 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 query value must have the same dimension as the vectors stored in the database. A kNN search either returns the exact brute-force nearest neighbours or an approximation of the nearest neighbours calculated by the Hierarchical Navigable Small World (HNSW) algorithm [42, 19]. HNSW is a graph-based algorithm [42].

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]. Elasticsearch's kNN implementation not only allows literal matching on search terms but also semantic search incorporating Kibana's text embedding ingest pipeline on search terms [19].

3.7.2 Flask

Flask is open source and written in Python by Armin Ronancher in 2004 [7, 50]. According to Copperwaite and Leifer and Mufid et al., Flask is one of the most popular Python web frameworks. It provides powerful libraries for core functionality such as routing, templating, and HTTP request parsing [15]. It can be extended with additional plugins without affecting the internal structure of the existing system [7].

Flask uses the Jinja Template Engine for template files including HTML pages, whereas static files such as Cascading Style Sheet (CSS) files are handled using the Werkzeug WSGI toolkit [7]. According to Aslam et al., Jinja is modeled after the Django template system. Werkzeug implements, for instance, requests and response objects [50].

All requests received from clients are passed to an instance of the Flask application [25]. Hence, the first step is to create an instance of the Flask class as shown in Listing 3.1.

```
app = Flask(__name__)
```

Listing 3.1: Initialization of a Flask application instance.

Clients send requests to the web server, which passes them to the Flask application instance. The queries are then routed to the corresponding functions. Routing is the process of mapping Uniform Resource Locator (URL) paths to functions [25]. To define a route, the `route` decorator is used as displayed in Listing 3.2.

```
@api.route('/documents/<id>', endpoint='document')
class Document(Resource):
    def get(self, id):
        client = Elasticsearch(CLIENT_ADDR)
        return query_database.get_doc_meta_data(client, doc_id=id)
```

Listing 3.2: Exemplary definition of a function to display routing with Flask. The `route` decorator is used to define the URL path.

URLs can contain dynamic components, which are enclosed in `<>` angle brackets. The values of these components are passed to the function as arguments [25]. By default, dynamic components are of type `string`. However, other types including `int` and `float` are supported.

An endpoint is a class with certain methods, which can be accessed using HTTP requests. Every endpoint can have multiple decorators, including `GET`, `POST`, `PUT` and `DELETE` [22]. The `GET` method is used to retrieve data from the server, whereas the other methods are used to either insert, update or delete data.

3.7.3 Angular

Angular is a framework for building web applications. It uses Node.js and TypeScript. Usually, the source code is structured into different modules, including components and services. Components are used to define the appearance of the application, while services contain the logic of the application and communicate with the backend.

Angular applications are created using the `ng new <name>` command line interface [61]. This command creates a skeleton, which can be customized to meet the needs of the application.

4 Own approach

In this thesis, a tool is developed that offers text queries, detailed document inspection and queries for semantically or visually similar documents to the user. This chapter describes how the theoretical basics from Chapter 3 interplay and how they are used to construct this tool. Section 4.1 outlines the steps carried out before the application is operative, Section 4.2 covers the resulting application and Section 4.3 discusses the dilemma faced when balancing memory usage and query time. Specific parameter choices are explained in Chapter 5.

4.1 Offline Processing

The tasks carried out before the application is operative are outlined in this section. They are considered to be offline preprocessing steps. A process works in an offline fashion if the process requires access to the whole data at once [29]. This section outlines implementation details of the way the data is derived, the database storing the data and the baseline topic analysis approach compared to this work's application.

4.1.1 Database

First, the content of the Elasticsearch database is described, then, the initialization, insertion and updating process of filling the database are explained and finally, the process of querying is outlined.

Content of the database

In this work, the database is filled once with data from the Bahamas leak. The data is a large unstructured corpus of PDF files. Since leak data does not change over time it is not necessary to update the database. After the initialization of the database, it is used for queries. Therefore, the workflow of processing the text corpus is carried out completely offline and in advance.

The index *Bahamas* stores different embeddings of the information derived from the text layer and metadata of the documents. As depicted in Figure 4.1, not only textual information is stored in the database, but also information about the appearance of the first page of the PDF. The structure of the index is presented in Table 4.1. The visual information is stored in the fields `pca_image`, `pca_optics_cluster` and `argmax_pca_cluster`.

Table 4.1: Fields of the Elasticsearch database index *Bahamas*.

Field name	Field description
_id	Unique identifier of document `i`. The identifier is generated by the sha256 hash algorithm from hashlib using the PDF file as input.
doc2vec	55 dimensional Doc2Vec embedding of `i`.
sim_docs_tfidf	TF-IDF embedding enhanced by an all-zero flag of `i`. The all-zero flag is one if the TF-IDF embedding consists of only zeros, zero else. If the embedding's dimensionality is greater than 2048, the encoder of a trained AE is used to compress the embedding.
google_univ_sent_encoding	512 dimensional USE embedding of `i`.
huggingface_sent_transformer	384 dimensional SBERT embedding of `i`.
inferSent_AE	InferSent embedding of `i`. Since the pretrained InferSent model embedding's dimension is 4096, the encoder of a trained AE has to reduce the dimension to 2048.
pca_image	13-dimensional PCA version of first page image of `i`.
pca_optics_cluster	Cluster of `i` identified by OPTICS on PCA version of image.
argmax_pca_cluster	Number of maximum PCA component as cluster of `i`.
text	Text of `i`.
path	Path to `i`.

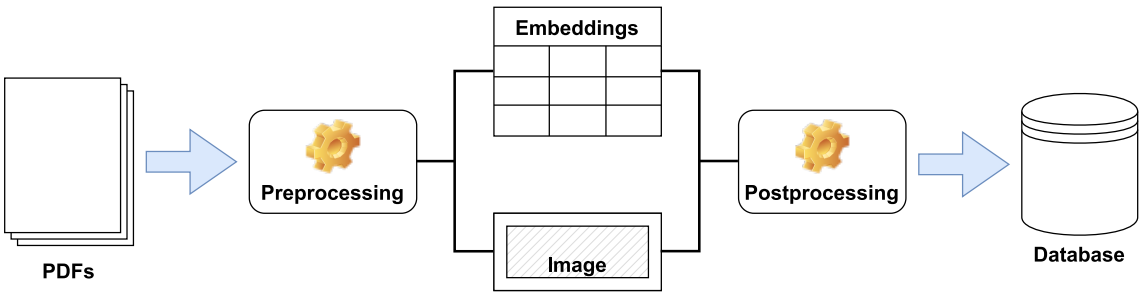


Figure 4.1: PDFs to Database. First, the data is preprocessed: The first page of a PDF file is converted to an image and the complete text is extracted. The images are stored in the database as well as the text and different embeddings of the text. Some values, such as the image or the InferSent embedding, have to be compressed to become a vector of at most 2048 dimensions.

Initialization, insertion and updating

To facilitate working with and running the code, the initialization of the database is split into multiple steps. As depicted in Figure 4.2, first the database is initialized by defining the index name and the mappings, i.e. the field names, types and sizes. This step is carried out using the method `create`.

Figure 4.2: Procedure of initialization and filling of the database.

Afterwards, the documents are created using the method `create`. The initial creation of a document only defines the fields `id`, `text` and `path`.

The embeddings are added to the documents in a third step. To increase the efficiency of this step, data parallelism, i.e. parallelizing the execution of a method across multiple input values, is applied. In this work, a set of paths to documents is split among multiple processes. First, the absolute paths of all documents are saved in a list. This list is partitioned into `num_cpus` many lists `sub_lists` of similar size. Each process works on a sublist. The `Pool` object from the multiprocessing module is used for data parallelism. The steps carried out are displayed in Listing 4.1. The embeddings are subsequentially inserted into the database for each sublist.

```
1 with Pool(processes=num_cpus) as pool:
2     for model_name in model_names:
3         proc_wrap = wrapper(model_name=model_name, baseDir=src_path)
4         pool.map(proc_wrap, sub_lists)
```

Listing 4.1: Usage of `Pool` for data parallelism. The paths to the documents are partitioned into sublists which are simultaneously inserted into the database. Since the `Pool` object does not work with a `lambda` function, a class `wrapper` is created which provides the same functionality.

The document embeddings are added to the database using the method `update` as displayed in Listing 4.2.

```
1 client.update(index='bahamas', id=id, body={'doc':
2     {MODELS2EMB[model_name]: embedding}})
```

Listing 4.2: Update of a database entry to insert a specific embedding.

Queries

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.

```
results = elastic_search_client.search(
    index='bahamas',
    size=count,
    from_=(page*count),
    query= {'match': {
                'text': {'query':text,
                        'fuzziness': 'AUTO',}
                },
            }, source_includes=SRC_INCLUDES)
```

Listing 4.3: Exemplary query to an Elasticsearch database index. The parameters `size` and `from_` define the number of results to return and the start index of the results. To enable fuzzy search a value for `fuzziness` has to be set.

Moreover, 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 [19]. 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 [19]. An exemplary query, which uses fuzzy search is given in Listing 4.3.

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. In this work, the approximate nearest neighbours search HNSW is used since it is faster and the results are good enough for the purpose of this work. The similarity measure used in this work is the cosine similarity. The other similarity measures provided by Elasticsearch are the `l2_norm` or so-called Euclidian distance and the `dot_product` which is the non-auto-normalized version of the `cosine` option. Since cosine is not defined on vectors with zero magnitude, embeddings that can return all zero vector representations, such as TF-IDF, are enhanced with an all-zero flag before inserting them into the database.

In this work, the only tool from the elastic stack used is Elasticsearch. Without Kibana, the used models are saved on disk as Pickle (PKL) files. Consequently, instead of using the kNN query structure for semantic search on embeddings provided by Elasticsearch and Kibana, the normal kNN search on a field that contains an embedding is used.

4.1.2 Eigendocs

In this work, the Eigenfaces approach from Subsection 3.5.2 is used to compress the images of the first page of the 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

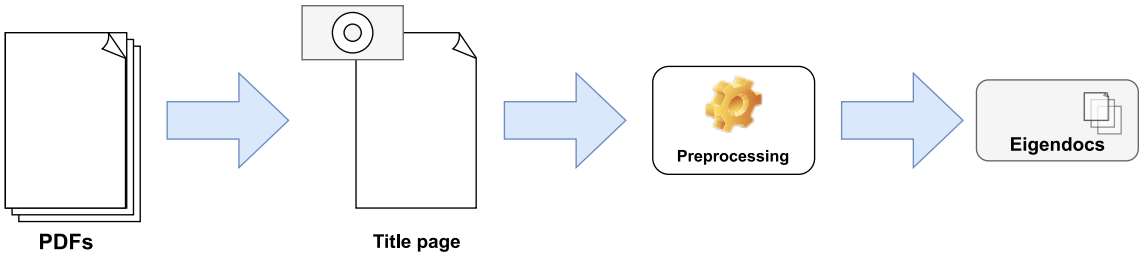


Figure 4.3: From PDFs to Eigendocs. Firstly, the first page of a document is converted to an image. Then, the image is preprocessed: It is placed on a white canvas, to ensure all images have the same dimensions. Moreover, it is converted to greyscale and normalized to values between zero and one. Afterwards, the two-dimensional image is reshaped into a one-dimensional array. Lastly, the image is compressed using Eigendocs.

procedure of the Eigenfaces adaption *Eigendocs* is displayed in Figure 4.3. Different stages of this approach are displayed in Figure 4.4.

Figure 4.4: 10 randomly selected documents from the test set. The number of images in the test set is 561, while the PCA model is fitted to 1680 training images. The original images are displayed in the first row. The second row shows the reconstruction from their compressed version in the fourth row. The third row shows the reconstruction error, i.e. the difference between the reconstructed and the original image. The last row presents the greyscale values of the compressed 13-dimensional image as a line.

The documents are first read from a directory. Subsequently, their first page is converted to an image and saved. The maximum height and width among all images in a corpus of 1000 randomly sampled images are calculated. The selection of 1000 images is used to reduce the run time of the script. The maximum height and width are used to create a white canvas for each image which forms the background. Every image is placed in the upper left corner of the canvas. Hence, assuming the selection of documents used to fit the PCA model is representative, scaling is not necessary and thus, the portion of white pixels on the right and bottom side encodes the sizes of the original image. Therefore, the relative size of images in the corpus is incorporated in the resulting representation of the

input images. However, some images of the test set are bigger than the maximum values of the selected images and as a consequence are scaled.

```
1 def rgb2gray(img):
2     return 0.299*img[:,:,0] + 0.587*img[:,:,1] + 0.114*img[:,:,2]
3 # more code
4 C = np.ones((max_w,max_h))
5 C[:doc.shape[0],:doc.shape[1]] = rgb2gray(doc)
6 documents.append(C.ravel())
```

Listing 4.4: Preprocessing of the input images from Dr. Christian Gruhl. Conversion of RGB pixel values to greyscale according to [41]. The background is a white canvas. The images are converted to one-dimensional greyscale values.

Afterwards, the images are converted to greyscale using line 5 of Listing 4.4. Before returning the image, the two-dimensional image vectors are converted to one-dimensional ones as displayed in line 6 of Listing 4.4. The decomposition is transformed using PCA as displayed in Listing 4.5. The implementation of PCA from sklearn intrinsically normalizes the data as described in Subsection 3.5.2.

```
1 pca = decomposition.PCA(n_components=n_components, whiten=True,
2     svd_solver="randomized")
```

Listing 4.5: Initialization of the PCA instace used to compress the images. Since the Eigenfaces approach uses SVD, the adaption Eigendocs has to be implemented likewise applying a `svd_solver`.

4.1.3 Embeddings

Firstly, the implementation of the AE used to compress high-dimensional embeddings is presented. Then, the models used to encode the textual data are outlined below with regard to implementation details.

Autoencoder

In this work, an AE is used to reduce the dimensionality of the InferSent and the TF-IDF embeddings. Since the InferSent model is pretrained, it is not possible to change the dimensionality of the embedding without a considerably big effort, i.e. retraining the model on a sufficiently large data corpus and reconfiguring the model's parameters. Therefore, it is not feasible to change the dimensionality of the InferSent embedding, but rather add a supplementary layer after the model to produce the final embedding. Similarly, the TF-IDF embedding dimension correlates with the vocabulary size and thus, the size of the data corpus. Further reducing the vocabulary size would decrease the TF-IDF model's

quality. Hence, the idea is to use the encoder of an AE to reduce the dimensionality of the InferSent and the TF-IDF embedding.

