UNI KASSEL VERSITÄT

Chair Intelligent Embedded Systems

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

BACHELORTHESIS

Department of Electrical Engineering and Computer Science
University of Kassel

Author Name:	Klara Maximiliane Gutekunst
Address:	*** REMOVED *** 34125 Kassel
Matriculation number:	*** REMOVED ***
E-Mail:	klara.gutekunst@student.uni-kassel.de
Department:	Chair Intelligent Embedded Systems
Examining board 1:	Prof. Dr. rer. nat. Bernhard Sick
Examining board 2:	Prof. Dr. Gerd Stumme
Supervisor:	Dr. Christian Gruhl
Date:	November 7, 2023

Abstract

Finding relevant documents and interconnections becomes significantly more difficult due to the sheer amount of documents available. Institutes, such as German tax offices, have access to leak data, for instance, the Bahama leak, containing huge amounts of documents and valuable information yet to be extracted. However, these institutes, companies and individuals do not have sufficient resources to explore individual documents in order to find a specific one or to identify inherent key topics. Hence, computational means, such as text mining or topic modeling, may help to overcome this obstacle. This thesis proposes an approach to finding relevant documents which share common topics from a large text corpus.

Contents

Abstract ii

Contents iii

List of abbreviations v

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research Questions . . . 2
- 1.3 Structure of the Thesis . . . 3

2 Related work 4

3 Fundamentals 6
- 3.1 Preprocessing . . . 6
  - 3.1.1 Tokenization . . . 7
  - 3.1.2 Stemming . . . 7
  - 3.1.3 Lemmatization . . . 7
  - 3.1.4 Stop-Word-Removal . . . 7
- 3.2 Embeddings . . . 7
  - 3.2.1 Term Frequency - Inverse Document Frequency (TF-IDF) . . . 8
  - 3.2.2 Document to Vector (Doc2Vec) . . . 9
  - 3.2.3 Universal Sentence Encoder (USE) . . . 10
  - 3.2.4 InferSent . . . 12
  - 3.2.5 Hugging face's SBERT . . . 13
- 3.3 Similarity Measurement . . . 13
  - 3.3.1 Euclidian distance . . . 14
  - 3.3.2 Cosine Similarity . . . 14
- 3.4 Topic Modelling . . . 14
  - 3.4.1 Latent Dirichlet Allocation (LDA) . . . 15
  - 3.4.2 Top2Vec . . . 15
  - 3.4.3 Word Clouds . . . 16
- 3.5 Compression of data . . . 16
  - 3.5.1 AE . . . 17
  - 3.5.2 Eigenfaces . . . 17
- 3.6 Clustering . . . 20
  - 3.6.1 DBSCAN . . . 21

3.6.2 OPTICS . . . 22
3.7 Database Elasticsearch . . . 24
3.8 Flask . . . 27
3.9 Angular . . . 28

4 Implementation 29
4.1 Offline Processing . . . 29
4.1.1 Database . . . 29
4.1.2 Eigendocs . . . 33
4.1.3 Embeddings . . . 34
4.1.4 Clustering using OPTICS . . . 39
4.1.5 Topic Modeling . . . 41
4.1.6 Slurm . . . 43
4.2 User Interface . . . 44
4.2.1 Backend . . . 44
4.2.2 Frontend . . . 45
4.3 Trade-off between memory and query time . . . 47

5 Experimental evaluation 48
5.1 Database . . . 48
5.2 Eigendocs . . . 49
5.3 Embeddings . . . 50
5.4 Clustering using OPTICS . . . 53
5.5 Comparison of models . . . 55
5.6 Comparison with baseline topic modelling approach . . . 61

6 Conclusion & Outlook 63
6.1 Conclusion . . . 63
6.1.1 Discussion . . . 63
6.1.2 Contribution . . . 65
6.2 Outlook . . . 67

Bibliography v

List of abbreviations

CSS	Cascading Style Sheet
LDA	Latent Dirichlet Allocation
TF-IDF	Term Frequency - Inverse Document Frequency
TF	Term Frequency
IDF	Inverse Document Frequency
BERT	Bidirectional Encoder Representations from Transformers
BoW	Bag of Words
Doc2Vec	Document to Vector
Word2Vec	Word to Vector
Top2Vec	Topic to Vector
CBOW	Continuous-Bag-of-Words
GloVe	Global Vectors
USE	Universal Sentence Encoder
PCA	Principal Component Analysis
kNN	k-nearest neighbor
API	Application Programming Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheet
JSON	JavaScript Object Notation
CSV	Comma Separated Values
PKL	Pickle
HNSW	Hiercharical Navigable Small World
OPTICS	Ordering Points To Identify the Clustering Structure
AE	Autoencoder
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
HDBSCAN	Hiercharical DBSCAN
KL	Karhonen-Loéve
SVD	singular value decomposition
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
SQL	Structured Query Language
NoSQL	Not only SQL
ACID	Atomicity, Consistency, Isolation, Durability
VSM	Vector Space Model
NN	Neural Network

DNN	Deep Neural Network
RNN	Recurrent Neural Network
RMSE	Root Mean Square Error
ML	Machine Learning
NLP	Natural Language Processing
PVDM	Paragraph Vector Distributed Memory
PV-DBOW	Distributed Bag of Words
SNLI	Stanford Natural Language Inference
BiLSTM	bi-directional Long Short-Term Memory
LSTM	Long Short-Term Memory
DAN	Deep Averaging Network
SBERT	Sentence-BERT
RSME	Root Mean Square Error
LDA	Latent Dirichlet Allocation
UMAP	Uniform Manifold Approximation and Projection
UI	User Interface
SSH	Secure Socket Shell
IR	Information Retrieval
SVM	Support Vector Machine
IES	Intelligent Embedded Systems
VSCode	Visual Studio Code
CPU	Central Processing Unit
GPU	Graphics Processing Unit
GB	Gigabyte
PDF	Portable Document Format
PNG	Portable Network Graphics

1 Introduction

The Bahamas leak is a collection of roughly 38 Gigabyte (GB) documents, which were leaked in 2016 [48]. Tax offices examine the data to identify tax evasion. However, it has proven to be challenging to identify relevant documents and their interconnections due to the amount of documents appertaining to the leak. Therefore, the goal of this thesis is to support the investigators of the tax offices.

This thesis assumes that there are similar documents and that it is valuable to explore semantically or visually similar documents simultaneously. Hence, the analysis of the text corpus includes analysis in terms of textual information, i.e. content, and visual information, i.e. appearance and layout. The information gathered ought to be used to identify similarities between documents and group them. Topics of a document cluster do not have to be labeled specifically but should be interpretable.

This thesis proposes an approach to group documents based on their appearance or semantic similarity, which is defined via different embedding strategies. Both the embeddings and the visual information are computed offline and stored in a database. A query to this database returns the most similar documents to the query document based on the field specified. The embeddings are compared using cosine similarity, i.e. the angle between embedding vectors, whereas visual information is clustered using different approaches including Ordering Points To Identify the Clustering Structure (OPTICS) beforehand. To visualize the resulting groups of documents methods from topic modeling including Word Clouds are employed.

Besides literature research, application and evaluation of the methods identified, certain preprocessing methods have proven to be eminent to successful work with unstructured text data.

1.1 Motivation

On a broader scope, this thesis aims to provide computational means to facilitate the work with large unstructured text data for individuals. The goal is to actively use Machine Learning (ML) techniques to analyze a large text corpus and thus, 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 individuals exploring large unstructured text data, this thesis aims to provide computational means to facilitate the work with large unstructured text data. In this work, different methods to derive semantic and visual information from unstructured text data are applied. The techniques ought to be compared and evaluated.

In the following, the research questions addressed are defined:

How can exploration of unstructured texts be facilitated by ML techniques? Unstructured texts emerge in many different domains. Initially, this work is motivated by the task of tax evasion detection for tax offices. However, this work has a broad target audience.

Which methods are scalable to large datasets? The amount of information grows rapidly. Hence, techniques to explore information have to be scalable. This thesis should state obstacles and possibly limitations encountered when scaling the techniques to large datasets.

Do different embedding methods produce similar results? The task at hand defines a result as a set of similar documents to a query document. Hence, one has to compare response sets of different methods.

Do semantic or visual embedding methods produce more meaningful embeddings? The methods encode different types of information, i.e. text layer versus layout, into embeddings. This thesis should explore which type of information is more meaningful for the task at hand.

In order to facilitate the exploration of the data, a User Interface (UI) shall be provided. 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 random sampling, and thus, wants to find similar documents.

1.3 Structure of the Thesis

The rest of this thesis is structured as follows. Chapter 2 covers related work. Chapter 3 provides background information on the techniques applied in this thesis. Chapter 4 describes the implementation of the methods, while Chapter 5 evaluates them. Chapter 6 discusses the results, concludes this thesis and gives an outlook on future work.

2 Related work

Due to the persistent growth of data, topic modeling remains a significant domain in research. On a broader scale, researchers aim to provide means by which data can be grouped without manual inspection. Text mining is used for document retrieval, web search or spam filtering [55]. Text classification can be used for Information Retrieval (IR), and text categorization [76].

Tong and Zhang use their LDA topic modeling strategy on Wikipedia articles and users' tweets to search, explore and recommend articles as well as to analyze Twitter users' interests [65]. They note, that their work can be beneficial for social and business research.

Since working with huge amounts of textual data is a prevalent challenge in text-mining tasks, Mitra and Chaudhuri have published a survey on IR from documents in which they explore methods to extract information from different formats, including texts and images [46]. Bird et al. published a book covering the Natural Language Processing (NLP) fundamentals, including preprocessing techniques, using Python [8].

In order to assess similarity among items of a corpus of textual data, documents have to be represented in terms of real-valued vectors. Mikolov et al.'s work covers continuous vector representations of words from very large datasets [42]. Specific models used in this work include the Doc2Vec model, the Sentence-BERT (SBERT) model [57], the InferSent model [14] and the USE model. Both Radu et al. and Mikolov and Le evaluate the Doc2Vec model, arguing that it overcomes several of TF-IDF's shortcomings [55] and that it is superior to Bag of Words (BoW) models [41]. Cer et al. compare the strengths of different USE architectures [9].

In contrast to the assumption of this thesis that the prevalent topics are static, in reality, topics may be dynamic or change over time. Alghamdi and Alfalqi and Vayansky and Kumar have published surveys on common topic modeling techniques, including LDA and more advanced models which take into account factors such as time [3, 68].

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 as outlined by Sidorov et al. and Charlet and Damnati [60, 11] and the Euclidean norm [37]. Khosla et al. evaluate and compare different norms in the context of IR from images [37].

The usage of the clustering algorithm Density-Based Spatial Clustering of Applications with Noise (DBSCAN) on Doc2Vec embeddings was proposed by Radu et al. [55]. Other researchers, for instance, Kanagala and Krishnaiah, compare related clustering algorithms including DBSCAN and OPTICS. Initially, OPTICS was introduced by Ankerst et al. in 1999 as a method to analyze the density-based structure of data from a database [5]. Patwary et al., Deng et al. and Agrawal et al. propose OPTICS extensions, i.e. for spatially and temporally evolving data or a parallel version [53, 17, 2].