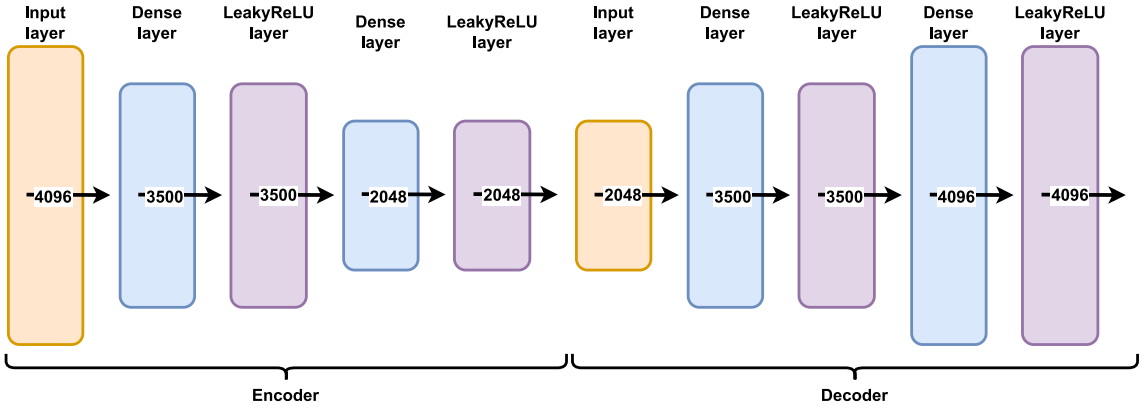


Figure 4.5: Architecture of the AE.

The implementation was provided by the blog post from [33]. It uses the library keras[1]. The architecture is adapted to fulfil the needs of the specific context. It is presented in Figure 4.5.

TF-IDF

The TF-IDF model has to be initialized and trained on the data corpus to build a data-specific vocabulary. An exemplary implementation is given in Listing 4.6. The `TfidfVectorizer` is provided by the `scikit-learn` package. The `input` parameter defines the input type, i.e. `content` means that the input is a list of strings or bytes, whereas `file` assumes the input has a `read` method and `filename` denotes a list of filenames as input [66]. An embedding is obtained using the command from Listing 4.7.

```
1 tfidf_model = TfidfVectorizer(input='content',
2             preprocessor=TfidfTextPreprocessor().transform, min_df=3,
3             max_df=int(len(docs)*0.07))
4 tfidf_model.fit(documents)
```

Listing 4.6: Initialization of the TF-IDF model. Firstly, an instance of the `TfidfVectorizer` class is created. Secondly, the `fit` method is called to fit the model on the documents.

```
1 tfidf_model.transform(text).todense()
```

Listing 4.7: Encoding a text using the TF-IDF model.

The `preprocessor` parameter defines the preprocessing, i.e. string transformation, stage. It is possible to override the default with a custom preprocessing function. The parameters

[1] https://keras.io/ (last accessed: 19/11/2023)

`min_df` and `max_df` define the minimum and maximum document frequency of a word in the corpus to be considered relevant. The default values are 1, i.e. a term has to appear at least once, and 1.0, i.e. a term appears at most in all documents, respectively [66].

By default, the `scikit-learn` implementation uses the `norm='l2'` configuration, i.e. the Euclidean norm. The implementation of TF-IDF in `scikit-learn` is different from the original TF-IDF definition. The difference is the calculation of the IDF part, which is given in Equation 4.1 from [66]. The one is added to M_{ij} due to the parameter `smooth_id=True` by default to prevent zero divisions and to avoid logarithmic divergences due to a zero argument [55]. After calculating the TF-IDF values, they are normalized by the Euclidean norm given in Equation 4.2.

$$\text{idf}(w_{ij}) = \log \frac{1+M}{1+M_{ij}} + 1 \tag{4.1}$$

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + ... + v_M^2}} \tag{4.2}$$

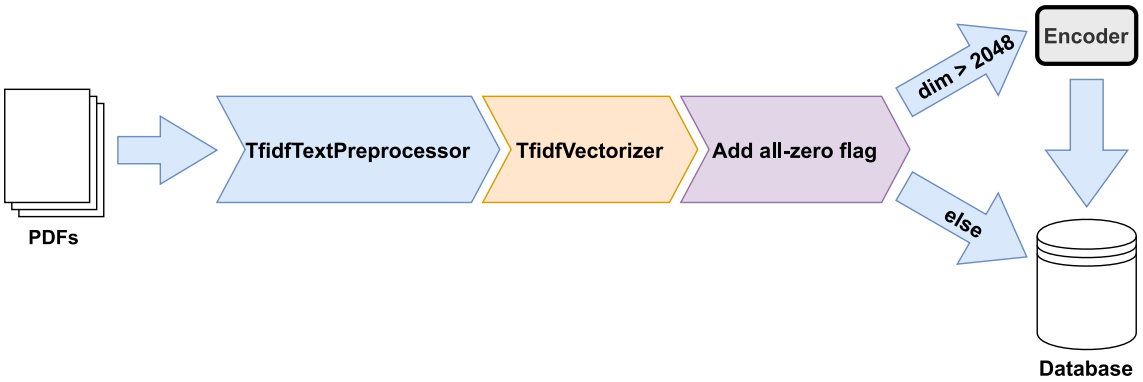


Figure 4.6: TF-IDF pipeline. Firstly, the text extracted from the documents is preprocessed using a custom preprocessor. Then, the TF-IDF values are obtained from the `TfidfVectorizer`. Afterwards, the all-zero flag is added to the TF-IDF weights. If the resulting dimensionality is bigger than 2048, the encoder of an AE is used to reduce the dimensionality. The results are stored in the database.

The pipeline in Figure 4.6 visualizes the steps carried out in this work to derive the TF-IDF embedding of a text and store it in the database. The text of the PDFs is extracted and preprocessed using a custom preprocessor. Thereafter, it is embedded using the `TfidfVectorizer`. The TF-IDF weights are the embedding. Before storing the TF-IDF weights in the database, they are enhanced with an all-zero flag. The all-zero flag ensures that no all-zero vectors are stored in the database by extending those that have a zero magnitude with a "1" entry and "0" otherwise. All-zero TF-IDF weights indicate that a document does not have any terms with the vocabulary in common. Since the vocabulary is kept relatively small with respect to the number of different words in the data corpus to reduce the dimensionality of the embeddings, it is not unlikely that a document does not contain any of the vocabulary terms. The all-zero flag is necessary because the cosine

similarity used to query for similar documents in the database cannot handle vectors of zero magnitude. This alteration does not change the cosine similarity between non-zero magnitude vectors, since the additional zero adds no supplementary information to the calculation of the cosine similarity. The vectors whose all-zero flag is one have a cosine similarity of one when being compared to each other.

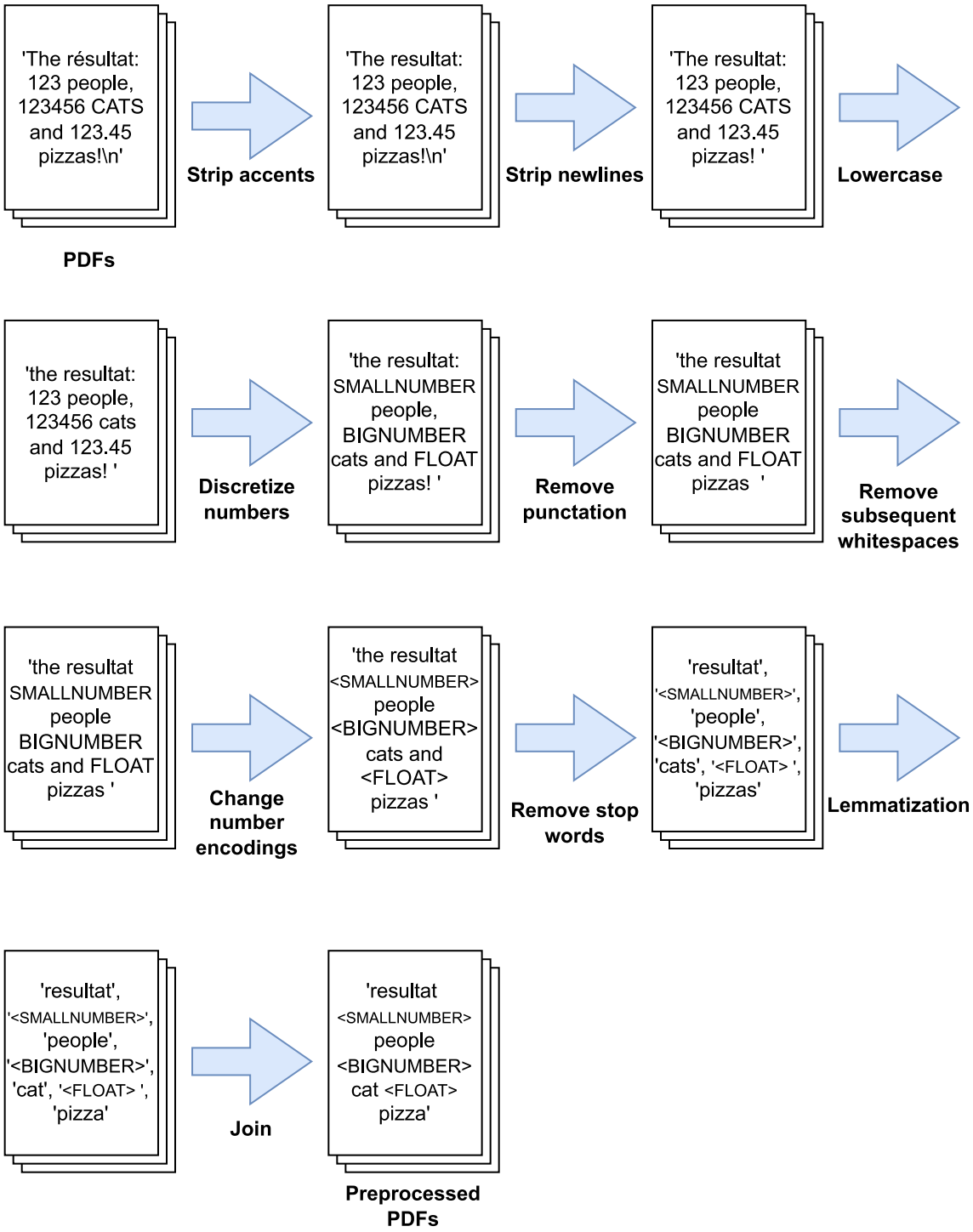


Figure 4.7: Preprocessing visualized using an example text. The stop word removal implicitly tokenizes the text.

The preprocessing steps of the custom preprocessor are visualized in Figure 4.7. Firstly, the accents are stripped from the text. Then, all new line symbols are replaced with a whitespace. Afterwards, the text is converted to lowercase. Then the numbers are discretized, i.e. all numbers between 0 and 99999 are replaced with the string `SMALLNUMBER`, numbers bigger than 99999 are replaced with the string `BIGNUMBER` and floats are replaced with the string `FLOAT`. The next step is to remove all punctuation symbols. To ensure empty tokens generated by prior preprocessing steps are omitted, all sequences of multiple subsequent whitespaces are discarded. After that, the symbols for numbers are enclosed with pointed brackets, e.g. `<SMALLNUMBER>`. Then, the text is tokenized, i.e. split at whitespaces, and stop words are omitted. The stop word list is provided by the `nltk` package and consists of common English stop words. Afterwards, the tokens are lemmatized. The lemmatizer used is the `WordNetLemmatizer` from the `nltk` package. The `WordNetLemmatizer` uses the English lexical database `WordNet` to return valid stems [53]. In the end, the tokens are joined to a string and returned.

Since the dimensionality of the TF-IDF embeddings is big for a large text corpus, an AE is used to reduce the dimensionality of the embedding if its dimensionality exceeds 2048.

Doc2Vec

The library `gensim` provides the Doc2Vec model used in this work. The model is initialized with input data of type `tagged documents`, which are documents with (numerical) tags. In this work, the default parameters are used. The default algorithm is PVDM [24]. The parameters `vector_size` and `window` define the dimensionality of the embeddings and the size of the window, i.e. the maximum distance between the current and the predicted word, respectively. The default value for `vector_size` is 100, whereas the default window size is 8 [24, 23]. The `min_count` parameter defines a threshold below which words will be ignored. Its default value is 5. The `workers` parameter denotes the number of threads to be used for training. The default value is 1 [24]. The `epochs` parameter specifies the number of iterations over the corpus. The default value is 10. By default, the hierarchical softmax algorithm, i.e. `hs=1`, is used for training [78]. Many Doc2Vec default values are adopted from Word2Vec since the `gensim` Doc2Vec implementation inherits from the Word2Vec implementation.

InferSent

The InferSent model is implemented using PyTorch [58]. The parameters used to initialize the model are presented in Listing 4.8. The parameter `version` in line 6 indicates whether the model is trained with GloVe or fastText for the value 1 or 2 respectively. Since the model is precomputed, it is not possible to change certain parameters, such as the word embedding dimension `word_emb_dim` or the dimension of the output vectors `enc_lstm_dim`.

```
'bsize': 64,
'word_emb_dim': 300,
'enc_lstm_dim': 2048,
'pool_type': 'max',
'dpout_model': 0.0,
'version': 1
```

Listing 4.8: Parameters of the InferSent model.

The steps necessary to create a working instance of the InferSent model are presented in Listing 4.9. After the InferSent model is initialized in line 1, the `state_dict` of the model is loaded in line 2. This dictionary consists of learnable parameters, i.e. weights and bias, of the model. The `state_dict` is obtained from the PKL file of InferSent as stated in [18]. The path to the word embeddings is set in line 3. Finally, in line 4, the vocabulary of the model is built. More precisely, only those embeddings needed are kept while the rest is discarded.

```
infersent = InferSent(params_model)
infersent.load_state_dict(torch.load(model_path))
infersent.set_w2v_path(w2v_path)
infersent.build_vocab(docs, tokenize=True)
```

Listing 4.9: Initializing the InferSent model.

In this work, a custom set of vector representations of words is used. The custom word embeddings are computed by a Word2Vec model trained on 2048 randomly selected documents from the Bahamas dataset which reduces the run time of the script. The only parameter which differs from the default settings of Word2Vec is the `vector_size` which is set to 300. After the Word2Vec model is trained, the word embeddings are saved in a file whose file path is the value of `w2v_path` in line 3 of Listing 4.8.

In this work, an AE is used to reduce the dimensionality of the InferSent embedding.

USE

The USE model implemented with TensorFlow [58]. In this work, the fourth version of the model is used. The implementation from Tfhub uses the DAN architecture [68]. The USE file has a size of about 1 GB. It is not necessary to preprocess the data for the model.

SBERT

The SBERT model is implemented with PyTorch [58]. An instance of the model is obtained as shown in Listing 4.10. The model contains a BERT transformer, which has a

`max_seq_length` of 128. It does not convert inputs to lowercase by default [57]. The output of the BERT transformer is passed to a pooling layer, which is initialized with the `pooling_mode` parameter. The default is `mean_pooling`, which calculates the mean of the output vectors of the transformer. The other options include `cls_token_pooling`, which returns the output of the first token and `max_pooling`, which returns the maximum value of the output vectors. The word embedding dimension is 384 by default [57].

```
1 SentenceTransformer('paraphrase-MiniLM-L6-v2')
```

Listing 4.10: Initialization of the SBERT model.