Since many algorithms suffer from the curse of dimensionality, Anowar et al. have proposed a survey on different dimensionality reduction techniques [6]. These techniques include Principal Component Analysis (PCA), LDA and singular value decomposition (SVD), which are well-known in the context of IR.

A specific approach in the domain of face recognition is adopted in this thesis. Eigenfaces projects face images into a lower-dimensional feature space which best encodes the variation among the faces [66]. Since 1991 this technique has been covered in a lot of papers [66, 75, 70, 63, 16, 62].

Due to the fact that online financial interactions become more popular, the crime rate in this domain has increased as well. Consequently, researchers have started developing anomaly detection techniques for financial data, such as credit card data [74, 47, 45, 39, 58, 33]. The work stated above and this thesis both concern the financial sector.

Some researchers have already developed complete topic modeling libraries. They merge a selection of the techniques stated above into a well-reasoned composite. Best-known examples include BERTopic, a composition of SBERT, a dimension reduction using Uniform Manifold Approximation and Projection (UMAP), Hiercharical DBSCAN (HDBSCAN) clustering and the application of TF-IDF on the clusters [32].

Other well-established topic modeling 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) [71]. Similarly, Chen et al. use a k-nearest neighbor (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 Topic to Vector (Top2Vec) model not only overcomes LDA's shortcomings [4, 51] but is also developed for topic modeling on a large collection of documents [4]. The Top2Vec library serves as a baseline model for the application developed in this work.

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 modeling 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. The database applied is described in Section 3.7. Finally, the libraries used to implement the web application are introduced in Section 3.8 and Section 3.9.

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 thus, should be omitted. Hence, preprocessing improves the performance and the results [55]. The next sections describe the preprocessing steps applied in this work. They are visualized in Figure 3.1 on an exemplary text.

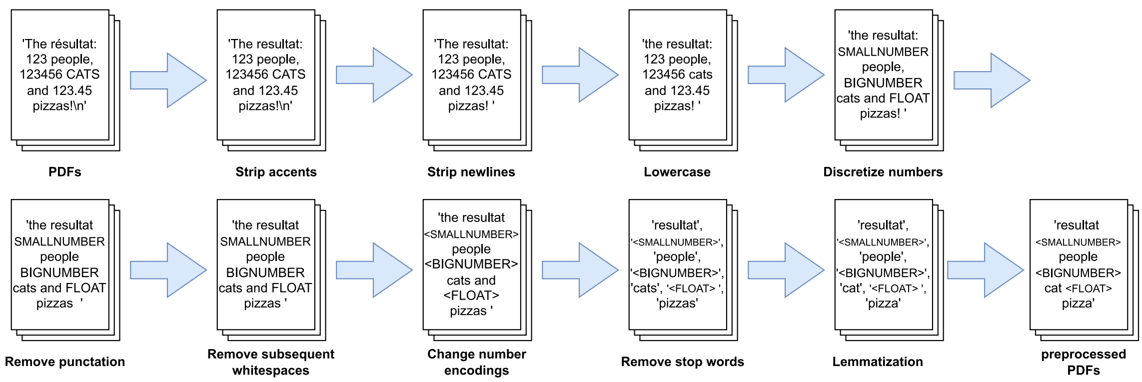


Figure 3.1: Preprocessing visualized using an example text.

3.1.1 Tokenization

Tokenization is the process of splitting a text into smaller pieces, so-called *tokens*. Tokens can be words and punctuation marks [8]. However, 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 [61].

3.1.2 Stemming

In order to avoid language inflections, i.e. treating words with similar meanings differently, stemming is applied [55]. According to Bird et al., *stemming* is the process of striping off any affixes, i.e. prefixes and suffixes [61], 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.

3.1.3 Lemmatization

Stemming and lemmatization are used to reduce the vocabulary size [55]. 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. *lemma*s, stored in a dictionary. Lemmatizers are usually slower than stemmers [8].

The *WordNetLemmatizer* from the `nltk` package requires a vocabulary. According to Radu et al., it is frequently used for lemmatization of English texts. It considers not only the meaning of words but also the order of the words [55].

3.1.4 Stop-Word-Removal

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 [61].

3.2 Embeddings

Usually, ML techniques require textual inputs to be converted to embeddings [41]. 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 VSM is a N-dimensional space [60]. Each dimension of a VSM corresponds to an index term, which is dependent on the embedding model. Every document embedding dimension explains the importance of the corresponding index term to the document. VSMs are commonly used due to their conceptual simplicity and because spatial proximity serves as a metaphor for semantic proximity [76, 9, 57, 4]. Representations in a VSM can improve the performance in NLP tasks [43]. According to Zhang et al., when representing text the first step is indexing, i.e. assigning indexing terms to the document. The second task is to assign weights to the terms that correspond to the importance of the term in the document. The weights assigned depend on the model.

The following section outlines the fundamentals of a selection of embeddings. Let a corpus of documents be denoted $D = \{d_1, d_2, ..., d_M\}$, the number of documents in the dataset $M = \|D\|$, and a sequence of terms w_{ij} or so-called document $d_i = \{w_{i1}, w_{i2}, ..., w_{iV}\}$, V being the length of the vocabulary, i.e. set of distinct words [55], and $j \in [0, V]$.

3.2.1 TF-IDF

TF-IDF provides a numerical representation of a word in a document [55]. It considers the frequency f_{w_{ij},d_i} of a word w_{ij} in a document d_i and the frequency of a word in the whole corpus. The frequency f_{w_{ij},d_i} is defined as $f_{w_{ij},d_i} = \frac{\#\text{occurrences of } w_{ij} \text{ in } d_i}{\#\text{words in } d_i}$.

TF-IDF is calculated as displayed in Equation 3.1 from [55]. The Term Frequency (TF) is determined utilizing $TF(w_{ij}, d_i) = f_{w_{ij},d_i}$, whereas the Inverse Document Frequency (IDF) is computed by $IDF(w_{ij}, D) = \log_2 \frac{M}{M_{ij}}$, M_{ij} being the number of documents the term w_{ij} appears in. IDF measures the importance of a term w_{ij} 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 [76]. 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 3.2.

$$TFIDF(w_{ij}, d_i, D) = TF(w_{ij}, d_i) \cdot IDF(w_{ij}, D) \tag{3.1}$$

TF-IDF has several drawbacks [55, 76]:

- TF-IDF does not consider semantic similarities between words.
- TF-IDF does not take into account the order of words in a document.

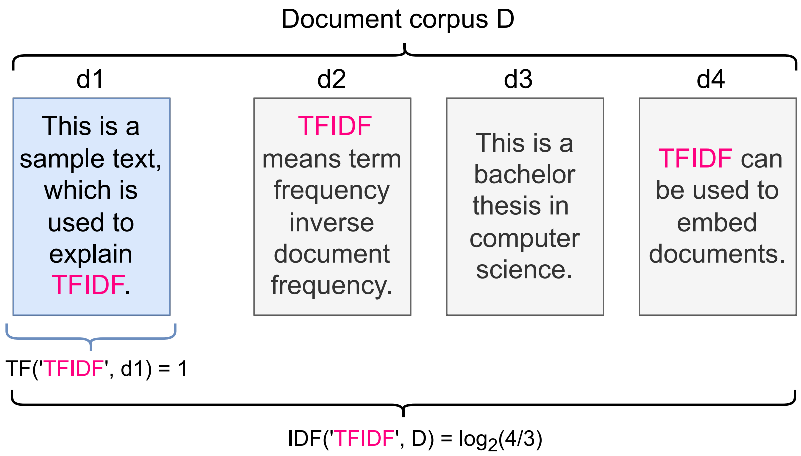


Figure 3.2: Exemplary calculation of TF and IDF for a document corpus D: TF only considers the documents of interest while IDF incorporates the importance of the word with respect to D.

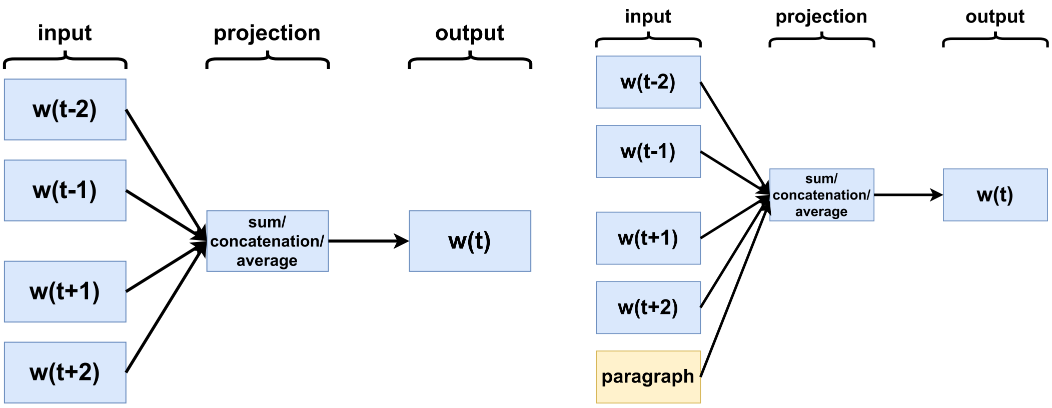
- TF-IDF often produces high dimensional representations which have to be postprocessed to reduce their dimensionality, e.g., by using PCA.
- The embeddings are not derived from a mathematical model of term distribution and thus, are occasionally criticised as not well reasoned.

3.2.2 Doc2Vec

Another term used for Doc2Vec is *Paragraph Vector* [55, 41]. Doc2Vec addresses the problems of TF-IDF by encoding texts as $N-$dimensional vectors learnt using the words' context [55]. Hence, it preserves semantic similarities between words and encodes linguistic regularities and patterns [43]. The model handles inputs of different dimensions and thus, tokens can be sentences, paragraphs or documents.

Doc2Vec is an adaption of the Word to Vector (Word2Vec) model, which maps words into a VSM [55]. Both approaches assume that words appearing in similar contexts are semantically similar. The Word2Vec embedding is obtained using a shallow Neural Network (NN), i.e. the NN has only one hidden layer. The embeddings are created by the hidden layer. There are two approaches to designing the architecture of the NN:

- Paragraph Vector Distributed Memory (PVDM): Predicts a word given a context [41, 42].
- Distributed Bag of Words (PV-DBOW): Predicts the context given a word [38, 43, 41].



(a) Continuous-Bag-of-Words (CBOW) architecture cf. [42].

(b) PVDM architecture cf. [41].

Figure 3.3: Both approaches predict the centre word using the context. PVDM is an adaption of CBOW to work on a set of documents or paragraphs instead of words.