4.1.4 Clustering using OPTICS

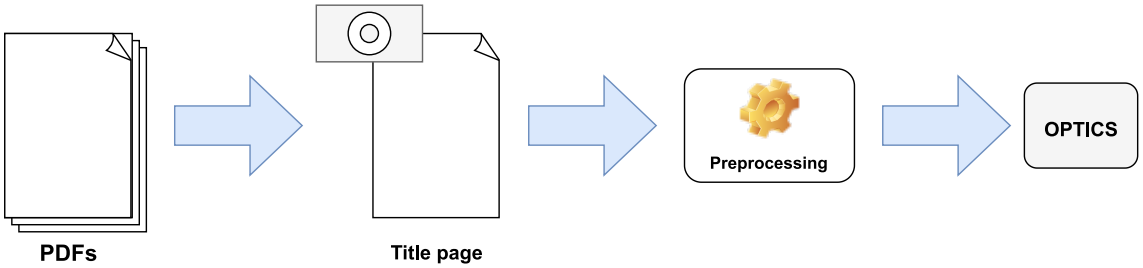


Figure 4.8: The first page of each document is converted to an image. The image is preprocessed, i.e. conversion to greyscale and resizing.

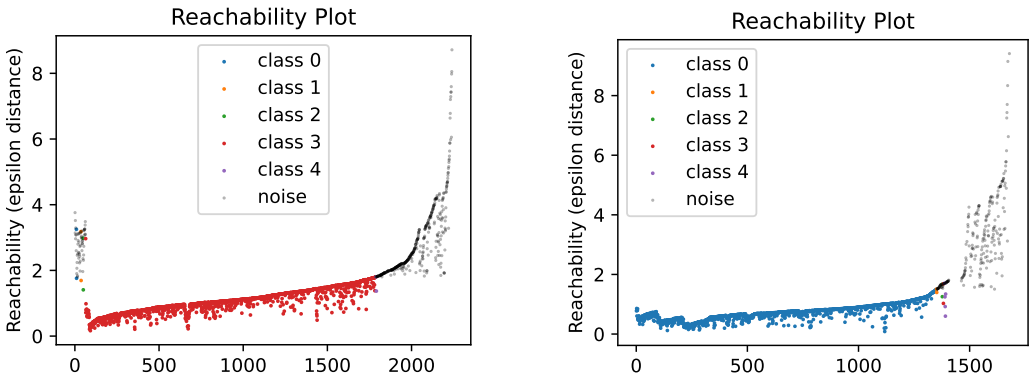
Similar to the approach from Ankerst et al., OPTICS is used to cluster the images of the first page of documents in this work. The procedure is displayed in Figure 4.8. There are two different preprocessing approaches:

1. The images are first preprocessed to 32x32 normalized greyscale pixels (cf. [5]) as visualized in Figure 4.10 and afterwards compressed to 13-dimensional vectors using PCA.
2. The technique Eigendocs from Subsection 3.5.2 is used to compress the images to 13-dimensional normalized greyscale images as displayed in Figure 4.4.

```
1 optics_model = OPTICS(cluster_method='dbscan', min_samples=2, max_eps=10,
2     eps=1.5)
```

Listing 4.11: Initialization of the OPTICS model. The minimum number of samples `min_samples` in a cluster corresponds to *minPts*.

The configurations used when initializing an OPTICS model greatly influence the clusters returned. The parameter `max_eps` is infinity by default but can be specified by the user to reduce complexity and runtime. According to literature, `max_eps` should be big enough to include almost all points in a single cluster. The way the reachability plot is used to extract clusters is dependent on the `cluster_method`. One can choose either `dbscan` or `xi`



(a) The reachability plot of the documents preprocessed according to item 1 (cf. [5]).

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

Figure 4.9: The plot was created using the OPTICS algorithm from the Python library scikit-learn. The underlying dataset consists of 2241 documents from the Bahamas leak. It shows the reachability distance of each document to its predecessor in the order list.

as a clustering method. The parameters `min_samples` and `eps` influence the cluster sizes and number of clusters found for a given clustering approach. The value of `eps` defines the distance between two points to still be considered neighbours and can be chosen by consulting the reachability plot which is displayed in Figure 4.9. The code to initialize an exemplary OPTICS model is displayed in Listing 4.11.

4.1.5 Topic analysis

Two topic analysis approaches are outlined below. The first one serves as the baseline model of this thesis and the second one is used multiple times throughout the application to visualize results obtained by queries.

Top2Vec

Angelov's Top2Vec model is provided in the Python library Top2Vec [4]. In his work, UMAP's hyperparameters are set to 15 nearest neighbours, cosine similarity as the distance metric and 5 as the embedding dimension. The word and document embeddings are generated by the Doc2Vec version PV-DBOW. The window and vector size are 15 and 300 respectively and a hierarchical softmax is used.

In this work, a class is implemented, which uses the Top2Vec library. When initiating an instance of this class, the Top2Vec model is trained on the given document corpus as displayed in Listing 4.12. The class provides methods to query for the number of topics as well as the most similar topics and documents to an input keyword. The most similar topics can be visualized using word clouds. The core functionalities are implemented by

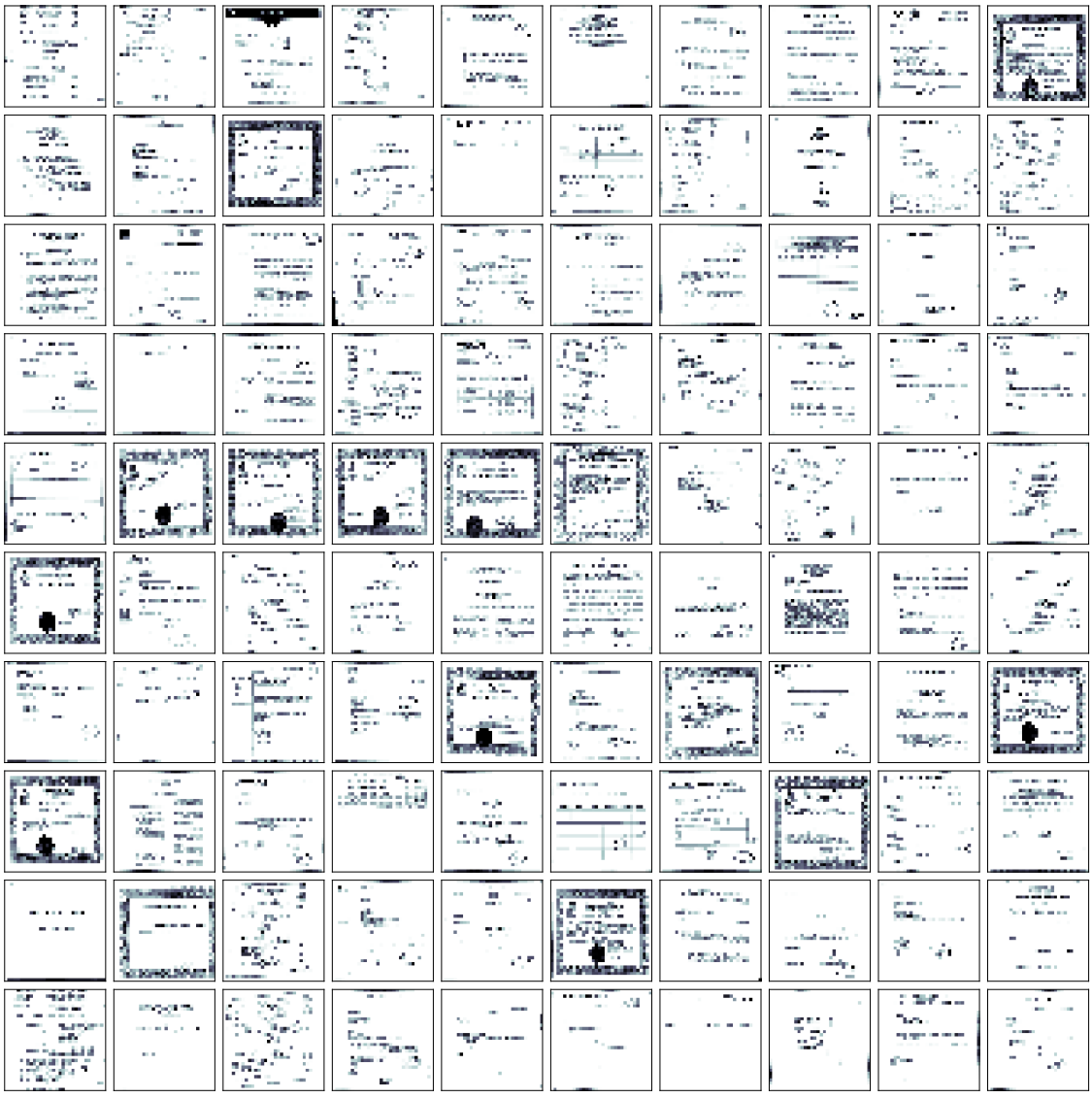


Figure 4.10: Preprocessing of 100 documents to 32x32 normalized greyscale pixels.

the Top2Vec library, but the class is used to modify the return values to be compatible with the UI.

```
1 Top2Vec(documents=self.documents, speed='fast-learn', workers=8)
```

Listing 4.12: Initialization of the Top2Vec model.

Word clouds

The implementation of word clouds in this thesis is based on the Python library `wordcloud` by Müller [51]. This implementation removes English stop words from the text by default. The input text is split into tokens using a regex. By default, plurals are removed if their singular version is present and their frequency is added to their singular version. By default, numbers are not included as tokens.

In order to ensure that the words presented are interpretable, the input text is preprocessed as displayed in Listing 4.13. The `WordNetLemmatizer` from the `nltk` package is used to ensure the stemmed words exist. A word cloud is initialized as shown in Listing 4.14.

```
1 lemmatizer = WordNetLemmatizer()
2 tokens = [lemmatizer.lemmatize(token) for token in tokens]
```

Listing 4.13: Custom preprocessing of word cloud input.

```
1 wordcloud = WordCloud(width=800, height=500, random_state=21,
2     contour_width=3, max_font_size=110, background_color='white',
3     max_words=5000).generate(','.join(tokens))
```

Listing 4.14: Initialization of a word cloud.

4.1.6 Slurm

Since the data corpus is too big to be processed locally on a Apple M2 Pro MNW83D/A with 16 GB RAM and 12 cores, the Chair Intelligent Embedded Systems (IES) has offered to provide computational means to solve this problem. The scripts can be processed by multiple nodes which are managed by Slurm. Slurm is an open-source management tool for Linux clusters [1]. It allocates resources, i.e. compute nodes, and provides the means to start, execute and monitor jobs [1, 73].

The so-called Slurm daemons control nodes, partitions, jobs and job steps [1]. 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 4.11.

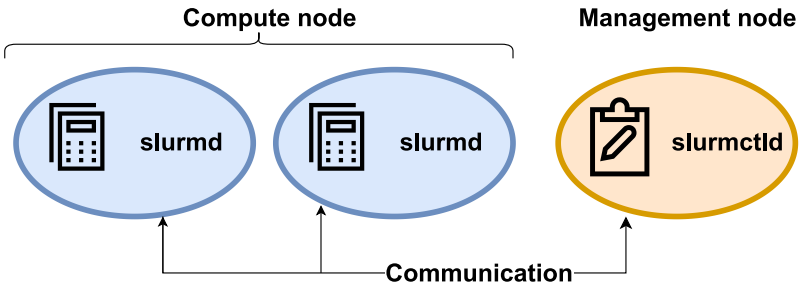


Figure 4.11: Slurm architecture. The management node has a `slurmctld` daemon, while every compute node has a `slurmd` daemon. The nodes communicate. The user can use certain commands, for instance `srun` and `squeue`, anywhere on the cluster.

Table 4.2: A selection of `sbatch` scripts used in this work.

Name of sbatch script	Description
ae_config.sh	Comparison of different AE architectures in terms of the metrics cosine similarity and Root Mean Square Error (RSME).
allocate_res.sh	Allocates resources to enable a SSH tunnel connection from a local VSCode instance to the server of the IES. When enabled, the database content can be displayed with the Elasticsearch plugin.
create_database.sh	Initializes the database by specifying fields.
create_documents.sh	Inserts the document's metadata information, i.e. path and text.
elasticContainer.sh	Starts the Elasticsearch container using the headless `podman-compose up` command.
init_database.sh	Initializes database, subsequentially inserts documents metadata, embeddings and clusters.
insert_clusters.sh	Inserts PCA weights, OPTICS and argmax clusters.
insert_embeddings.sh	Subsequentially inserts embeddings of documents.
own_w2v_model.sh	Creates and saves custom Word2Vec model.
run_pdf2png.sh	Converts and saves the PNG version of the first page of the PDFs.

A job is started by a `sbatch` script. This script defines the `partition`, the `job-name`, the number of `nodes`, the `cpus-per-task`, the memory `mem` allocated, the `time` limit and the path to store `error` and `output` logs. It is possible to work on multiple CPUs simultaneously to divide the workload of a task. In this work, multiple `sbatch` scripts are used to carry out a variety of tasks. A summary of the tasks and scripts is given in Table 4.2.

4.2 Web interface

A basic web interface is provided to facilitate the comparison of the models explored in this thesis. However, the focus of this work is on the methods and not on the application. The tool consists of a backend and a frontend which are described in Subsection 4.2.1 and Subsection 4.2.2.

4.2.1 Backend

The framework used for the backend is Flask. There are multiple endpoints, which are used to retrieve data from the server:

- Documents: Returns a list of documents, which best match the query. The information returned for each document is the respective `id`, `path`, and `text`. The query can be of type `match_all`, which returns all documents in the database, or a fuzzy full-text query if `text` is specified, or a kNN query on a certain field of the database if both `knn_type` and `knn_source` are given. Moreover, the number `count` and start index `page` of the results returned can be specified. By default, the first 10 documents are returned.
- Document: Returns the metadata, i.e. text and path, of a document with the specified `id`. The URL to access this endpoint is `/documents/<id>`.
- PDF: Returns the PDF file. This endpoint is used to display the PDF in the detail component of the frontend. The URL to access this endpoint is `/documents/<id>/pdf`.
- WordCloud: Returns the bytes of a word cloud image. Depending on additional parameters, the word cloud is either generated from one document or a group of similar documents. If the `knn_type` is specified, a query for the `count` most similar documents is performed. By default, `count` is 10. The URL to access this endpoint is `/documents/<id>/wordcloud`.
- Term Frequency: Returns the term frequency calculated for the specified document. The URL to access this endpoint is `/documents/<id>/term_frequency`.
- TopicWordCloud: If `term` is specified this endpoint returns a word cloud of the terms that describe the topics most similar to the query term. The parameter `count` specifies the number of topics to be returned. Its default value is 3. The topics are generated by Top2Vec. The URL to access this endpoint is `/topics/wordcloud`.
- Topics: Returns the topics generated by Top2Vec. The topics are described by the words closest to the topic vectors. The URL to access this endpoint is `/topics`.

4.2.2 Frontend

The framework used for the frontend is Angular. There are three main components, which are used to display the data:

- Home: The home component is used to display the results of a text query. It consists of a search bar, which is used to enter the query term, and a list of results. If no text query is entered the first documents of the database, i.e. the result of a `match_all` query, are displayed. The search component is shown in Figure 4.12.
- Detail: The detail component is used to display the details of a document. The document name and ID are located on the left side of the screen. Beneath the document name and ID, a button which opens the term frequency image on a new page is located. Moreover, the word cloud of the document is displayed. The word cloud is generated from the text of the document. On the right side of the screen, there is a PDF viewer which displays the pages of the document. Beneath the PDF viewer, the file names and a word cloud of the most similar documents are displayed after a query for them is initiated by the user. The detail component is shown in Figure 4.13.
- Topic: The topic component is used to display the topics of the documents. The topics are lists of words generated by `top2vec`. The user can query for the most similar topics to a term. The results are displayed as a word cloud. The upper limit of the number of topics can be defined by the user. The topic component is shown in Figure 4.14.

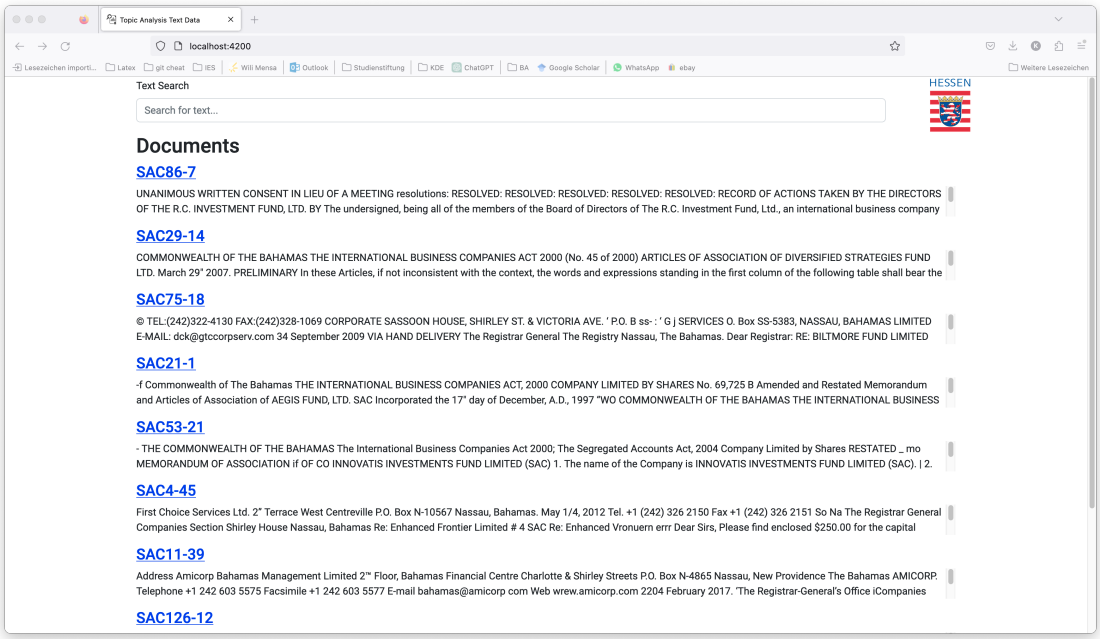


Figure 4.12: Home component of the frontend. The search bar is used to enter the text query. The results of the query are displayed below the search bar.

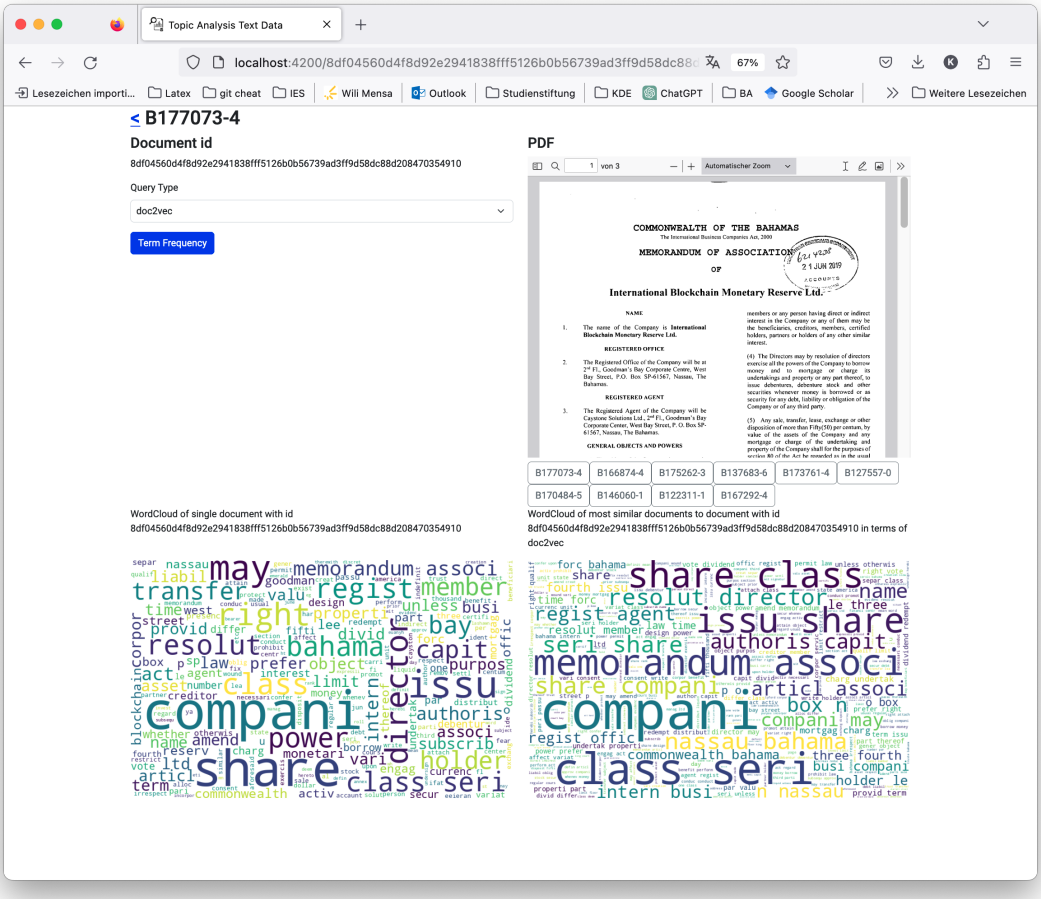


Figure 4.13: Detail component of the frontend. The chosen document is displayed, as well as its most similar documents in the database. The word clouds of the document and the most similar documents are displayed.

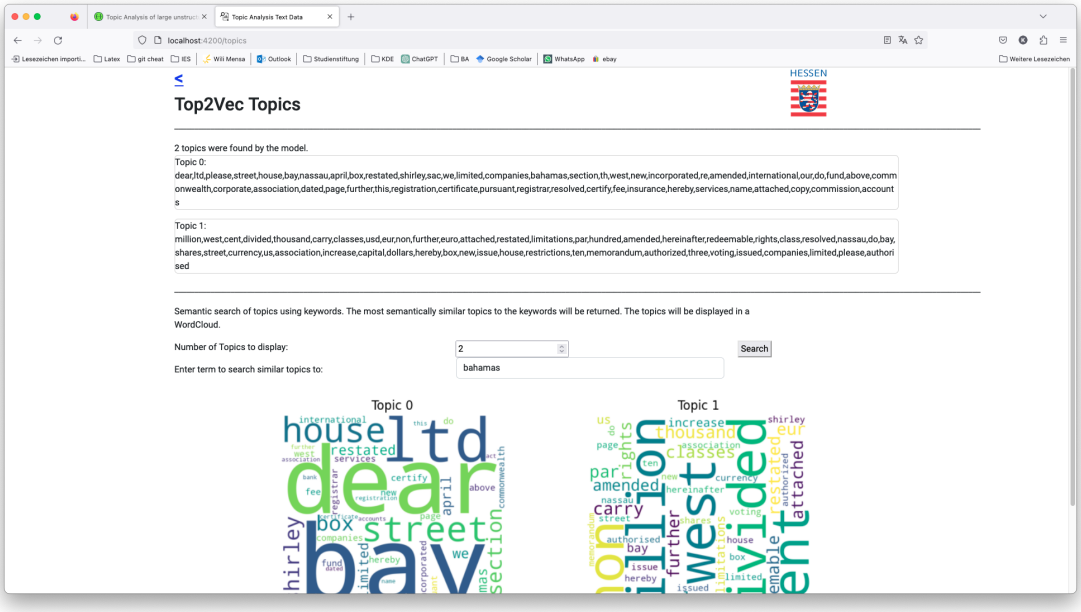


Figure 4.14: Topic component of the frontend. The topics identified by Top2Vec are listed. Below them, the user can query for the most similar topics to a term. The results are displayed as a word cloud.

4.3 Trade-off between memory and query time

At the beginning of this thesis, it was unclear to what degree the tool, i.e. the database fields and query results should be precomputed. A tool which is trained once offline is beneficial due to the amount and the nature of the data. The Bahamas leak is static and thus, the database does not need to be updated with new documents.

In the course of filling the database with information, one had to face obstacles not only regarding excessive memory usage but also long run times of methods. Early on it became evident that one either had to reduce accuracy and details in order to achieve less memory or one had to settle for minutes to hours of calculations and bigger costs in terms of memory consumption.

Beforehand, it was not clear which information, i.e. fields in the database, seemed worth the time and memory. For instance, initially, the image of the first PDF page of each document was saved alongside the other fields within the database. After scaling the amount of data stored in the database to about 2900 documents, this approach caused severe issues in terms of memory usage. Hence, this field is omitted.

5 Evaluation

Since the dataset has no ground truth, the procedure used to pick the parameter values is not comparable to ground truth-based approaches. Hence, the evaluation is informal and the methods applied have arisen from regular consultation with experts from the tax office. Run times of different configurations are measured and compared. Parameters are chosen with respect to model-specific procedures, such as reachability plots for OPTICS. The models are compared to each other and the tool constructed from the composition of the models is compared to the baseline topic analysis model Top2Vec.

5.1 Database

There is a variety of parameter values to choose from when working with databases. The choice of the similarity metric is discussed first. Secondly, the reasons for choosing Elasticsearch as a database are presented.

Similarity measurements

According to Reimers and Gurevych, the similarity measurements discussed in Section 3.3 obtained roughly the same results in their experiments [58].

As the similarity between vectors is usually calculated using some form of cosine similarity, rather than Euclidean distance in literature, cosine similarity is preferable over Euclidean distance. Since the models may produce embeddings which are not normalized, the cosine similarity is used instead of the dot product.

Elasticsearch

According to Grinberg, Structured Query Language (SQL) databases are a good choice for efficiently storing structured data. This is because their paradigm ACID, i.e. Atomicity, Consistency, Isolation, Durability, provides high reliability. Not only SQL (NoSQL) databases, on the other hand, are more flexible and can be used to store unstructured data [25]. They do not require a predefined schema and can therefore accept documents of arbitrary structure [22]. Usually, NoSQL databases do not offer services such as `JOIN`s. NoSQL databases are said to outperform out-of-the-box SQL databases. Since the dataset consists of unstructured documents and the task at hand does not require performing any `JOIN`s, a NoSQL database is favourable. Elasticsearch is chosen since it is well known to

provide near real-time search and to operate on big data. Subsequently, it is a good fit for the underlying dataset.

Since Elasticsearch stores vectors of at most 2048 dimensions, the TF-IDF and InferSent embeddings are problematic. Besides imposing limits to the dimensionality of the embeddings, Elasticsearch offers a variety of convenient functionalities, such as the built-in kNN search. Therefore, in this work, Elasticsearch is used regardless of the dimensionality constraints imposed by the database. Hence, the techniques are adjusted to the database and not vice versa.

The time necessary to fill the Elasticsearch database has been evaluated and improved throughout this work. The current time measurements are shown in Figure 5.1. The times correspond to calculation of 2048 embeddings. It is possible to measure this compuation time individually since the task of filling the database is modulized. Modulizing is beneficial since it is possible to update the embeddings without having to recreate the database. Moreover, it facilitates debugging and comparing the models used to create the embeddings.

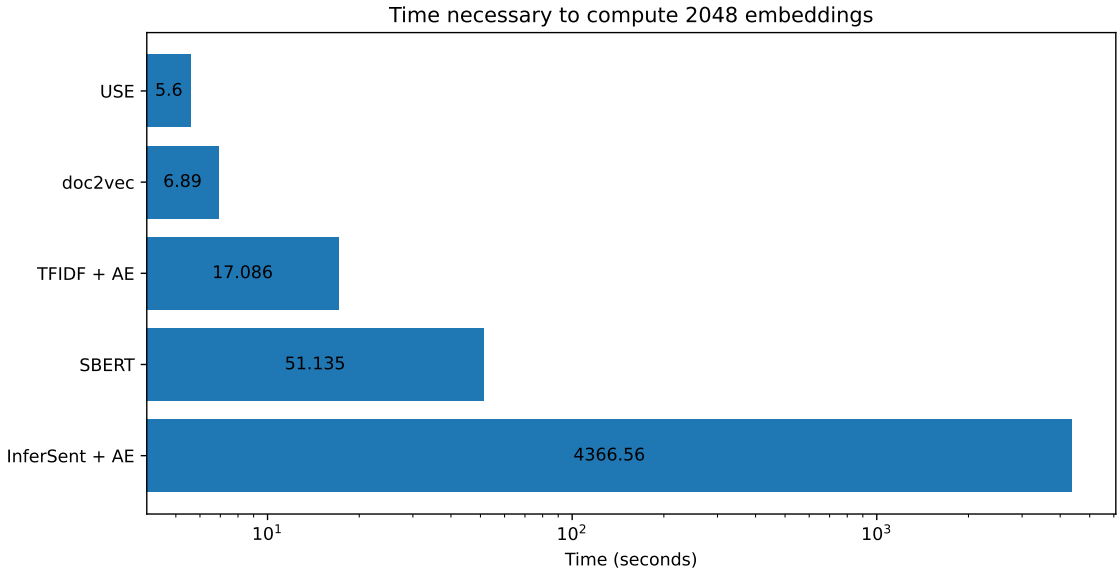


Figure 5.1: Time per module of creating the Bahamas database using a random selection of 2048 documents. The x-axis is logarithmic. The reference time is measured using `cProfiler` on a Apple M2 Pro MNW83D/A with 16 GB RAM and 12 cores.