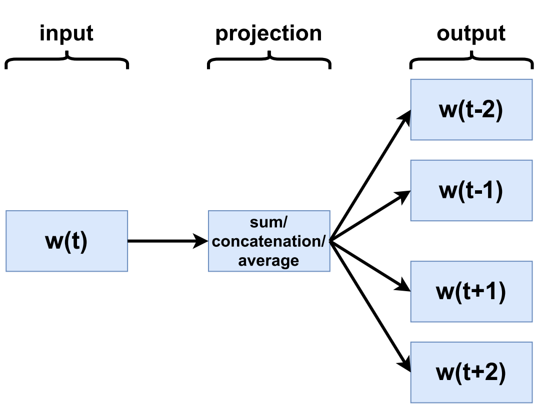
PVDM extends CBOW to work on a corpus of documents instead of on a set of words [55]: As usual, vectors representing the words are obtained using the CBOW model. CBOW has a single-layer architecture [54]. The word vectors can be concatenated, averaged or summed up [41]. Each document is mapped to a vector using an additional document-to-vector matrix. Both document and word vectors are initialized randomly, but trained to convey meaning in terms of semantic differentiation. In order to train the model, center words are predicted using the context. The context consists of words within a sliding window and their document, respectively represented as vectors [41]. The document vector is added to incorporate the document's topic and thus, acts like a memory [41, 4]. Mikolov and Le concatenate the document vector to the word vectors. The resulting vector is the prediction of the central word. The approaches are displayed in Figure 3.3.

According to Mikolov and Le, both CBOW and PVDM are trained using stochastic gradient descent and backpropagation. They also state that the document vectors are unique, while the word vectors are shared across the whole corpus.

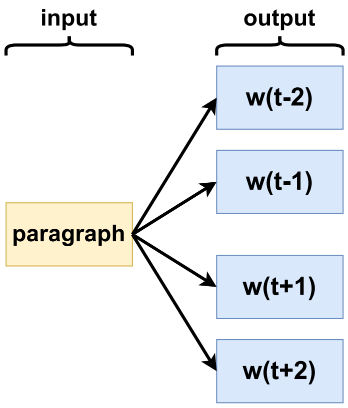
The PV-DBOW approach is the adaption of the Word2Vec algorithm Skip-Gram and predicts the context given the document [41]. The approaches are displayed in Figure 3.4.

3.2.3 USE

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



(a) Skip-gram architecture cf. [42].



(b) PV-DBOW architecture cf. [41].

Figure 3.4: Both approaches predict the context. PV-DBOW is an adaption of Skip-gram to work on a set of documents or paragraphs instead of words.

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. Those input vectors can be obtained from the text strings using models such as the BoW model [27]. The embeddings are averaged and subsequently passed to a feedforward Deep Neural Network (DNN) [9]. The architecture of the DAN model is depicted in Figure 3.5.

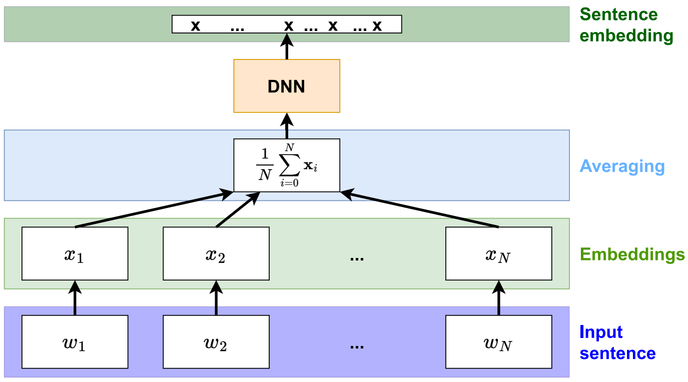


Figure 3.5: Architecture of the DAN model used for USE based on the textual description from [14]. The input words and bi-grams $(w_1, w_2, ..., w_N)$ are embedded. The embeddings are averaged and subsequently passed to a feedforward DNN, which produces a 512-dimensional sentence embedding.

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, 57].

3.2.4 InferSent

InferSent is a sentence embedding method trained in a supervised manner on the SNLI dataset [14, 57]. The trained model is transferable to other tasks.

Conneau et al. have compared multiple architectures in their work. The bi-directional Long Short-Term Memory (BiLSTM) architecture with max pooling was found to be the best option for the sentence encoder [14]. According to Reimers and Gurevych, it consists of a single siamese BiLSTM layer [57]. 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 Long Short-Term Memory (LSTM) respectively. Hence, the sentence is read from both directions and thus, considers past and future context. The architecture of the BiLSTM model with max pooling is depicted in Figure 3.6.

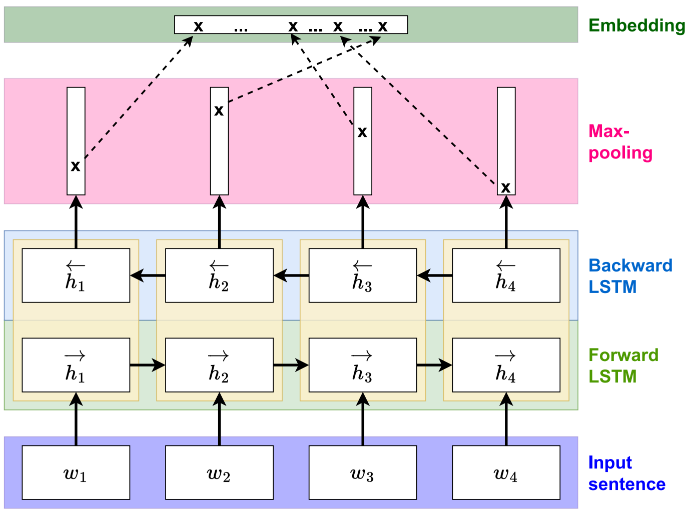


Figure 3.6: Architecture of the BiLSTM model with max pooling used for InferSent cf. [14]. The input sentence $(w_1, w_2, ..., w_T)$ is read from both directions by a forward and a backward LSTM producing $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ respectively. After concatenating $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ to h_t, max pooling is applied. The output is a fixed-sized embedding.

A LSTM is a Recurrent Neural Network (RNN) that is capable of learning long-term dependencies. 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. The max pooling layer selects the maximum value for each dimension of the hidden vectors.

3.2.5 Hugging face's SBERT

SBERT is an enhancement of Bidirectional Encoder Representations from Transformers (BERT). BERT is a pre-trained transformer network. It predicts a target value, for i.e. classification or regression tasks, based on two input sentences [57]. The input sentences are separated by a special token `[SEP]`. The base model applies multi-head attention over 12 transformer layers, whereas the large model applies multi-head attention over 24 transformer layers. The final label is derived from a regression function, which receives the output of the 12^{th} or 24^{th} layer, respectively. Reimers and Gurevych state that BERT is not suitable for specific pair regression tasks, since the number of input sentence combinations is too big. Another shortcoming of BERT is that it does not produce independent embeddings for single sentences. 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 [57].

SBERT provides fixed-sized embeddings for single sentences [57]. 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 3.7. 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.

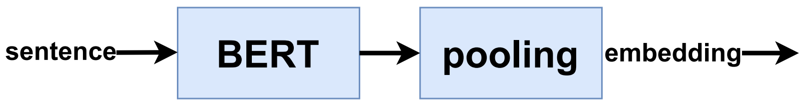


Figure 3.7: Architecture of SBERT cf. [57]. BERT is extended by a pooling layer. The input is a sentence and the output is a fixed-sized embedding.

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 [57]. Moreover, due to SBERT's transformer architecture, it is more computationally efficient than USE on Graphics Processing Units (GPUs).

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 [61, 37].

There are several similarity measures, such as the dot product quantifying the number of shared tokens of two texts, the (soft) cosine similarity [60, 11], which is the normalized dot product and calculates the angle between two vectors, and many more [61, 37, 57]. 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 points 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 points a, b is defined as $d_E(a,b) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}$ c.f. [37].

3.3.2 Cosine Similarity

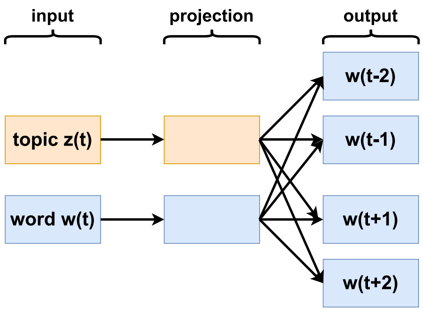
The similarity between two texts is measured by the cosine of the angle between their respective real-valued vectors. The cosine similarity is defined in Equation 3.2 from [60]. For positive vectors, for instance, produced by TF-IDF, it is a value between 0 and 1.

$$cosine(a,b) = \frac{a \cdot b}{\|a\| \times \|b\|} = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{3.2}$$

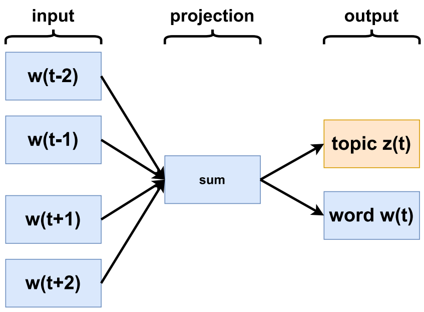
The formula from Equation 3.2 assumes that the vectors, which span the VSM are orthogonal and thus, completely independent. However, in practical applications, the index terms which span the VSM are often semantically dependent.

3.4 Topic Modelling

Since more and more textual data emerges, methods to analyze and extract information from texts become more important. One of these methods is topic modeling. A topic can be defined as a cluster of words that occur frequently. Subsequently, it is described by a probability distribution over a vocabulary. A document can be represented by one or more topics. Common topic modeling algorithms include LDA [3].



(a) Skip-gram architecture cf. [51].



(b) CBOW architecture cf. [51].

Figure 3.8: 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 $z(t)$.

3.4.1 LDA

LDA is a generative model, which recreates the original document word distributions with minimal error given the topics [3, 4]. A discrete multinomial probability distribution over a vocabulary consisting of W words is called a topic. A document is a mixture of K latent topics. Hence, each document from the document corpus D is represented by specific topic probabilities.

The probability distribution learnt by LDA only considers the statistical relationship of word occurrences in documents [51]. The content of a document can be described by the top N words with the highest conditional probability given a topic [51]. Due to the fact that LDA considers high probability words to be informative, words that occur frequently are deemed important even though they might be uninformative in reality [4, 51].

3.4.2 Top2Vec

The approach Top2Vec addresses several problems of state-of-the-art topic modeling approaches [4]. Opposed to LDA, 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. Moreover, it considers the semantic meaning of words unlike LDA. In contrast to LDA, Top2Vec only associates one topic with a document.

Top2Vec is based on Word2Vec and Doc2Vec. The documents are embedded using PV-DBOW. The two learning architectures CBOW and Skip-gram from Word2Vec are adapted to train the model as depicted in Figure 3.8 [51]. The Skip-Gram learning task is to predict the document a word came from [4, 51]. Similar to Doc2Vec, Top2Vec not only embeds words and documents in the same feature space but also topics [4, 51]. The similarity between embeddings can be measured using the cosine similarity function [51]. Angelov regards each point in the VSM as a topic, described by its nearest words.

Angelov states that topics are continuous and can be described by different sets of words [4]. Hence, topic modeling can be defined as the task of finding sets of informative words that describe a document. Documents in dense areas of the topic space are considered to be about the same topic. The density-based clustering algorithm HDBSCAN is used to find these dense areas. Since HDBSCAN has difficulties finding dense clusters in high-dimensional data, the dimensionality reduction method UMAP is applied [4]. The steps of the topic modeling procedure Top2Vec are depicted in Figure 3.9.