5.2 Eigendocs

In order to determine the optimal number of components used for Eigendocs the cumulative explained variance and the reconstruction error are plotted as displayed in Figure 3.15 from Subsection 3.5.2. The first plot indicates that 90% of the variance is explained by 441 components. Usually, that would have been the number of dimensions of the subspace onto

which the documents would have been projected. However, when working with clustering algorithms like OPTICS, the number of dimensions should be reduced even further to achieve valid clusters. Therefore the reconstruction error with respect to different numbers of components is taken into consideration.

```
sqr_dif = (X_test - X_test_pca_inverse)**2
reconstr_err.append(np.sqrt(np.mean(sqr_dif))/
    (np.sum(np.abs(1-X_test))/X_test.shape[0]))
```

Listing 5.1: Adaption of the RSME: Firstly, the squared differences between the original and the reconstructed images are calculated. Since the values are normalized, a 1 corresponds to a white pixel. Then, the absolute values of all non-white pixels of the test set are summed up. The average number of non-white pixels is calculated by dividing the sum by the number of images in the test set. This approach considers pixels of value $p \in [0, 1]$ as $(p \cdot 100)\%$ white and thus, they are incorporated in the sum.

Usually, a RSME is minimized to determine the optimal parameter configurations. In this case, the reconstruction error shall be interpreted. To facilitate the interpretability of the reconstruction error, its calculation is adapted to incorporate the content of the images. At first sight, the majority of image pixels are white, i.e. do not convey any information. Therefore, the reconstruction error is divided by the average number of non-white pixels. Hence, the reconstruction error of an image is weighted by the amount of information it conveys. The calculation is given in Listing 5.1. The result is displayed in Figure 3.15. Since the reconstruction error increases less rapidly after 10 to 20 components, the number of components is set to 13, which has been an “elbow” point in a similar trial using a not randomly selected dataset of 195 images.

Some impressions of the Eigendocs algorithm are displayed in Figure 4.4. Assuming that the selection of documents is representative, the preprocessing of the documents using Eigendocs should have encoded information about the dimensionality of the images. However, this assumption is not valid since bigger images exist. Therefore, the idea of incorporating information about the image’s dimensions is not entirely implemented.

5.3 Embeddings

As discussed in Subsection 4.1.3, there is a range of possible parameter values to choose from when implementing embedding models. The section below states which findings have led to the parameter values applied in this work.

TF-IDF

The main obstacle to overcome is the high dimensionality of the TF-IDF embeddings. Hence, the goal of the parameter selection is to find a way to reduce the dimensionality of the vocabulary to 2048 which is the maximum dense vector dimensionality of Elasticsearch. However, the quality of the embeddings should not decline too much.

The choice of the preprocessor is investigated with regard to the goal of minimizing the vocabulary size. Both the default and a custom preprocessor are tested on a data corpus of 2048 randomly selected documents concerning the vocabulary (size). While the default preprocessor had a vocabulary size of 5893, the custom preprocessor had a size of 5585. The relative differences between vocabulary sizes seem to be inversely proportional to the dataset size since the trend is already visible for two different data corpus sizes in Table 5.1. The custom preprocessor is chosen because it had a smaller vocabulary size. The differences between both vocabularies are visualized in Figure 5.2.

Table 5.1: Comparison of vocabulary sizes resulting from the default and the custom TF-IDF preprocessor on different data corpus sizes.

	first trial	**second trial**
document corpus size M	195	2048
custom preprocessor vocabulary size A	1521	5585
default preprocessor vocabulary size B	1641	5893
(B-A)/M	120/195 = 0,6153846154	308/2048 = 0,150390625

(a) The terms only present in the vocabulary produced by the default preprocessor.

(b) The terms only present in the vocabulary obtained from the custom preprocessor.

Figure 5.2: The word clouds visualize which words are unique to both vocabularies on a random selection of 2048 documents.

As stated in Section 5.1, the TF-IDF embeddings can be problematic with regard to the dimensionality limitations imposed by Elasticsearch. The parameters `min_df` and `max_df` are set to values which keep the vocabulary size small and thus, the dimensionality of the embeddings is reasonably small. Furthermore, this work employs dimensionality reduction

techniques to reduce the dimensionality of the embeddings if the embeddings have a higher dimensionality than 2048.

Doc2Vec

Since no labeled data is available, the evaluation of the Doc2Vec embeddings is limited. Therefore, the Doc2Vec embeddings are evaluated by comparing them to other embeddings. The Doc2Vec model is not tuned in terms of hyperparameter selection, but the default settings are used since there is no way to evaluate the resulting embeddings.

InferSent

The `max` pooling type is used for the InferSent model, since Conneau et al. found by conducting experiments using different pooling techniques that it is the best option [14].

Initially, in this work, the Global Vectors (GloVe) word embeddings were used for the InferSent model. However, since the file of precomputed GloVe word embeddings has a size of 5.65 GB and thus, slows down the model, ultimately another word embedding is used. The time necessary to compute and insert 195 documents for specific embeddings is displayed in Figure 5.3. The custom word embedding used in this work is a Word2Vec model trained on a selection of 2048 randomly selected documents from the Bahamas dataset.

Pennington et al. state that GloVe outperforms Word2Vec on the same corpus, vocabulary and window size in terms of quality and speed [55]. Hence, the quality of the results obtained in this work may have suffered from using a custom Word2Vec instead of GloVe. However, since the computation time of the project is a crucial factor, the custom Word2Vec is used.

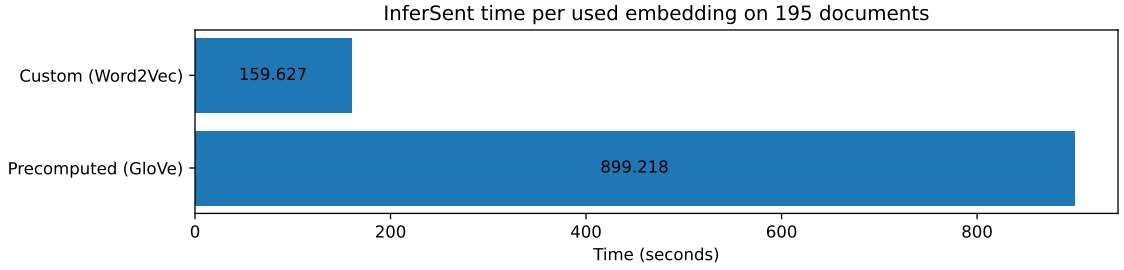


Figure 5.3: Reference time necessary to calculate and insert 195 InferSent embeddings for different precomputed word embeddings on a Apple M2 Pro MNW83D/A with 16 GB RAM and 12 cores. Using a custom Word2Vec model is around 5.5 times faster than GloVe.

USE

Since there are no parameters to customize the evaluation of the USE embeddings is limited. Therefore, the USE embeddings are evaluated by comparing them to other embeddings.

AE

In order to determine which architecture for the hidden or so-called latent space of the AE is the best option, different architectures are tested and compared in terms of RSME and cosine similarity. The RSME is calculated as given in Listing 5.2. The cosine similarity is calculated as given in Listing 5.3. Due to the fact that cosine similarity values are bound by 0 and 1, they are easier to rank than metrics that can yield any real number. However, cosine similarity is usually applied to calculate the angle between two vectors and thus, one has to be cautious when interpreting the results. For instance, the vectors $(0, 1)^T$ and $(0, 2)^T$ have a cosine similarity of 1, even though they are not the same vectors. Since an AE is supposed to reconstruct the input rather than return a dependent or related vector, this metric should be combined with a traditional metric. The dataset used for the evaluation is a selection of 195 documents from the Bahamas dataset.

```
1 rsme = np.linalg.norm(inverse_embedding - embeddings)
2         / np.sqrt(embeddings.shape[0])
```

Listing 5.2: Computation of the RSME between the original and the reconstructed embedding.

```
1 cos_sim = statistics.mean([np.dot(inverse_emb, embedding)
2         /(np.linalg.norm(inverse_emb)*np.linalg.norm(embedding))
3         for inverse_emb, embedding in zip(inverse_embedding, embeddings)])
```

Listing 5.3: Computation of the cosine similarity between the original and the reconstructed embedding.

The scores of different architectures are shown in Figure 5.4. The x-axis displays the number of neurons in each layer for the respective experiments. The input space is 4096-dimensional since that is the dimensionality of InferSent embeddings. The output of the encoder is 2048-dimensional which is the maximum dimensionality supported by Elasticsearch for dense vectors. While most of the architectures produce similar results, one architecture stands out. Combining 2500-, 3000- and 3500-dimensional layers in the hidden space produces the worst RSME results. The smallest RSME and the biggest cosine similarity are achieved by adding a 3500-dimensional layer in the hidden space. However, the results of the best architecture do not differ greatly from the others.

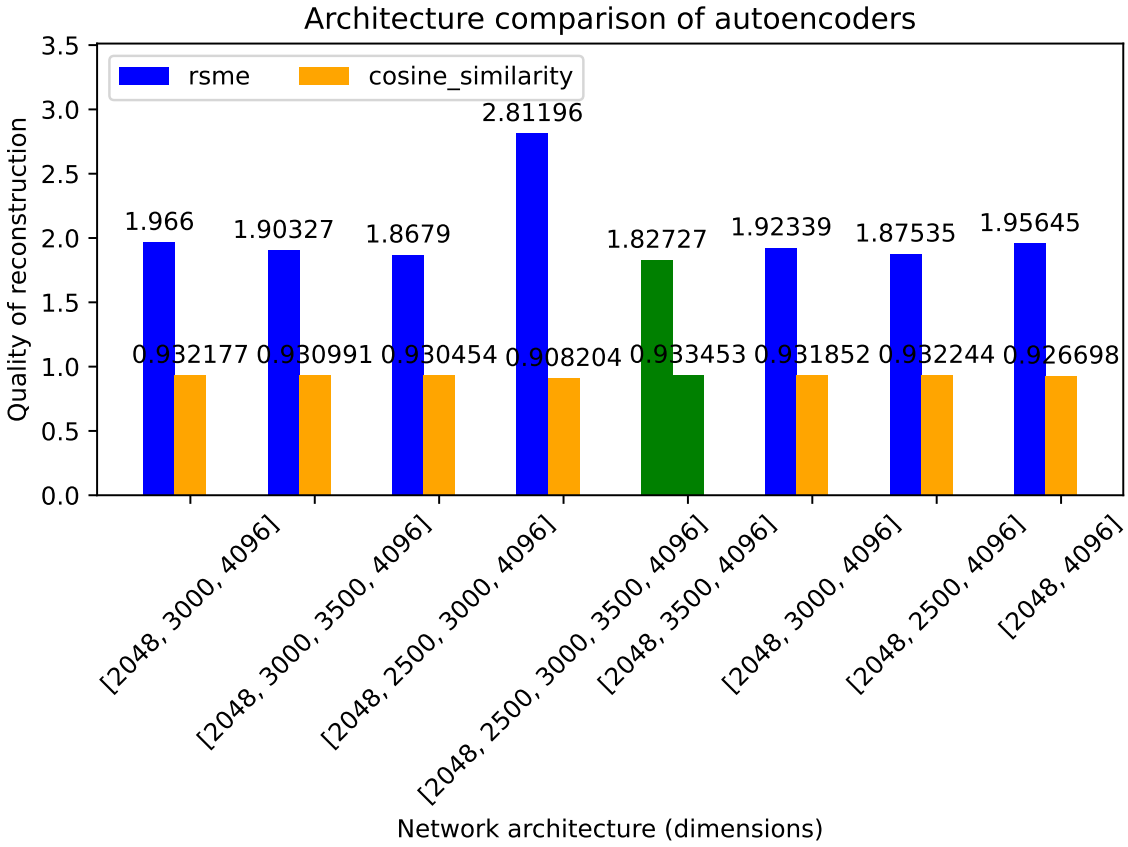
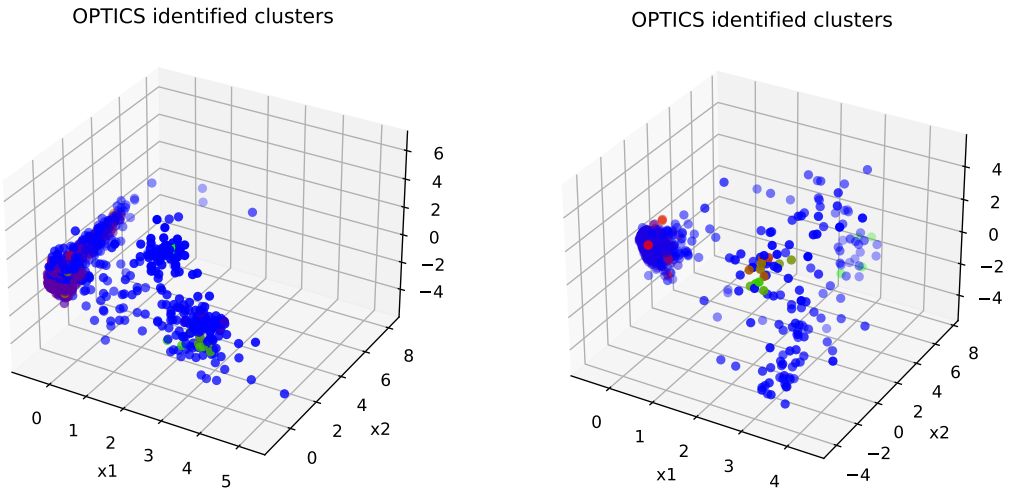


Figure 5.4: The effect of different AE architectures on the reconstruction error. The error is measured in terms of RSME (blue bars) and cosine similarity (yellow bars) between the original and the reconstructed image. The smallest RSME and the biggest cosine similarity belong to the architecture best suited to this task and are coloured green.

5.4 Clustering using OPTICS



(a) Preprocessing according to item 1 (cf. [5]).

(b) Preprocessing according to item 2 (i.e. Eigendocs).

Figure 5.5: The clusters are extracted from the respective reachability plots in Figure 4.9 by OPTICS. The points in the three-dimensional space correspond to the weights of the first three principal components. The blue points are noise points, whereas any other colour denotes a cluster.

The algorithm OPTICS is applied to data, which is preprocessed according to item 1 (cf. [5]) and item 2 (i.e. Eigendocs) from Subsection 4.1.4. The clusters from Figure 5.5 are extracted from the respective reachability plots in Figure 4.9. The three-dimensional plots visualize the first three dimensions of the data and thus, the weights of the first three principal components assigned by the Eigendocs algorithm. By visual inspection and comparison of both plots, it can be seen that the projection by the Eigendocs approach of item 2 scatters the objects further in the three-dimensional space. One could argue that this is due to the fact, that the input data encodes not only the visual appearance in terms of page layout but also the size of the document. Possibly, the objects are grouped by document size.

To analyze the results of the clustering, the content of the clusters is examined. Since the documents are not labeled, the content of the clusters is analyzed by visual inspection and displayed in Figure 5.6. The yellow images belong to the group identified as noise. The images preprocessed according to item 1 are partitioned into more clusters than the Eigendocs approach. Most of the certificates are classified as noise for both approaches in the trials carried out.

The approach from item 1 (cf. [5]) omits information about the images' original size. This information is encoded in the Eigendocs approach. Hence, the preprocessing approach chosen to create the OPTICS input for the Elasticsearch database index is Eigendocs.

According to Deng et al., OPTICS was developed to improve DBSCAN flaws. With respect to the evolution of these clustering methods, i.e. DBSCAN being OPTICS basis, DBSCAN

(a) Preprocessing according to item 1 (cf. [5]).

(b) Preprocessing according to item 2 (i.e. Eigendocs).

Figure 5.6: In this visualization, at most 10 random elements of a cluster are displayed. The yellow images belong to the group denoted as noise. Most certificates are classified as noise.

is chosen for the clustering method in Listing 4.11. In order to reduce calculation complexity the maximum ε is 10. The distance between two points to still be considered neighbours is defined after a visual inspection of the reachability plot. Considering the intrinsic structure of the Eigendocs data, it is set to 1.5 to return meaningful clusters.

5.5 Comparison of models

This evaluation does not aim to find the best model but to compare the similarity of the models' query results. It is a qualitative evaluation of a selection of documents from the Bahamas leak. This selection is a 2048 document corpus that is randomly chosen without replacement. A query defines a field, i.e. embedding model, and a query document. The query response consists of the documents that are considered most similar to the query document in terms of cosine similarity.

The differences between the models are illustrated by visualizing the first five response documents of a sample query. The text of the query document is encoded using the respective model. A kNN query is used to obtain the results from the local database containing 2048 documents. The query results are displayed in Figure 5.7. The query document, i.e. the image surrounded by a border, is omitted from the query response. The documents in the query response are listed according to descending similarity to the query document. All models except Doc2Vec and USE returned only documents of *CREDIT SUISSE*. Apart from this difference, the query responses of the models are very similar.

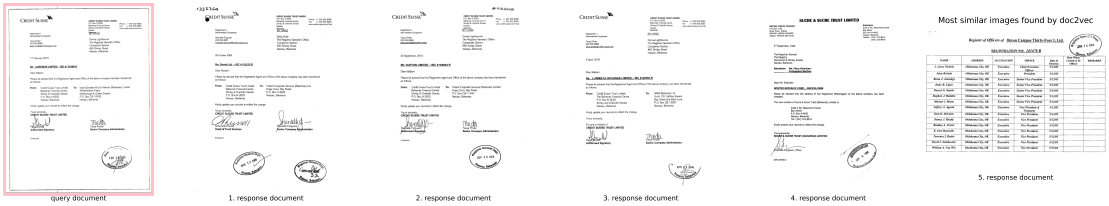
To further investigate the differences between the models, the query responses of the models are compared qualitatively on query documents that are considered unusual in terms of their appearance. The document in Figure 5.8 is a table consisting of little text compared to other samples from the data corpus. The document in Figure 5.9 is mostly handwritten, which is unusual since most other samples are computer-generated.

The models produced good query responses on a query document consisting of little text as shown in Figure 5.8. Even though at first glimpse, the response documents seem unrelated to the query document, they share multiple words, such as *director*. Similar response documents do not have to be of similar visual appearance since the text embeddings only consider information from the text layer.

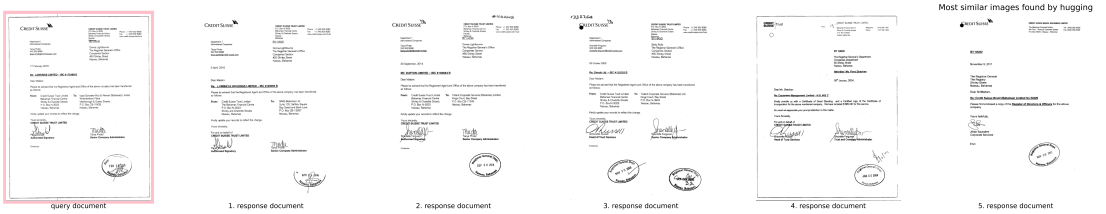
There are query documents such as the one in Figure 5.9 that reveal the dissimilarity between certain models. Most models' response documents are similar to Figure 5.9(b). These response documents depict the same type of document, i.e. handwritten receipts for an annual fee. The TF-IDF model, however, returns different response documents. The response documents from Figure 5.9(a) are not handwritten and cover different content, i.e. requesting payment and three documents concerning an address change. TF-IDF's results could thus be considered to be of poor quality.

Since not only textual information but also visual information is encoded in the database, the next step is to compare the query responses of approaches that consider visual similarity. The query responses are clustered using OPTICS or `argmax` of the PCA compression.

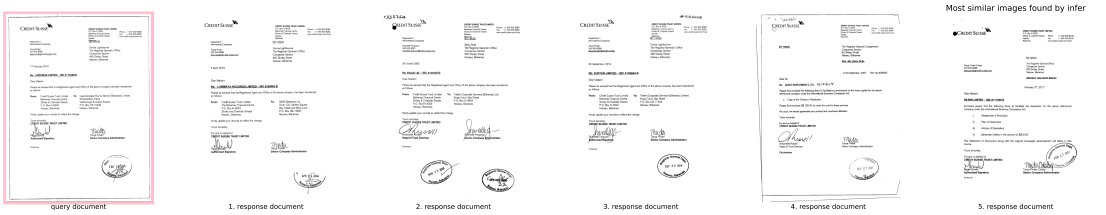
The first exemplary query document in Figure 5.10 is an image of a usual document. Both clustering approaches yield similar results. More specifically, the responses share two documents. Moreover, the last document in the response of both approaches differs most



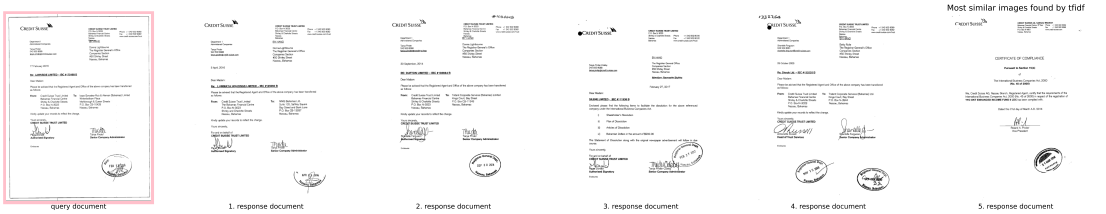
(a) Doc2Vec



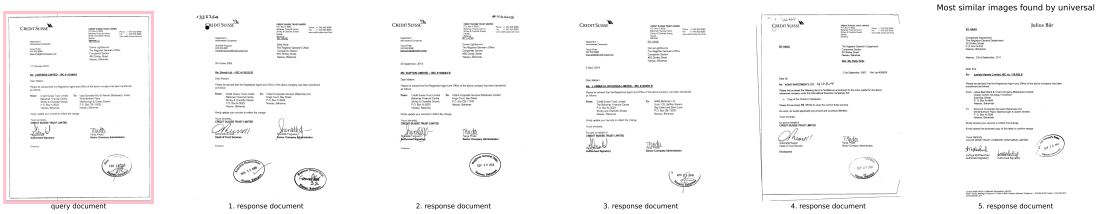
(b) SBERT



(c) InferSent



(d) TF-IDF



(e) USE

Figure 5.7: Exemplary query response for different embeddings on the same query document.

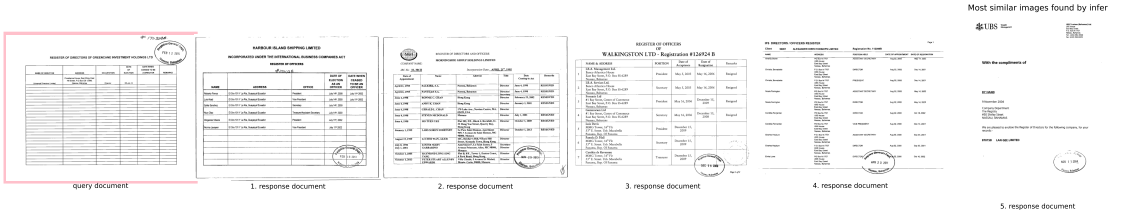
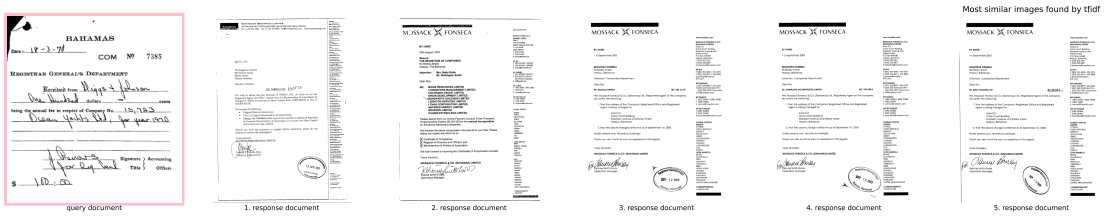
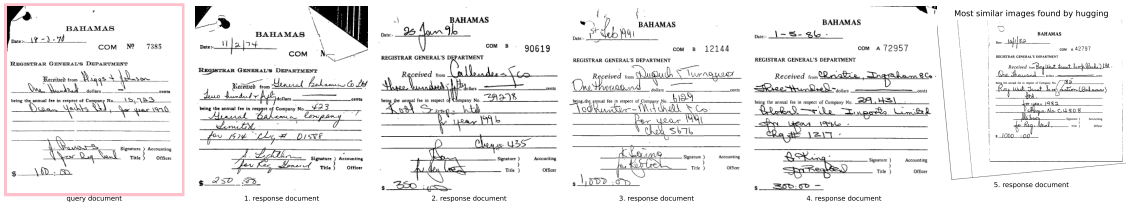


Figure 5.8: InferSent query responses on a query document consisting of little text.



(a) TF-IDF



(b) SBERT

Figure 5.9: Qualitative comparison of query responses. The majority of the query document consists of handwritten text. The results of the TF-IDF model are not similar to the query document. The other models, including SBERT, produce results that are more similar to the query document.

from the group. Most documents have a similar visual appearance, i.e. they have a stamp. None of the result documents originate from the same company as the query document.

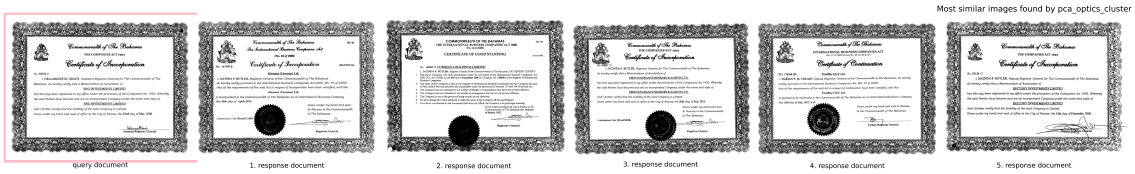
The second exemplary query document in Figure 5.11 is a certificate. The OPTICS clustering approach yields a response that is more similar to the query document than the `argmax` of the PCA compression because all its responses are from the same document type as the query document. Hence, the OPTICS clustering approach is considered to be superior to the `argmax` approach.

(a) OPTICS

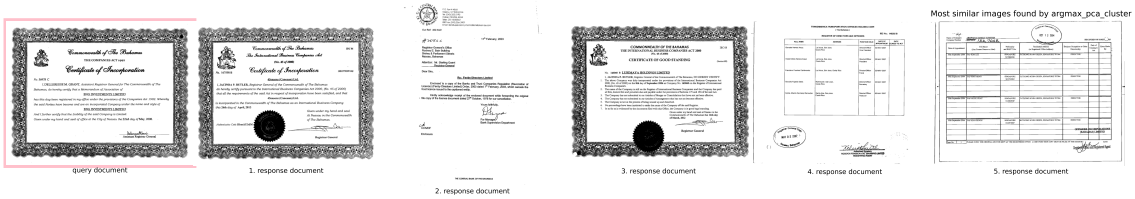
(b) `argmax` of PCA compression

Figure 5.10: Qualitative comparison of query responses. The response documents are clustered using OPTICS or `argmax` of the PCA compression. They are not compared in terms of textual but visual similarity.

Another approach to compare the response documents is to visualize the intersection of the query results. The Venn diagram is chosen since it displays intersections of all items of



(a) OPTICS



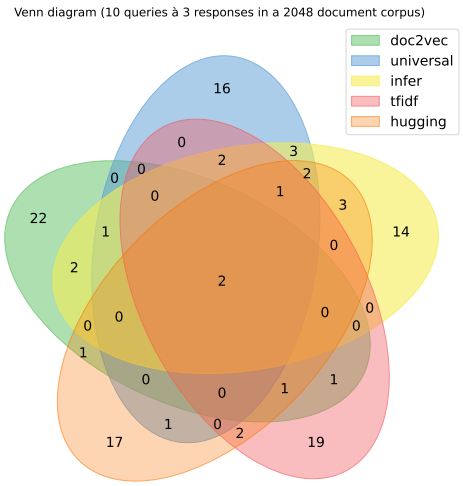
(b) `argmax` of PCA compression

Figure 5.11: Qualitative comparison of query responses. The response documents are clustered using OPTICS or `argmax` of the PCA compression. They are not compared in terms of textual but visual similarity. The query document is a certificate.

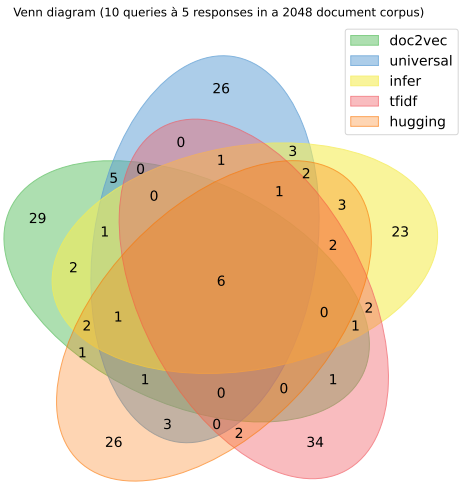
the power set of the models. To build a Venn diagram, the number of documents that are shared between the query results of several models is computed. First, all query responses of a model irrespective of the query document are saved in a set. The cardinalities of the intersections of multiple sets are displayed in Figure 5.12. Since five models encode textual information, the Venn diagram consists of five circles. One should be cautious when interpreting the layout of the Venn diagram since the cardinality of an intersection of documents does not correlate with its area in this visualization.

The Venn diagrams in Figure 5.12 display the total number of shared response documents for 10 queries and 3, 5 or 10 response documents respectively. The models produce rather dissimilar query results. Any combination of more than two models among the green (Doc2Vec), orange (SBERT), red (TF-IDF) and blue (USE) models seem to produce rather dissimilar results as the number in the respective areas is close to zero across all Venn diagrams.