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.

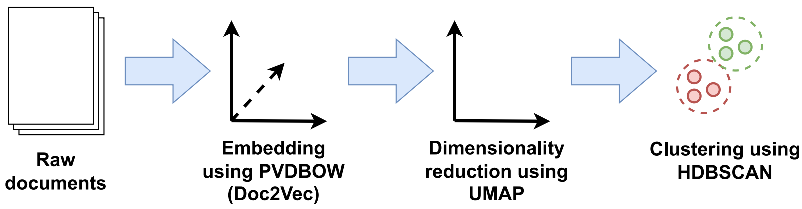


Figure 3.9: Procedure of topic modeling using Top2Vec.

3.4.3 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 that 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 AE

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) [44]. 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.

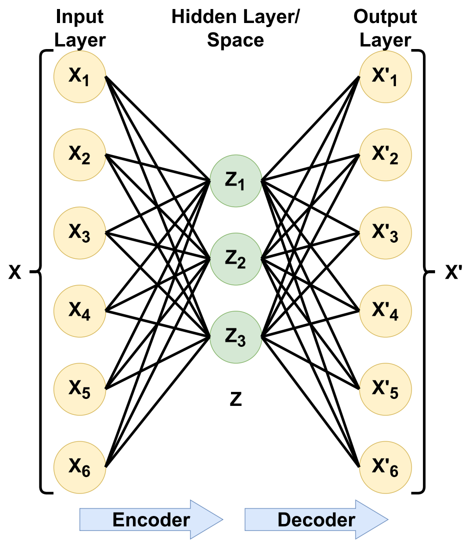


Figure 3.10: Structure of an AE cf. [44]

An undercomplete AE is a feed-forward NN, which consists of an encoder and a decoder. It learns efficient (non-correlated) encodings of the input data [44]. It is *undercomplete* because the dimensionality of the hidden layer, or so-called hidden space, is lower than the dimensionality of the input layer [34]. *Feed-forward* means that the information flows from the input layer to the output layer [34]. However, while training, the network employs backpropagation to update the parameters of the network [34].

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 [34]. The input and output layers have the same dimensionality.

3.5.2 Eigenfaces

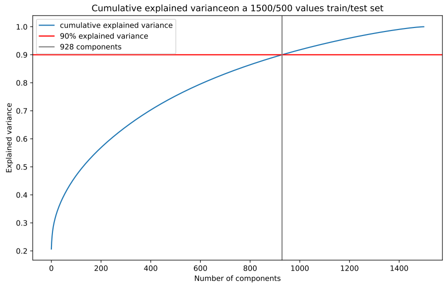
According to Turk and Pentland, the idea of Eigenfaces is inspired by information theory. Opposed to former approaches in the domain of face recognition which relied on the classification of images based on a set of predefined facial features, such as distance between eyes, Eigenfaces does not use predefined features [66]. 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 [66, 70]. These features do not necessarily correspond to human facial features [66]. Similar pictures, i.e. of the same person, should lie on a manifold in the lower-dimensional feature space [63]. The decomposition of input images not only reduces the complexity but also facilitates modeling probability density of a face image [63].

The greyscale input images are two-dimensional arrays of numbers: $\mathbf{x} = \{x_i, i \in \mathbf{S}\}$, $\mathbf{S}$ being a square lattice [75, 66]. 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 [75]. Some authors stress that the background is removed to omit values outside the face area [66]. In literature, typically, the original images' dimension is 512x512 [66]/ 64x64 [16], whereas the projected images' dimension is 16x16 [66]/ 250 [16]. Turk and Pentland stress that the data should be normalized, i.e. centered: $\Phi_k = \mathbf{x}_k - \psi$. Φ_k being the difference of the k-th training image and the average image $\psi = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k$, N being the number of training images.

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 as $\mathbf{x} = \sum_{i=1}^{n}\hat{x}_i\mathbf{e}_i$, $\mathbf{e}$ being an orthogonal basis [75]. If all basis vectors are used, the original image can be reconstructed using a linear combination of the basis vectors [66, 16]. The number of basis vectors is limited by the minimum of the training set size N [66] 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 is chosen such that $\hat{x}_i$ is small for $i \geq m$ [75]. 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 [66]. 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 [66].

In the context of Eigenfaces one basis used for decomposition is the Karhonen-Loéve (KL) basis, i.e. PCA [75, 66]. 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 $\mathbf{x}$ [75]. 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 [75, 16, 62]. The eigenvalues λ_i can be interpreted as the amount of variance explained by the corresponding eigenvector $\mathbf{e}_i$, which is equivalent to in-



(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. Around 13 and 30 is an "elbow" point.

Figure 3.11: Two approaches to determine the number of Eigenfaces m used to compress the input images.

formation 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 3.11 (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 3.3 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 3.11 (b).

$$\text{RSME} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - x_i')^2}{N}} \tag{3.3}$$

In order to reduce calculation complexity, C is approximated. Zhang et al. propose the approximation $\mathbf{C} \simeq \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k\mathbf{x}_k^T = \frac{1}{N}\mathbf{X}\mathbf{X}^T$, with $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^n$ [75].

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 $\mathbf{X}^T\mathbf{X}$ by $\mathbf{e}_i = \frac{1}{\sqrt{\lambda_i}}\mathbf{X}\mathbf{v}_i$ as discussed in more detail in [75]. Hence, the problem is reduced to a N by N matrix, which is computationally less expensive to solve, since $N \ll n$. Eigenvectors can be calculated using SVD [75]. 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 [66]. According to [66], 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, i.e. clusters [53, 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 [53]. 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 OPTICS [53].

- **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 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 [53]. 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 [53].
- **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 3.12 [35].

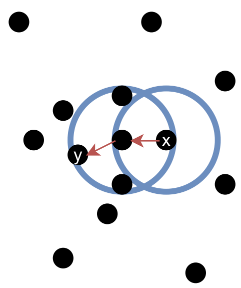


Figure 3.12: Density reachability cf. [5]. The object $y \in X$ is density reachable from $x \in X$, since it exists a chain of directly density reachable objects between x and y.

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 3.13.

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 bor-

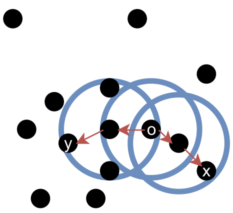


Figure 3.13: Density connectivity cf. [5]. The objects x and y are density connected since there is an object o, from which both x and y are density reachable.

der point is assigned to a cluster. The non-core point cluster assigning is non-deterministic [53]. 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 [53]. 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 [55]. Radu et al. found that their DBSCAN model assigned most objects noise when the dimensionality was 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 3.14.

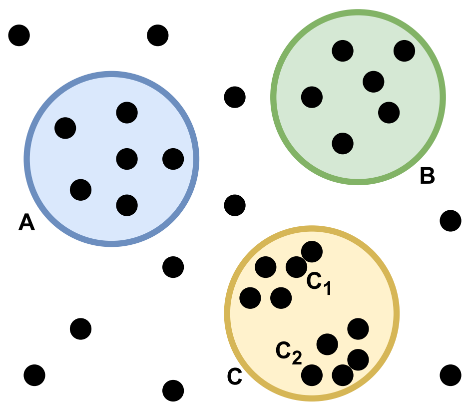


Figure 3.14: Clusters with different densities cf. [5]. Since C_1 and C_2 have different densities than A and B, a clustering algorithm using one global density parameter would detect the clusters A, B and C, rather than A, B, C_1 and C_2 .

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, 53]: 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 [53].

$$RD(y) = \begin{cases} \text{NULL} & \text{if } |N_\varepsilon(x)| < minPts \\ max(core\_dist(x), dist(x,y)) & \text{otherwise} \end{cases} \tag{3.4}$$

OPTICS saves the reachability distance $RD(y)$, as calculated in Equation 3.4 from [53], 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 [53]. If $\varepsilon < RD(y)$, then y is not density reachable from any of its predecessors and thus, one can determine whether two points are in the same cluster for the information saved by OPTICS [53, 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 [5]. 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 $\varepsilon < RD(y)$ [53]. The relationship between ε, cluster density and nested density-based clusters is displayed in Figure 3.15.

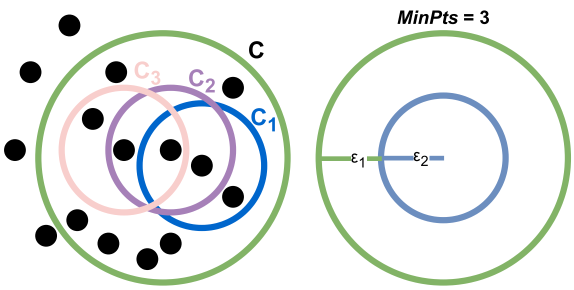


Figure 3.15: The relationship between ε, cluster density and nested density-based clusters cf. [5]. For a constant $minPts$, clusters with higher density such as C_1, C_2 and C_3, i.e. a low ε_2 value, are completely contained in lower density clusters such as C given $\varepsilon_1 > \varepsilon_2$. 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. According to [5], 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 Database Elasticsearch

Elasticsearch is a widely used non-relational database, which was designed to store and perform full-text search on a large corpus of unstructured data [69]. 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 [69, 73]. According to Zamfir et al., Elasticsearch provides Wikipedia's full-text search and suggestions as well as Github's code search and Stack Overflow's geolocation queries and related questions. It enables near real-time search by index refreshing periods of one second. Needless to say, Elasticsearch is qualified to handle Big Data.

Elasticsearch is a document store, which stores schemaless key-value pairs called documents [28]. 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 [73]. 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 [69]. 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 [73].

By specifying the unique `_id` of a document and the database `index`, it is possible to retrieve a specific document from the database using the `GET Application Programming Interface (API)`. The query is real-time by default. The parameters `_source_excludes` or `_source_includes` can be used to define the structure of the response [20].

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 [26]. The default preprocessing steps of the so-called default analyzer include tokenization and lowercasing [26]. Omitting stop words is disabled by default, but it is possible to provide custom stop words or use the English stop word list [26]. 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. When the user queries a specific phrase Elasticsearch expands the query to include synonyms of the query terms [25]. The maximum number of expansion terms is set to 50 by default but can be configured by the user [24]. By default, the multi-terms 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 [24].

Another search option of Elasticsearch is the kNN search. The return value of a kNN search is the `k` nearest neighbours in terms of a certain distance function of a query vector [40]. The query is a dense vector of 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 Hiercharical Navigable Small World (HNSW) algorithm [40, 21]. HNSW is a graph-based algorithm [40]. The term `navigable` refers to the graphs used, which are graphs with (poly-)logarithmic scaling of links traversed during greedy traversal concerning the network size [40]. The idea of a `hiercharical` algorithm is to create a multilayer graph, grouping links according to their link length, as displayed in Figure 3.16. The search starts on the uppermost layer, i.e. the layer containing the longest links, greedily traversing the layer until reaching the local minimum. It uses this local minimum as the starting point at the next lower layer and the process is repeated until the lowest layer is reached [40]. The layers of the graph are built incrementally, and a neighbour selection heuristic, as depicted in Figure 3.17, not only creates links between close elements, but also between isolated clusters to ensure global connectivity [40].

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 [21]. Elasticsearch's kNN implementation not only allows literal matching on search terms but also semantic search [21].

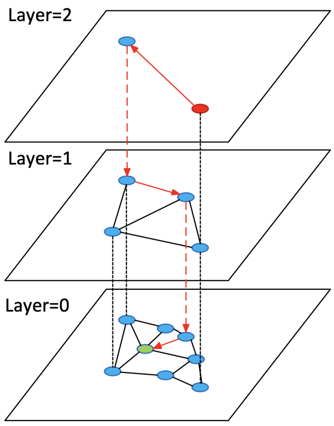


Figure 3.16: Structure of HNSW layers from [40]. The search starts on the uppermost layer, i.e. the layer containing the longest links, greedily traversing the layer until reaching the local minimum. The local minimum is used as the starting point at the next lower layer and the process is repeated until the lowest layer is reached.

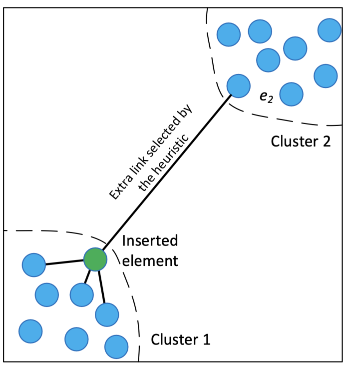


Figure 3.17: Neighbour selection heuristic of HNSW from [40]. The heuristic creates diverse links, i.e. links between close elements (e.g., green circle and elements in cluster 1) and between isolated clusters (e.g., green circle and e_2) to ensure global connectivity.

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 [22].

3.8 Flask

Flask is open source and written in Python by Armin Ronancher in 2004 [7, 49]. 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 Hypertext Transfer Protocol (HTTP) request parsing [15]. It is extensible and thus, 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 Hypertext Markup Language (HTML) pages, whereas static files such as Cascading Style Sheet (CSS) files are handeled 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 [49].

All requests received from clients are passed to an instance of the Flask application [31]. Hence, the first step is to create an instance of the Flask class shown in Listing 3.1.

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

Listing 3.1: Initialization of 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 [31]. 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: Exemplartary 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 [31]. By default, dynamic components are of type `string`. However, other types including `int` and `float` are supported.

During development, the Flask application can be run using `flask run` to start the built-in development web server [31]. By enabling debug mode, the server automatically reloads the application when changes are detected.

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` [28]. The `GET` method is used to retrieve data from the server, whereas the other methods are used to either insert, update or delete data.

3.9 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 service modules contain the logic of the application and communicate with the backend.

Angular applications are created using the `ng new NAME` command line interface [59]. This command creates a skeleton, which can be customized to meet the needs of the application. By running `ng serve` the application can be served locally.

4 Implementation

This chapter describes how the theoretical basics from Chapter 3 interplay and how they are used in this thesis. It is divided into Section 4.1, which outlines the steps carried out before the application is operative, Section 4.2, which is about the resulting application and Section 4.3, which discusses the dilemma faced when balancing memory usage and query time. Specific parameter choices are explained in Chapter 5.

4.1 Offline Processing

This section outlines implementation details of the data fed into the database, the database itself and the baseline topic modeling 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 a large unstructured corpus of Portable Document Format (PDF) files. After the initialization of the database, it is used for queries. Therefore, the workflow is completely offline.

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.

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 + 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`.

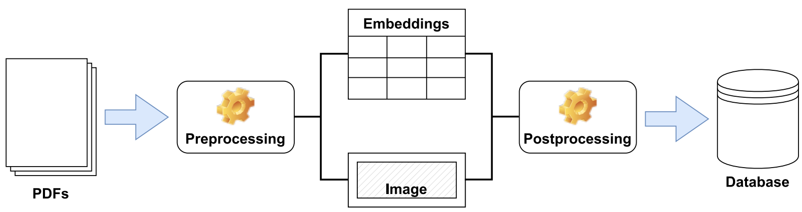


Figure 4.1: PDFs to Database. First, the data is preprocessed: The first page of a PDF file is converted to an image and the complete text is extracted. The images are stored in the database as well as the text and different embeddings of the text. Some values, such as the image or the InferSent embedding,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

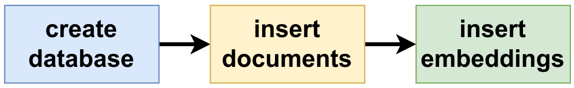


Figure 4.2: Procedure of initialization and filling of the database.

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, the data to be split among multiple processes is a set of paths to documents. The `Pool` object from the multiprocessing module is used for data parallelism. The steps carried out are displayed in Listing 4.1. First, the absolute paths of all documents are saved in a list. This list is divided in `num_cpus` many sublists of similar size. Each process works on a sublist. The embeddings are subsequentially inserted into the database for each sublist.

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

Listing 4.1: Usage of `Pool` for data parallelism. The paths to the documents to insert are divided 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.

```
client.update(index='bahamas', id=id, body={'doc':
    {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 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, which calculates the `_score` of a document according to Equation 4.1 from [23], where `query` is the query vector and `vector` is the vector representation of the document in the database. The other similarity measures provided by Elasticsearch are `l2_norm` or so-called Euclidian distance and `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.

$$\frac{1 + \mathrm{cosine}(\mathrm{query}, \mathrm{vector})}{2} \tag{4.1}$$

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, the normal kNN search on a field that contains an embedding is used.

4.1.2 Eigendocs

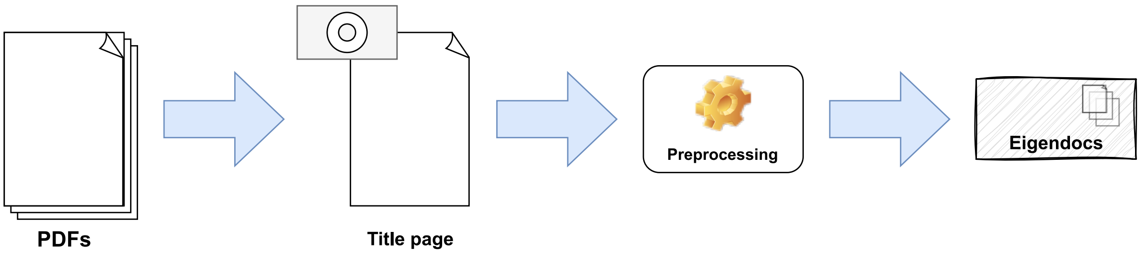


Figure 4.3: From PDFs to Eigendocs. Firstly, the first page of a document is converted to an image. Then, the image is preprocessed: It is placed on a white canvas, to ensure all images have the same dimensions. Moreover, it is converted to greyscale and normalized to values between zero and one. Afterwards,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

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 procedure of the Eigenfaces adaption *Eigendocs* is displayed in Figure 4.3.

The documents are first read from a directory. Subsequently, their first page is converted to an image and saved. When initially filling the database, these images are read from their directory. Firstly, the maximum height and width among all images in a corpus of 1000 randomly sampled images are calculated. These dimensions are used to create a white canvas for each image which forms the background. Every image is placed in the upper left corner. 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 dimension of the former image. Therefore, the relative size of images in the corpus is incorporated in the resulting representation of the input images. However, some images are bigger than the maximum values of the selected images and as a consequence are scaled.

```
0.299*img[:,:,0] + 0.587*img[:,:,1] + 0.114*img[:,:,2]
```

Listing 4.4: Conversion of RGB pixel values to greyscale from a script by Dr. Christian Gruhl.

Afterwards, the images are converted to greyscale using Listing 4.4 (Listing 4.5, line 2). Before returning the image, the two-dimensional image vectors are converted to one-

```
1 C = np.ones((max_w,max_h))
2 C[:doc.shape[0],:doc.shape[1]] = rgb2gray(doc)
3 documents.append(C.ravel())
```

Listing 4.5: Preprocessing of the input images from Dr. Christian Gruhl. The background is a white canvas. The images are converted to one-dimensional greyscale values.

dimensional ones as displayed in line 3 of Listing 4.5. The decomposition is transformed using PCA as displayed in Listing 4.6. The implementation of PCA from sklearn intrinsically normalizes the data as described in Subsection 3.5.2.

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

Listing 4.6: Initialization of the PCA instace used to compress the image data. Since the Eigenfaces approach uses SVD, the adaption Eigendocs has to be implemented likewise applying a `svd_solver`.

4.1.3 Embeddings

The models used to encode the textual data from the data corpus are outlined below with regard to implementation details.

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 4.8. The `TfidfVectorizer` is provided by the `scikit-learn` package. When initializing the model, the parameters define not only the input type but also the way the data is preprocessed. 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 [64]. An embedding is obtained using the command from Listing 4.7.

```
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 `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 [64].

By default, the `scikit-learn` implementation uses the `norm='l2'` parameter, i.e. the Euclidean norm [64]. 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 4.2 from [64]. The one is added to M_{ij} due to the parameter `smooth_id=True` by default to prevent zero divisions [64] and to avoid logarithmic divergences due to a zero argument [54]. After calculating the TF-IDF values, they are normalized by the Euclidean norm $v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2+v_2^2+...+v_M^2}}$.

$$\mathrm{idf}(w_{ij}) = \log \frac{1+M}{1+M_{ij}} + 1 \tag{4.2}$$

```python
tfidf = TfidfVectorizer(input='content',
            preprocessor=TfidfTextPreprocessor().transform, min_df=3,
            max_df=int(len(docs)*0.07))
tfidf.fit(documents)
```

Listing 4.8: 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.

In this work, the text of the PDFs is first extracted, then preprocessed using a custom preprocessor and afterwards embedded using the `TfidfVectorizer`. The TF-IDF weights are considered 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. The pipeline in Figure 4.4 visualizes these steps.

In this work, a custom preprocessing function is used. The preprocessing steps are visualized in Figure 3.1 on an exemplary text. 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

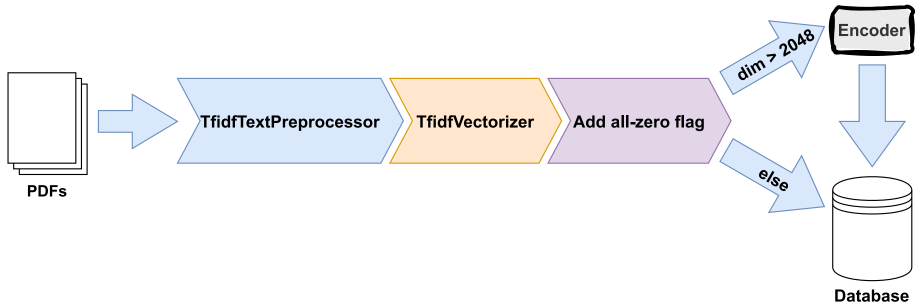


Figure 4.4: TF-IDF pipeline. Firstly, the text extracted from the documents is preprocessed using a custom preprocessor. Then, the TF-IDF values are obtained from the `TfidfVectorizer`. Afterwards, the all-zero flag is added to the TF-IDF weights. If the resulting dimensionality is bigger than 2048, the encoder of an AE is used to reduce the dimensionality. The results are stored in the database.