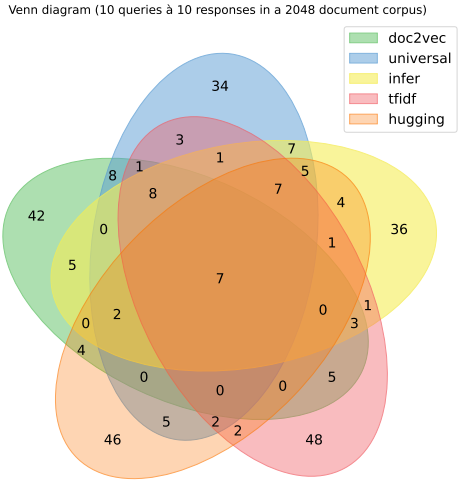
Since the Venn diagrams compare the responses of the models irrespective of the query document, another approach is to compare the query results of the models for each query document individually. This approach first constructs a matrix of the number of shared query results between all model pairs summed up over all query documents. The matrix consists of five rows and five columns, where each row and column represents a model. The cell values are the number of shared query results between the models of the row and column. It is possible to normalize the matrix to obtain the portion of shared query results. If two models produce the same query results, the cell value is either the total number of query results or 1 if the matrix is normalized. Since the matrix is symmetric, only the upper triangular matrix is computed and the other half is mirrored. The matrix is visualized using a heatmap as displayed in Figure 5.13. The code snippet in Listing 5.4 shows the calculation of the similarity matrix.



(a) 3 response documents per query.



(b) 5 response documents per query.



(c) 10 response documents per query.

Figure 5.12: 10 documents are randomly sampled from a 2048 document corpus. For each sampled document and model, a kNN query is conducted. The respective response documents excluding the query document are saved. The cardinality of the intersection of all response documents irrespective of query document for different models is visualized in terms of Venn diagrams.

```
sim_matr = np.matrix(np.zeros((len(model_names), len(model_names))))
for id in df.index:
    for i, model in enumerate(model_names):
        for j in range(i, len(model_names)):
            sim_matr[i, j] += np.sum([df.loc[id,
                model_names[j]].count(item) for item in df.loc[id, model]])
            sim_matr[j, i] = sim_matr[i, j]
if normalize:
    sim_matr /= np.array(len(df.index)* len(df.iloc[0,0]))
```

Listing 5.4: Calculation of the similarity matrix used to produce the heatmap.

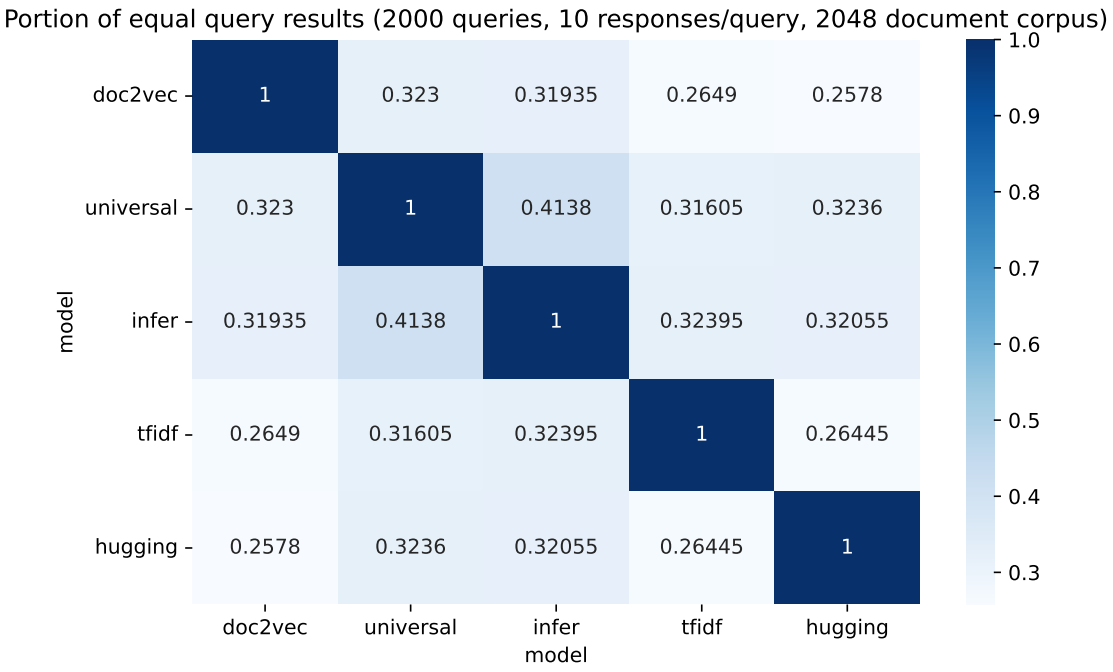


Figure 5.13: Heatmap visualizing portion of shared query results on 2000 queries with 10 responses each on a 2048 document corpus.

The heatmap in Figure 5.13 shows that the models yield dissimilar query responses. USE and InferSent produce the most similar responses, which indicates the maximum value of shared query results. However, the maximum is less than 0.5 and thus, rather dissimilar. Any two-element combination of Doc2Vec, TF-IDF and SBERT produces the most dissimilar query results, namely only around 25% of shared response documents per query.

Since the normalization does not consider the distribution of the cardinalities of the intersections of the query results among the models, another approach is to calculate the mean and standard deviation of the cardinalities of the intersections of the query results. Therefore, 30 trials are conducted. Each trial consists of 10 randomly sampled query documents with 10 responses each (excluding the query document in the response). The normalized similarity matrix for each model pair concerning one trial is calculated using Listing 5.4. The cardinalities are stored in a dictionary of lists indexed by the respective model pairs. Finally, the mean and standard deviation are computed for each model pair and stored in a Comma Separated Values (CSV) file. The results are displayed in Table 5.2. The standard deviation does not exceed 0.1 for any model pair and is similar among all combinations. As observed before, any two-element combination of Doc2Vec, TF-IDF and SBERT has the lowest $(18 - 20\%)$ mean portion of shared query results.

Table 5.2: Mean and standard deviation of the average portion of shared response documents of different models on a 2048 document corpus. One trial produced five real values, i.e. one portion per model. A portion is obtained from 10 randomly sampled query documents with 10 responses each (excluding the query document in the response). The sample selection is based on the same dataset for each trial and thus, query documents can be selected for multiple trials. There were 30 trials.

model 1	**model 2**	**mean**	**std**
Doc2Vec	USE	0.26	0.09
Doc2Vec	InferSent	0.26	0.07
Doc2Vec	TF-IDF	0.2	0.08
Doc2Vec	SBERT	0.18	0.06
USE	InferSent	0.33	0.08
USE	TF-IDF	0.25	0.08
USE	SBERT	0.25	0.06
InferSent	TF-IDF	0.26	0.08
InferSent	SBERT	0.24	0.06
TF-IDF	SBERT	0.18	0.05

5.6 Comparison with baseline topic analysis approach

The baseline topic analysis Top2Vec offers a variety of built-in functionalities to the user. It is possible to retrieve human interpretable inherent topics of a set of documents, as well as the topics most similar to certain search terms and word clouds of these results. Hence, this library meets the needs articulated by this work.

Opposed to Top2Vec, this thesis proposes a composite of different approaches to encoding visual and semantic information and query for them using a database. To be more specific, this thesis not only relies on one semantic embedding model but offers several techniques and an approach to incorporate visual information.

Moreover, the tool implemented in this thesis can display the term frequency of the document chosen in the detail component. The Top2Vec library does not offer a comparable service.

However, it is not possible to query for topics of the corpus which best describe a search term. Alternatively, one can perform a fuzzy text search on the documents. The user can inspect the PDF of a document upon clicking on its name in the list of documents. The detail view enables the investigation of similar documents in terms of different embedding approaches.

Due to Top2Vec's architecture, documents and words are mapped into the same VSM. Hence, the topic vector definition and representation by its closest words are more meaningful than the approach of the thesis. In this thesis, a topic is represented by frequent words in the set of documents that are not necessarily meaningful.

6 Conclusion

To conclude this thesis, the research questions are revised. The insights acquired by exploring different techniques with the goal of the exploration of large unstructured text data are discussed in Section 6.1. Then, Section 6.2 points out the scientific contributions.

6.1 Discussion

RQ1 seeks to discover whether visual embedding methods are suitable for the task of finding similar documents in large unstructured text corpora.

In this work, firstly, the images are preprocessed using the Eigendocs approach as discussed in Subsection 4.1.4 and Section 5.4. Then, the numerical data obtained from preprocessing the images is then reduced using PCA. Determining the number of components to be used for PCA in a meaningful scientific way was problematic (cf. Subsection 3.5.2 & Section 5.2): The cumulative explained variance did not indicate a small number of components to use. The RSME calculation was found to be unsuitable for the dataset since multiple random selections of documents from the dataset did not indicate a clear "elbow" point. The compressed images are clustered using the OPTICS algorithm and the `argmax` approach (cf. Section 5.5).

In general, the query responses from Section 5.5 which are based on visual information consist of visually similar documents. Hence, in terms of qualitative evaluation, the visual representation is considered to be valuable means to find visually similar documents in large unstructured corpora. Consequently, the answer to RQ1 is positive but acknowledges the lack of scientific justification for the proof of concept's configuration.

RQ2 aims to find out whether different embedding methods produce similar results. When comparing different semantic embedding methods in Section 5.5, slight differences between the models become evident. TF-IDF, Doc2Vec and SBERT are most dissimilar to each other. The TF-IDF approach performs rather poorly on unusual query documents such as handwritten ones.

The distinct response documents and their order are different for the same query document regarding different semantic embeddings. Different semantic embeddings can yield response documents containing the same company name. While the semantic responses' contents are considered similar to the query document and each other, the visual responses are more dissimilar from the query document and each other.

To answer RQ2, the content and visual appearance of the response documents of different embedding techniques is similar, but the actual documents in the response sets differ.

RQ3 poses the question of how the results of the system are presented to experts. The idea of the representation is to enable users to explore the data corpus. They should be able to query for terms, to find similar documents in the text corpus and to derive the inherent topics of the documents. The proof of concept is the implementation of a web interface which offers these services. The web interface is introduced in Section 4.2.

RQ4 addresses the evaluation of the performance of the system. In this work, the system is evaluated with respect to multiple parameters. Section 5.1 discusses the time to compute the different embeddings in Figure 5.1. In Section 5.3, some embedding models are evaluated with respect to their time consumption for different configurations (cf. Figure 5.3).

Another way to evaluate the performance of the system is to compare the results of different models via the intersection of their response sets (cf. Section 5.5). These intersections can be visualized with Venn diagrams (Figure 5.12) and heatmaps (Figure 5.13). To ensure the results are not random, the statistical properties of the response sets are calculated and presented in Table 5.2.

Additional findings which were obtained in the course of working on this thesis are presented in the following.

The InferSent and the TF-IDF model produce embeddings of large dimensionalities. Since changing the dimensionality would require retraining the models or risking the loss of quality, the dimensionality is not changed in this work.

The database Elasticsearch was chosen because it offers built-in functionalities, such as kNN and fuzzy text search. It is well-suited for flexible data since it is possible to insert incomplete documents into the database. However, the maximum dimensionality of the embeddings is limited to 2048. If the task at hand requires higher dimensional embeddings, such as the ones produced by the InferSent model, another database may be more suitable.

6.2 Contribution

In the context of this thesis, several ML techniques to derive semantic and visual information from unstructured data are explored.

In order to find relevant documents in a text corpus, the corpus is made searchable. This is achieved by constructing a pipeline that preprocesses the documents and stores them in a database in an offline fashion. The local Elasticsearch database stores 2048 randomly chosen documents from the whole dataset, whereas the database on the IES server stores around 497504 incomplete documents. Owing to Elasticsearch's implementation, (fuzzy) text queries and kNN search queries can be conducted with minimal latency (cf. Subsection 3.7.1).

Concerning RQ1, the visual embedding method Eigendocs is implemented. It is an adaption of the prevalent Eigenfaces approach to the task of finding similar documents. The idea of projecting items into a lower dimensional space is kept. The preprocessing is extended by placing the document images onto a white canvas as described in Subsection 4.1.2.

The semantic embedding methods TF-IDF, Doc2Vec, InferSent, USE and SBERT are explored. The configurations of the models are altered to reduce their runtime. Since the dimensionalities of TF-IDF and InferSent embeddings are too big to be stored in an Elasticsearch database, the dimensionality of the embeddings is reduced by the encoder of an AE (cf. Section 4.1.3).

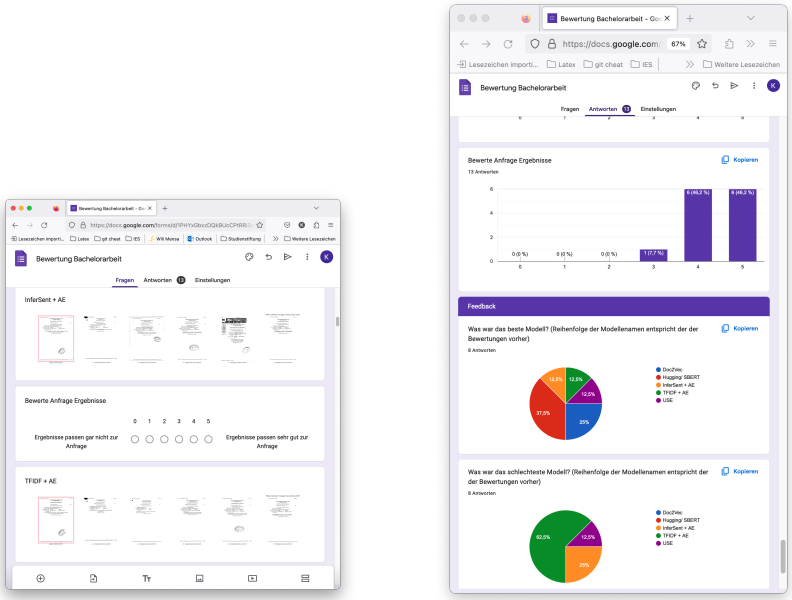
In the matter of RQ3, a web interface is implemented, which provides the possibility to conduct text queries and allows to examine a document of interest in more detail (cf. Section 4.2): The detail component not only contains a PDF viewer, word clouds of the most frequent words in the document or query response, but also an option to display the file names of the response documents for different embeddings. The tool implemented in this work is not designed for a productive environment since the focus is on the comparison of different models rather than usability.

Since the dataset is not labeled, the evaluation of the results is not trivial. Therefore, with regard to RQ2 and RQ4 multiple evaluation methods are implemented (cf. Section 5.5). The first method is a Venn diagram that depicts the intersection of the query responses of the power set of different embedding models. The second method is a heatmap that illustrates the average portion of shared response documents between different embedding models. Moreover, the mean and standard deviation of the portion of shared response documents are calculated to further investigate the distribution of the results obtained above. Furthermore, the time consumption of the computation of embeddings for different models is evaluated (cf. Section 5.1 & Section 5.3). Lastly, in Subsection 3.4.1, the tool is compared to a baseline topic analysis approach called Top2Vec.