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. The stop word corpus consists of common English stop words. Afterwards, the tokens are lemmatized. The lemmatizer used is `WordNetLemmatizer` from the `nltk` package. `WordNetLemmatizer` uses the English lexical database `WordNet` to return valid stems [52] In the end, the tokens are joined to a string and returned.

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. The parameter `dm` determines the training algorithm used. The value `dm=1` specifies the PVDM algorithm, while `dm=0` specifies the PV-DBOW algorithm [77]. The default algorithm, i.e. PVDM, is used in this work [30]. 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 [30, 29]. 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 [30]. 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 [77]. Many Doc2Vec default values are adopted from Word2Vec since the `gensim` Doc2Vec implementation inherits from the Word2Vec implementation.

InferSent

The InferSent model is provided by PyTorch [57]. The parameters used to initialize the model are presented in Listing 4.9. The parameter `version` indicates whether the model was 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 4.9: Parameters of the InferSent model.

The steps necessary to create a working instance of the InferSent model are presented in Listing 4.10. 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 path to the word embeddings is set in line 3. Finally, in line 4, the vocabulary of the model is built or more precisely, only those embeddings needed are kept while the rest is discarded.

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

Listing 4.10: Initializing the InferSent model.

Initially, in this work, the InferSent model was based on GloVe word embeddings. GloVe is a global log-bilinear regression model for unsupervised learning of vector representations of words [54]. According to Pennington et al., the model captures global corpus statistics directly. The GloVe model is trained on ratios of co-occurrence probabilities of words from the corpus, which correlate with the relationship between the words. Pennington et al. introduce rules for a weighting function to ensure neither rare nor frequent co-occurrences are overweighted. It is possible to download embeddings computed by GloVe, instead of using the algorithm to generate them. The precomputed word embeddings are stored in a 5.65 GB text file. The file contains 840 B tokens and a vocabulary of 2.2 M cased 300-dimensional vector representations of words [18]. According to Cer et al., GloVe introduces bias in terms of ageism, racism and sexism into the 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. The only parameter which differs from the default settings is the `vector_size` which is set to 300. After the Word2Vec model is trained, the word embeddings are saved in a file. The file is post-processed to be compatible with the InferSent model. To be more precise, only lines that consist of at least two whitespace-separated char sequences are kept. Usually, word embeddings stored in a text file are structured in a way that the first char sequence is the word and the following numbers are the vector representation of the word.

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

USE

The USE model is provided by TensorFlow [57]. In this work, the fourth version of the model is used. The implementation from Tfhub uses the DAN architecture [67]. It is about 1 GB and thus, loading the model takes a while. It is not necessary to preprocess the data for the model [67].

SBERT

The SBERT model is provided by PyTorch [57]. An instance of the model is obtained as shown in Listing 4.11. The model contains a BERT transformer, which has a `max_seq_length` of 128. It does not convert inputs to lowercase by default [56]. 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 are `cls_token_pooling`, which returns the output of the first token, `max_pooling`, which returns the maximum value of the output vectors, and `pooling_mode_mean_sqrt_len_tokens`. The word embedding dimension is 384 by default [56].

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

Listing 4.11: Initialization of the SBERT model.

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 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.

The implementation was provided by a blog post [1]. It uses the library keras. The architecture is adapted to fulfil the needs of the specific context. It is presented in Figure 4.5. In this work, only the encoder is used.

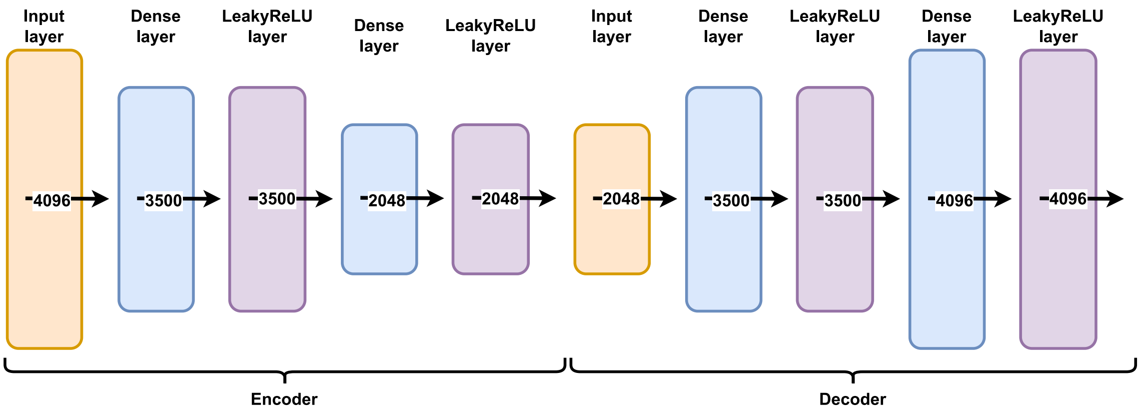


Figure 4.5: Architecture of the AE.

4.1.4 Clustering using OPTICS

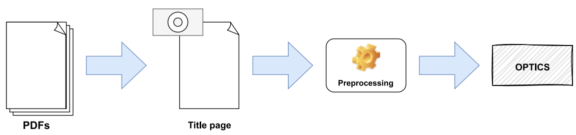


Figure 4.6: The first page of each document is converted to an image. The image is preprocessed, i.e. conversion to greyscale and resizing.

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 4.6. There were two different preprocessing approaches:

1. The images were first preprocessed to 32x32 normalized greyscale pixels (cf. [5]) as visualized in Figure 4.7 and afterwards compressed to 13-dimensional vectors using PCA.

2. The technique Eigendocs from Subsection 3.5.2 was used to compress the images to 13-dimensional normalized greyscale images as displayed in Figure 4.8.

TODO: better images

[1] https://blog.paperspace.com/autoencoder-image-compression-keras/ (last accessed: 06/11/2023)

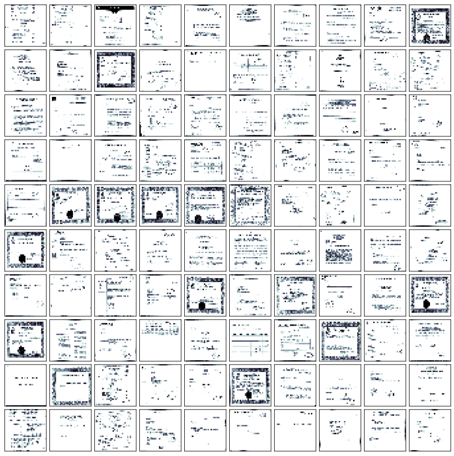
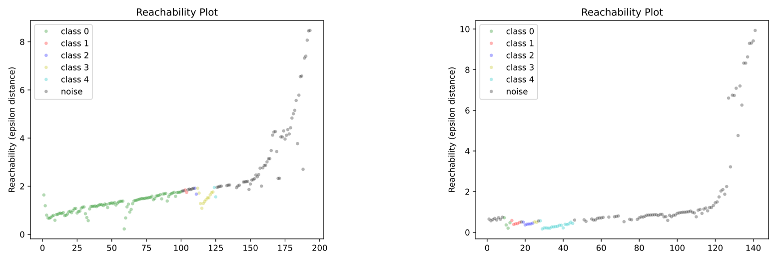


Figure 4.7: Preprocessing of 100 documents to 32x32 normalized greyscale pixels.

Figure 4.8: 10 randomly selected documents from the test set. 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.



(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. It shows the reachability distance of each document to its predecessor in the order list.

The reachability distance ordered by OPTICS is displayed in Figure 4.9.

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 cluster. The way the reachability plot is used to extract clusters is dependent on the `cluster_method`. One can choose either `dbscan` or `xi` 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. The code to initialize an exemplary OPTICS model is displayed in Listing 4.12.

```
optics_model = OPTICS(cluster_method='dbscan', min_samples=2, max_eps=10,
    eps=0.5)
```

Listing 4.12: Initialization of the OPTICS model. The minimum number of samples `min_samples` in a cluster corresponds to *minPts*.

4.1.5 Topic Modeling

Two topic modelling 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]. The non-changeable default values and settings of the model are listed below. The word and document embeddings are generated by the Doc2Vec version PV-DBOW. It has a window size of 15, uses hierarchical softmax, a minimum count of 50, a vector size of 300 and a sub-sampling threshold of 10^5. UMAP's hyperparameters are set to 15 nearest neighbours, cosine similarity as the distance metric and 5 as the embedding dimension in Angelov's work.

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

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

Listing 4.13: Initialization of the Top2Vec model.

Word Cloud

The implementation of Word Clouds in this thesis is based on the Python library *wordcloud* by Müller [50]. 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 4.14.. The `WordNetLemmatizer` from the `nltk` package is used to ensure the stemmed words exist.

A Word Cloud is initialized as shown in Listing 4.15.

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

Listing 4.14: Custom preprocessing of Word Cloud input.

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

Listing 4.15: 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, 72].

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 4.10.

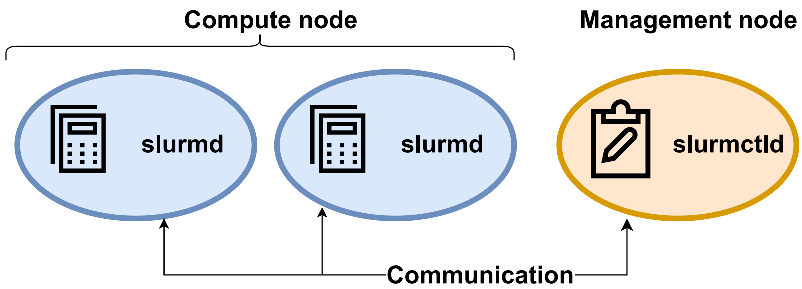


Figure 4.10: Slurm architecture. The management node has a `slurmctld` daemon, while every compute node has a `slurmd` daemon. The nodes communicate. The user can use certain commands, for instance `srun` and `squeue`, anywhere on the cluster.

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 Central Processing Units (CPUs) simultaneously to divide the workload of a task. In this work, multiple `sbatch` scripts were used to carry out a variety of tasks. A summary of the tasks and scripts is given in Table 4.2.

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 Secure Socket Shell (SSH) tunnel connection from a local Visual Studio Code (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 Portable Network Graphics (PNG) version of the first page of the PDFs.

4.2 User Interface

A basic UI 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 UI. The UI is divided into a frontend and a backend.

4.2.1 Backend

The framework used for the backend is Flask. In this work, only the `GET` method is used. 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 query can be of type `match_all`, which returns all documents in the database, or a fuzzy full-text query, or a kNN query on a certain field of the database. Moreover, the number and start index of the results returned can be specified.
- Document: Returns the document with the specified `id`.
- PDF: Returns the path to a PDF file. In order to access the path information a query for a document with the specified `id` is performed.

- WordCloud: Returns the bytes of a WordCloud image. Depending on additional parameters, the WordCloud is either generated from one document or a group of similar documents.

- Term Frequency: Returns the term frequency calculated for the specified document.

- TopicWordCloud: Returns a Word Cloud of the terms that describe the topics most similar to query term. The topics are generated by Top2Vec.

- Topics: Returns the topics generated by Top2Vec. The topics are described by the words closest to the topic vectors.

In order to test the endpoints during development, a swagger documentation for every endpoint is provided.

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 4.11.

- 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 upon pressing 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 4.12.

- 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 4.13.

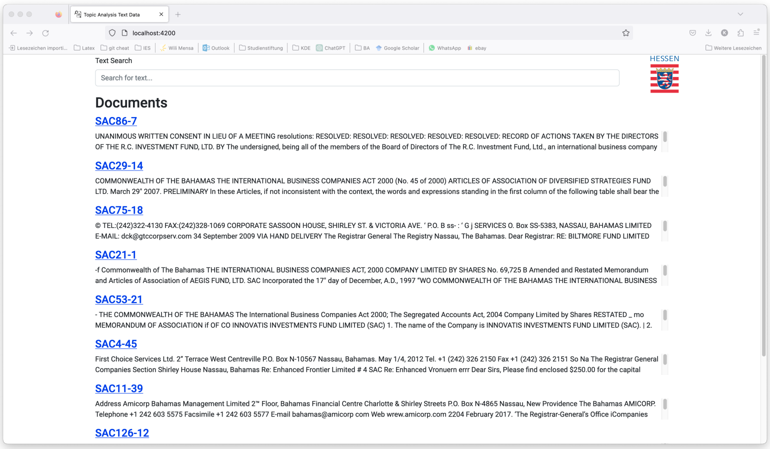


Figure 4.11: Home component of the frontend. The search bar is used to enter the text query. The results of the query are displayed below the search bar.

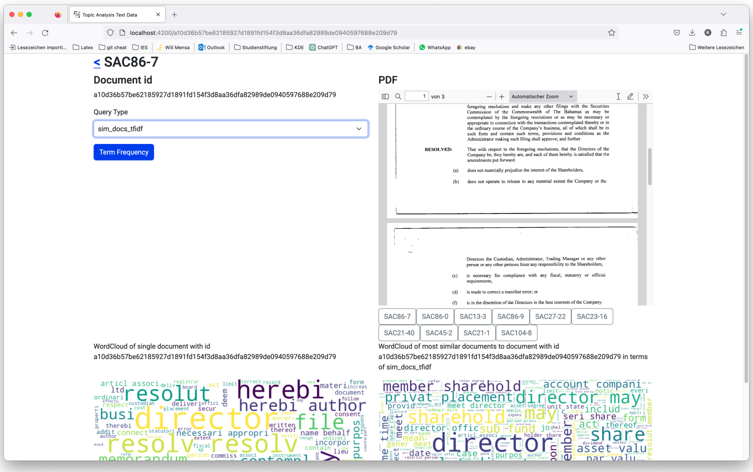


Figure 4.12: Detail component of the frontend. The chosen document is displayed, as well as its most similar documents in the database. Word Clouds of the document and the most similar documents are displayed.

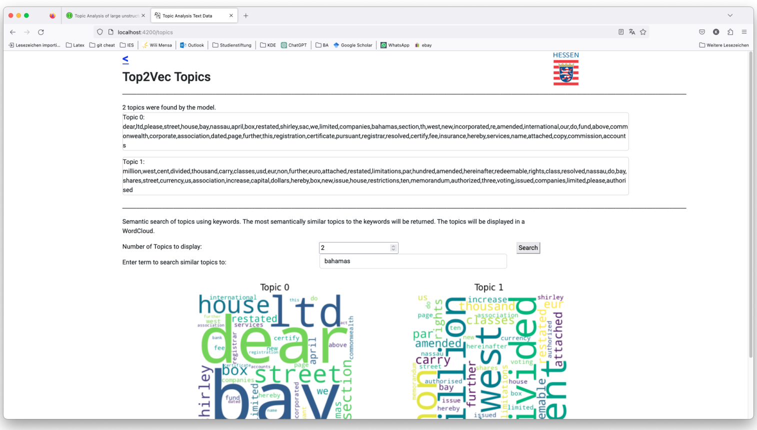


Figure 4.13: Topic component of the frontend. The topics identified by Top2Vec are listed. Below them, the user can query for the most similar topics to a term. The results are displayed as a Word Cloud.

To change between the components, the routes have to be defined. The routes are defined in the `app-routing.module.ts` file, as shown in 4.16.

```
const routes: Routes = [
    { path: 'topics', component: TopicsComponent},
    { path: ':id', component: DocumentDetailComponent},
    { path: '', component: HomeComponent},
  ];
```

Listing 4.16: Definition of routes in Angular in the `app-routing.module.ts`.

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. The decision to build a tool which was trained once offline seemed to be the only option facing the amount of data. Since Elasticsearch provides fast queries, it is sufficient to rely on real-time queries.

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 as 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 was omitted.

5 Experimental 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.

5.1 Database

There is a variety of parameter values to choose from when working with databases and embeddings. These parameters include similarity metrics and the choice of query types.

Similarity measurements

According to Reimers and Gurevych, the similarity measurements discussed in Section 3.3 obtained roughly the same results in their experiments [57].

As the similarity between vectors is usually calculated using some form of cosine similarity, rather than Euclidean distance in literature, cosine is preferred 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 [31]. They do not require a predefined schema and can therefore accept documents of arbitrary structure [28]. Usually, NoSQL databases do not offer services such as `JOIN` [28]. NoSQL databases are said to outperform out-of-the-box SQL databases [28]. 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 was 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.

The time necessary to perform the steps of filling the Elasticsearch database has been evaluated and improved throughout this work. The current time measurements are shown in Figure 5.1. Portioning the work with the database is beneficial since it is possible to update the embeddings without having to recreate the database. Moreover, modulizing the initialization and filling of the database facilitates debugging and comparing the models used to create the embeddings.

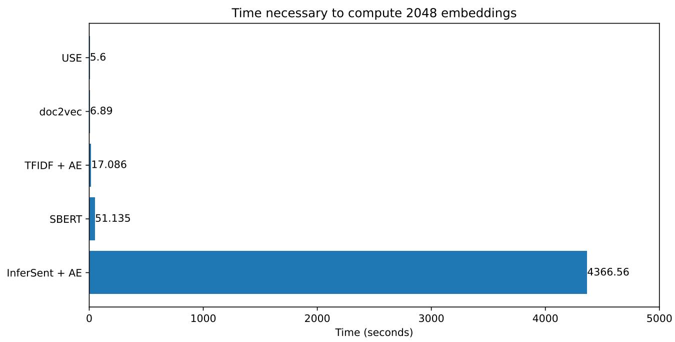


Figure 5.1: Time per module of creating the Bahamas database using a random selection of 2048 documents. The reference time was measured using `cProfiler` on a Apple M2 Pro MNW83D/A with 16 GB RAM and 12 cores.

5.2 Eigendocs

Assuming the layout holds information about the document type, the first page of each document is used to extract this information. In the course of working with low-quality versions of the documents to minimize the memory necessary to store them, some documents looked similar. Therefore, the idea arose to use clustering algorithms to group the documents according to their appearance.

In order to determine the optimal number of components used for Eigendocs the cumulative explained variance and the reconstruction error were plotted as displayed in Figure 3.11 from Subsection 3.5.2. The first plot indicated that 90% of the variance is explained by 95 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 cluster 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 was taken into consideration.

The calculation of the reconstruction error 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

```
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 to incorporate the content of the images assuming white pixels do not convey information.

information. Hence, the reconstruction error is divided by the portion of non-white pixels. The calculation is given in Listing 5.1. The result is displayed in Figure 3.11. The "elbow" points are visible at 13 and 30. Since the number of components ought to be kept reasonably small, 13 is chosen.

The results of the Eigendocs algorithm are displayed in Figure 4.8. 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 document 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 the maximum vector dimensionality of Elasticsearch. However, the quality of the embeddings should not decline too much.

The choice of the preprocessor was investigated with regard to the goal of minimizing the vocabulary size. Both the default and a custom preprocessor were tested on a data corpus of 2048 randomly selected documents with regard to the vocabulary (size). While the default preprocessor had a vocabulary size of 5893, the custom preprocessor had a size of 5585. The relative difference likely shrinks with a larger data set, since the trend is already visible for two different data corpus sizes in Table 5.1. The custom preprocessor was chosen because it had a smaller vocabulary size. The differences between both vocabularies were compared and 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.

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

Initially, there should have been two different TF-IDF models. The first one should have been used to obtain documents which are similar to the query document. Therefore, terms that occur only once in the corpus should have been removed from the vocabulary. The second approach should have been used to obtain specific documents from the corpus. Hence, the vocabulary should consist of very document-specific terms and thus, `max_df` would have been relatively low, to omit terms that occur in many documents. However, the restrictions imposed by the database implementation in terms of dimensionality limitations made it impossible to explore many parameter ranges. Therefore, only one TF-IDF model was used in the end, whose parameters `min_df` and `max_df` were set to values which kept the vocabulary and thus, the dimensionality of the embeddings reasonably small.

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.

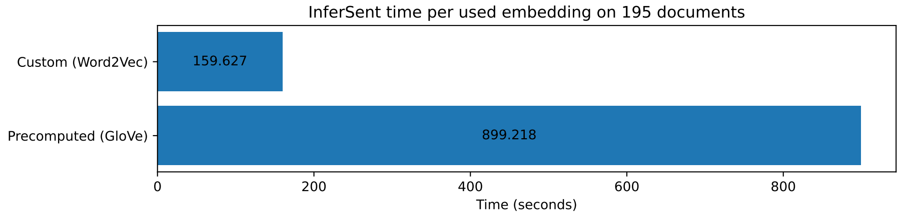


Figure 5.3: Reference time necessary to calculate and insert 195 InferSent embeddings for different precomputed word embeddings on a Apple M2 Pro MNW83D/A with 16 GB RAM and 12 cores. Using a custom Word2Vec model is around 5.5 times faster than GloVe.

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 was 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 was 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 [54]. 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 was used.

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 were 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 result obtained. 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 tarditional metric. The dataset used for the evaluation is a selection of 195 documents from the Bahamas dataset.

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

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

```
cos_sim = statistics.mean([np.dot(inverse_emb, embedding)
        /(np.linalg.norm(inverse_emb)*np.linalg.norm(embedding))
        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. While most of the architectures produced similar results, one architecture stood out. Combining 2500-, 3000- and 3500-dimensional layers in the hidden space produced the worst RSME results. The best results were 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.

5.4 Clustering using OPTICS

The algorithm OPTICS was applied to data, which was preprocessed according to item 1 and item 2. The clusters from Figure 5.5 were 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 combination of resizing and PCA of item 1 scatters the objects further along the x_2 axis. One could argue that the narrow distribution of the objects in the Eigendocs plot 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 and are thus, less scattered along this dimension.