7 Outlook

When investigating both semantic and visual embedding methods, differences between the models became evident. Overall, the textual embedding methods produced more meaningful responses than the visual embedding methods. However, this is not surprising since the textual embedding methods prioritize documents containing equal or semantically similar terms and thus, return documents of similar content or originating from the same company as the query document. Visual embedding methods, on the other hand, return visually similar documents.

It is complicated to compare the responses of semantic and visual embedding methods since they operate on fundamentally different data. A more thorough evaluation could include a survey. A selection of the results of this work is incorporated into the first approach to constructing a survey. 13 people with different academic backgrounds have partaken in the survey. A sample question and an illustrative result from the survey are displayed in Figure 7.1. However, constructing a survey is complicated since semantic similarities should be evaluated on a textual level, i.e. content, which is difficult for non-experts and not natural since humans are prone to assess similarities by visual inspection. Moreover, identification of the target audience is difficult since the target audience of the tool could be expanded to be more general than the tax office.



(a) A question of the survey. (b) Selection of results of survey.

Figure 7.1: A first survey approach from [30].

Similar to Pennington et al.'s work, in this thesis, for many models used, any unspecified parameters are set to their default values, assuming that they are close to optimal acknowledging that this simplification should be revised in a more thorough analysis.

The TF-IDF approach performs rather poorly on unusual query documents. There are multiple factors that could have contributed to this result. Firstly, the vocabulary is drastically reduced to satisfy the database's constraints concerning dense vector dimensionality. Thus, TF-IDF may either be unsuitable for the task of finding similar documents when the vocabulary size is restricted or further research is required to find more suitable means to compress the embedding before inserting it into the database. Secondly, the evaluation of the different preprocessors of TF-IDF is carried out on small datasets consisting of 195 and 2048 documents. This dataset may not be representative of the whole corpus.

In this work, the precomputed GloVe embeddings are replaced by a custom Word2Vec model. However, Pennington et al. state that GloVe outperforms Word2Vec on the same corpus, vocabulary and window size in terms of quality [55]. Hence, the quality of InferSent might have deteriorated due to the replacement of GloVe by Word2Vec.

When preprocessing the document images in the Eigendocs approach, the images are placed on a white canvas assuming its dimensions are bigger or equal to all other documents in the corpus. Since this assumption was not true, the images selected to find the dimensionalities of the canvas were not representative. Future work should include a more thorough analysis of the maximal image sizes in the corpus.

The parameter selection for PCA is not representative of the whole dataset, due to the fact that the dataset used for calculating the reconstruction error is too small. Moreover, the resulting plot is not optimal for conducting the "elbow method", since no significant change in the slope is evident.

Different AE architectures are experimentally evaluated on a selection of 195 documents. Since the dataset is too small and not drawn randomly from the whole data corpus the results are not representative. Thus, future work should include a more elaborate evaluation of different AE architectures on a bigger document corpus.

The comparison of the different embedding methods in terms of query response similarity was carried out on the data which was stored in the database. For future work, the comparison should be carried out on a separate dataset to evaluate the performance of the models on unseen data.

The evaluation of the similarity between query results of different models so far has not considered the individual weights for respective query responses because it was difficult to find means to interpret and visualize semantic meaningful weight relationships. Hence, future work could include the weights of the query responses in the evaluation.

Moreover, the similarity of the query documents is not considered in the evaluation. To further improve the evaluation, the number of occurrences of query documents in the

response documents of other queries could be examined. Another approach to evaluation could be to assess the quality of the images which were returned by multiple models. Possibly, one could create a hypothesis about whether better responses correlate with the number of models that returned them.

The elastic stack offers a wide range of tools, for instance, Kibana that can be used to manage models and to create ingest pipelines to embed new documents. If models are managed by Kibana, the models no longer have to be managed by the user and thus, the system would most likely be more user-friendly and less prone to errors.

Another issue is the fact that the database contains neither all embeddings nor all documents. The Bahamas leak contains 38 GB of data. Even though multiprocessing using Pool is used to split the workload across up to 100 processes, the embedding process is not finished after several days. Hence, more advanced coding techniques have to be applied to speed up the embedding process.

The domain of financial fraud and tax evasion is very interesting. Thus, future work could include the development of a working system for the tax office based on the system implemented in this thesis. The techniques explored in this work could be used to find similar documents to a query document and thus, facilitate initial exploration of a large data corpus.

Bibliography

[1] Slurm: Quick start user guide. URL `https://slurm.schedmd.com/quickstart.html`. [Accessed 16.09.2023].

[2] 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.

[3] Rubayyi Alghamdi and Khalid Alfalqi. A survey of topic modeling in text mining. *International Journal of Advanced Computer Science and Applications*, 6:147–153, 2015.

[4] Dimo Angelov. Top2vec: Distributed representations of topics. *arXiv:2008.09470*, 2020.

[5] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, 28:49–60, 1999.

[6] Farzana Anowar, Samira Sadaoui, and Bassant Selim. Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne). *Computer Science Review*, 40(100370):1–13, 2021.

[7] Fankar Armash Aslam, Hawa Nabeel Mohammed, Jummal Musab Mohd. Munir, and Murade Aaraf Gulamgaus. Efficient way of web development using python and flask. *International Journal of Advanced Research in Computer Science*, 6:54–57, 2015.

[8] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Sebastopol, CA, USA, 1st edition, 2009.

[9] 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. *arXiv:1803.11175*, pages 1–7, 2018.

[10] Allison Chaney and David Blei. Visualizing topic models. *International AAAI Conference on Web and Social Media*, 6:419–422, 2021.

[11] Delphine Charlet and Géraldine Damnati. Simbow at semeval-2017 task 3: Soft-cosine semantic similarity between questions for community question answering. *Orange Labs*, pages 315–319, 2017.

[12] Qiuxing Chen, Lixiu Yao, and Jie Yang. Short text classification based on lda topic model. In *International Conference on Audio, Language and Image Processing (ICALIP)*, pages 749–753, 2016.

[13] Rob Churchill and Lisa Singh. The evolution of topic modeling. *ACM Comput. Surv.*, 54:1–35, 2022.

[14] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv:1705.02364*, 2018.

[15] Matt Copperwaite and Charles Leifer. *Learning Flask Framework*. Packt Publishing, 2015.

[16] Laura Dayton, Dante Rousseve, Neil Sehgal, and Sindura Sriram. Final project report: Methods of facial recognition. *CSCI*, 2020.

[17] Z. Deng, Y. Hu, M. Zhu, and et al. A scalable and fast optics for clustering trajectory big data. *Cluster Computing*, 18:549—562, 2014.

[18] download-infersent. Infersent. URL `https://github.com/facebookresearch/InferSent`. [Accessed 14.11.2023].

[19] Elasticsearch-guide. Elasticsearch guide. URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html`. [Accessed 15.09.2023].

[20] Elasticsearch-kNN-embedding. 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] Fabio. Deep averaging network.ipynb. URL `https://github.com/f0bs/Machine_Learning/blob/master/Deep%20Averaging%20Network.ipynb4`. [Accessed 04.10.2023].

[22] Daniel Gaspar and Jack Stouffer. *Mastering Flask Web Development: Build Enterprise-Grade, Scalable Python Web Applications*, volume 2. Packt Publishing, 2018.

[23] gensim-doc2vec-init. gensim.models.doc2vec. URL `https://tedboy.github.io/nlps/generated/generated/gensim.models.Doc2Vec.__init__.html`. [Accessed 01.10.2023].

[24] gensim-word2vec-init. Word2vec embeddings. URL `https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec`. [Accessed 01.10.2023].

[25] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 2018.

[26] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv:2203.05794*, 2022.

[27] Machine Learning Group. Credit card fraud detection, 2017. URL `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`. [Accessed 18.11.2023].

[28] Stanford NLP Group. The stanford natural language inference (snli) corpus. URL `https://nlp.stanford.edu/projects/snli/`. [Accessed 18.12.2023].

[29] Christian M. Gruhl. *Novelty Detection for Multivariate Data Streams with Probalistic Models*. Kassel university press, 2022.

[30] Klara M. Gutekunst. Empirische bewertung von suchanfragen. URL `https://forms.gle/EU8UUxnWc7hWBngj8`. [Accessed 20.11.2023].

[31] Klara M. Gutekunst. Identifying fiscal fraud with anomaly detection techniques. Technical report, University of Kassel, 2023. URL `https://github.com/KlaraGtknst/identifying-fiscal-fraud`.

[32] Tom Hanika. Artificial intelligence. Technical report, 2023. Lecture script.

[33] impl-src-ae. Image compression using autoencoders in keras. URL `https://blog.paperspace.com/autoencoder-image-compression-keras/`. [Accessed 06.11.2023].

[34] Dan Jurafsky and James H. Martin. *Speech and Language Processing*, volume 3. 2023.

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

[36] Pooja Kherwa and Poonam Bansal. Topic modeling: A comprehensive review. *EAI Endorsed Transactions on Scalable Information Systems*, 7:1–16, 2019.

[37] Gunjan Khosla, Navin Rajpal, and Jasvinder Singh. Evaluation of euclidean and manhanttan metrics in content based image retrieval system. In *International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 12–18, 2015.

[38] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. *PMLR*, 37:957–966, 2015.

[39] Tzu-Hsuan Lin and Jehn-Ruey Jiang. Credit card fraud detection with autoencoder and probabilistic random forest. *Mathematics*, 9:2683–2699, 2021.

[40] Tzu-Hsuan Lin and Jehn-Ruey Jiang. Credit card fraud detection with autoencoder and probabilistic random forest. *Mathematics*, 9(21), 2021. URL `https://www.mdpi.com/2227-7390/9/21/2683`.

[41] Fredrik Lundh, Jeffrey A. Clark, and Contributors. The image class. URL `https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.convert`. [Accessed 14.12.2023].

[42] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv:1603.09320*, pages 1–13, 2018.

[43] Tomas Mikolov and Quoc Le. Distributed representations of sentences and documents. *JMLR*, 32:1–9, 2014.

[44] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, pages 1–12, 2013.

[45] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv:1310.4546*, 2013.

[46] Sumit Misra, Soumyadeep Thakur, Manosij Ghosh, and Sanjoy Kumar Saha. An autoencoder based model for detecting fraudulent credit card transaction. *Procedia Computer Science*, 167:254–262, 2020. International Conference on Computational Intelligence and Data Science.

[47] Sumit Misra, Soumyadeep Thakur, Manosij Ghosh, and Sanjoy Kumar Saha. An autoencoder based model for detecting fraudulent credit card transaction. *Procedia Computer Science*, pages 254–262, 2020.

[48] Giulia Moschini, Régis Houssou, Jérôme Bovay, and Stephan Robert-Nicoud. Anomaly and fraud detection in credit card transactions using the arima model. *Engineering Proceedings*, 5:56–67, 2021.

[49] 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].

[50] Mohammad Robihul Mufid, Arif Basofi, M. Udin Harun Al Rasyid, Indhi Farhandika Rochimansyah, and Abdul rokhim. Design an mvc model using python for flask framework development. In *International Electronics Symposium (IES)*, pages 214–219, 2019.

[51] Andreas Müller. word cloud. URL `https://github.com/amueller/word_cloud/tree/main`. [Accessed 05.10.2023].

[52] Li-Qiang Niu and Xin-Yu Dai. Topic2vec: Learning distributed representations of topics. *International Conference on Asian Language Processing (IALP)*, pages 193–196, 2015.

[53] nltk-lemma-wordnet. nltk.corpus.reader.wordnet module. URL `https://www.nltk.org/api/nltk.corpus.reader.wordnet.html`. [Accessed 27.10.2023].

[54] 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 *High Performance Computing, Networking, Storage and Analysis*. ACIM, 2013.

[55] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. *EMNLP*, page 1532–1543, 2014.

[56] 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. *AQTR*, pages 1–6, 2020.

[57] Nils Reimers and Iryna Gurevych. sentence-transformers/paraphrase-minilm-l6-v2. URL `https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2#sentence-transformersparaphrase-minilm-l6-v2`. [Accessed 04.10.2023].

[58] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv:1908.10084*, 2019.

[59] Larry R.MEDSKER and L. C. JAIN. *Recurrent neural networks. Design and Applications*, volume 5. 2001.

[60] Yusuf Sahin and Ekrem Duman. Detecting credit card fraud by decision trees and support vector machines. *World Congress on Engineering*, 2188:442–447, 2012.

[61] Shyam Seshadri. *Angular Up and Running: Learning Angular, Step by Step*. O'Reilly Media, Inc., 2018.

[62] Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, 18:491–504, 2014.

[63] Gerd Stumme and Robert Jäschke. Internet-suchmaschinen. Technical report, 2011. Lecture script.

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

[65] Yichuan Tang and Xuan Choo. Intrinsic divergence for facial recognition. *Centre for Theoretical Neuroscience*, pages 1–17, 2008. Paper.

[66] tfidf-scikit-learn. Tf–idf term weighting. URL `https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction`. [Accessed 29.09.2023].

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

[68] UniversalSentEnc-dev. universal-sentence-encoder. URL `https://tfhub.dev/google/universal-sentence-encoder/4`. [Accessed 04.10.2023].

[69] Ike Vayansky and Sathish A.P. Kumar. A review of topic modeling methods. *Information Systems*, 94(101582):1–15, 2020.

[70] A. Voit, A. Stankus, S. Magomedov, and I. Ivanova. Big data processing for full-text search and visualization with elasticsearch. *IJACSA*, 8:1–8, 2017.

[71] Chang Wang and Sridhar Mahadevan. Multiscale manifold learning. *AAAI Conference on Artificial Intelligence*, 27:912–918, 2013.

[72] Ziqiang Wang and Xu Qian. Text categorization based on lda and svm. In *International Conference on Computer Science and Software Engineering*, pages 674–677, 2008.

[73] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[74] V. Zamfir, M. Carabas, C. Carabas, and N. Tapus. Systems monitoring and big data analysis using the elasticsearch system. *International Conference on Control Systems and Computer Science*, pages 188–193, 2019.

[75] Vladimir Zaslavsky and Anna Strizhak. Credit card fraud detection using self-organizing maps. *Information and Security*, 18:48–63, 2006.

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

[77] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. Tfidf, lsi and multi-word in information retrieval and text categorization. *IEEE International Conference on Systems, Man and Cybernetics*, pages 108–113, 2008.

[78] Radim Řehůřek. Doc2vec paragraph embeddings, 2022. URL `https://radimrehurek.com/gensim/models/doc2vec.html`. [Accessed 01.10.2023].

Eidesstattliche Erklärung

Hiermit erkläre ich, Klara Maximiliane Gutekunst, dass ich die vorliegende Arbeit mit dem Titel "Identification of Key Information with Topic Analysis on Large Unstructured Text Data" 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, 21. November 2023

Klara Maximiliane Gutekunst