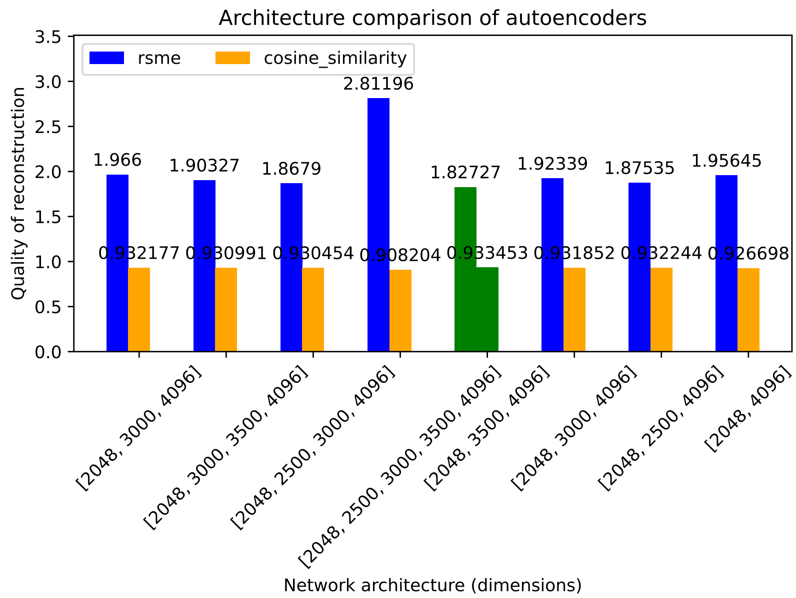
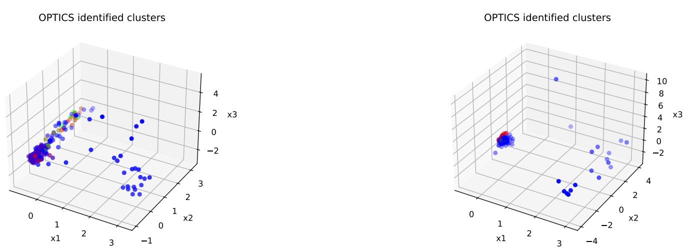


Figure 5.4: The effect of different AE architectures on the reconstruction error. The error is measured in terms of RSME (blue bars) and cosine similarity (yellow bars) between the original and the reconstructed image. The smallest RSME and the biggest cosine similarity belong to the architecture best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

To analyze the results of the clustering, the content of the clusters was examined. Since the documents are not labeled, the content of the clusters was analyzed by visual inspection. The content of the clusters is displayed in Figure 5.6 and Figure 5.7. The yellow images belong to the group identified as noise. The images preprocessed according to item 1 were partitioned into multiple small and one big cluster. The Eigendocs images' clusters have similar sizes. The row of noise images is way longer than the other rows in Figure 5.7. Most of the certificates are classified as noise for both approaches.

The preprocessing approach used to create the OPTICS input for the Elasticsearch database index is Eigendocs since it also encodes information about the document size.



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

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

Figure 5.5: The clusters were extracted from the respective reachability plots in Figure 4.9 by OPTICS. The blue points are noise points, whereas any other colour denotes a cluster.

Figure 5.6: The yellow images belong to the group denoted noise. Most certificates are classified as noise. There is one big cluster and multiple small clusters. The images were preprocessed as discussed in item 1 to 32x32 greyscale pixels.

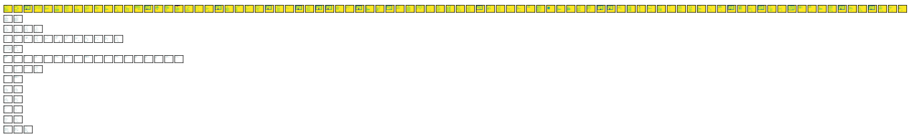


Figure 5.7: Most certificates are classified as noise. The rest of the clusters have similar sizes. The images were preprocessed as discussed in item 2 (i.e. Eigendocs) to 13-dimensional greyscale pixels.

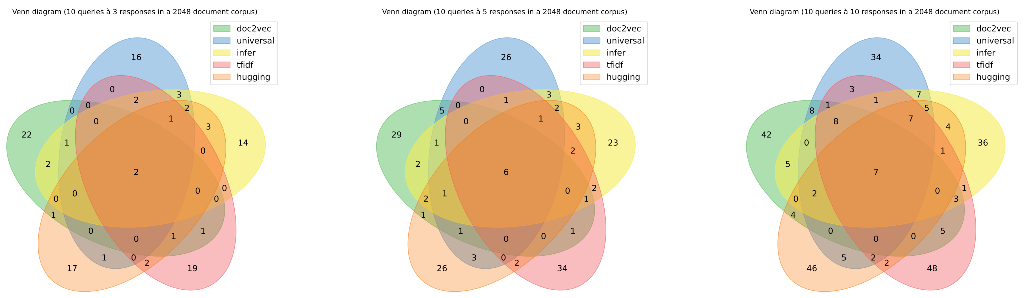
According to Deng et al., OPTICS was developed to improve DBSCAN flaws. With respect to the evolution of these clustering methods, i.e. DBSCAN being OPTICS basis, DBSCAN is chosen for the cluster method in Listing 4.12. In order to reduce calculation complexity the maximum ε is 10. The distance between two points to still be considered neighbours is defined after a visual inspection of the reachability plot. Considering the intrinsic structure of the data it is set to 0.5 to return meaningful clusters.

5.5 Comparison of models

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.

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 was 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 model's embedding of the query document in terms of cosine similarity. These responses are retrieved by a kNN query from the database for each model and stored in a Comma Separated Values (CSV) file. To facilitate working with the data, the document IDs can be encoded as monotonically increasing integers.

The first approach to compare the response documents is to visualize the intersection of the query results. The Venn diagram was chosen since it displays intersections of all items of the power set of the models. To build a Venn diagram, the number of documents that are



(a) 3 response documents per query.

(b) 5 response documents per query.

(c) 10 response documents per query.

Figure 5.8: 10 documents were randomly sampled from a 2048 document corpus. For each sampled document and model a kNN query was conducted. The respective response documents excluding the query document were saved. The cardinality of the intersection of all response documents irrespective of query document for different models is visualized in terms of Venn diagrams.

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 5.8. 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 an intersection of documents that produces an empty set has a non-empty area in this visualization.

The Venn diagrams in Figure 5.8 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 at the bottom is close to zero across all Venn diagrams. The blue (USE) and orange (SBERT) models, as well as the yellow (InferSent), orange (SBERT) and blue (USE) models, yield consistently similar results.

Since the Venn diagrams compare the responses of the models irrespective of the query document, a second 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. It is possible to normalize the matrix to obtain the portion of shared query results. Thereafter the matrix is visualized using a heatmap. The matrix consists of five rows and five columns, where each row and column represents a model. The code snippet in Listing 5.4 shows the calculation of the similarity matrix.

The cell values are the number of shared query results between the models of the row and column. They are calculated by summing up the number of shared query results per document query. It is possible to return the total number of shared documents per model pair or to normalize the matrix by dividing each cell value by the total number of query

```
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.

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 then visualized using a heatmap as displayed in Figure 5.9.

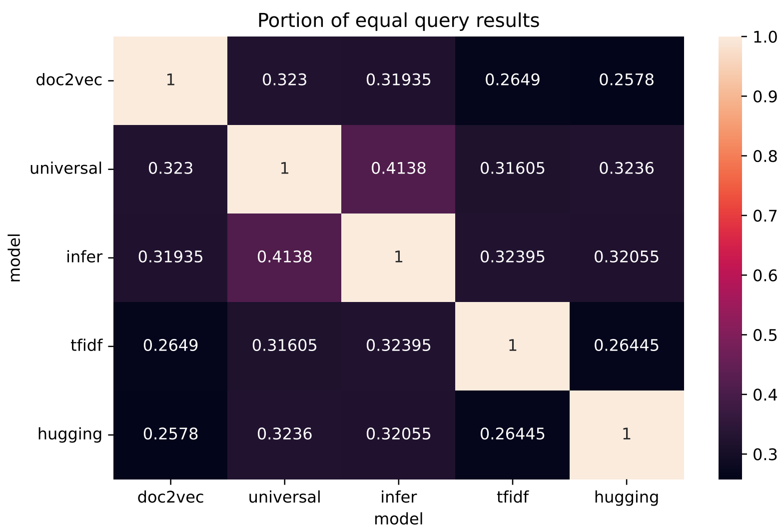


Figure 5.9: Heatmap visualizing portion of shared query results on 2000 queries à 10 responses on a 2048 document corpus.

The heatmap in Figure 5.9 shows that the models yield dissimilar query responses. It arose from 2000 queries à 10 responses on a 2048 document corpus. The similarity matrix was normalized to present the portion of shared query results for a query. 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.

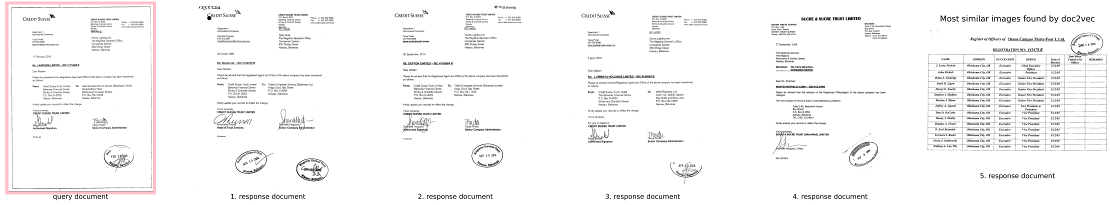
Table 5.2: Mean and standard deviation of the average portion of shared response documents of different models on a 2048 document corpus. One trial produced five real values, i.e. one portion per model. A portion is obtained from 10 randomly sampled query documents à 10 responses (excluding the query document in the response). There were 30 trials.

model 1	**model 2**	**mean**	**std**
Doc2Vec	Doc2Vec	1.0	0.0
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	USE	1.0	0.0
USE	InferSent	0.33	0.08
USE	TF-IDF	0.25	0.08
USE	SBERT	0.25	0.06
InferSent	InferSent	1.0	0.0
InferSent	TF-IDF	0.26	0.08
InferSent	SBERT	0.24	0.06
TF-IDF	TF-IDF	1.0	0.0
TF-IDF	SBERT	0.18	0.05
SBERT	SBERT	1.0	0.0

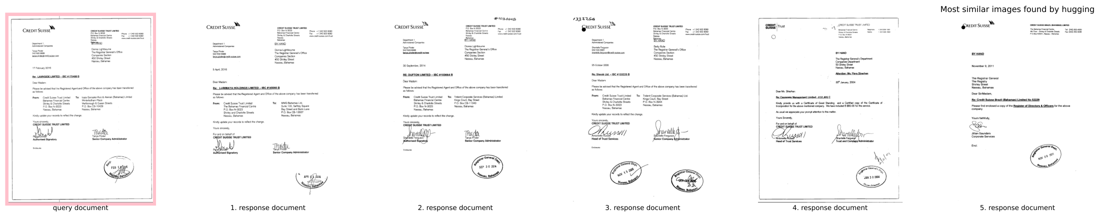
Therefore, 30 trials were conducted. Each trial consisted of 10 randomly sampled query documents à 10 responses (excluding the query document in the response). The normalized similarity matrix for each model pair concerning one trial was calculated using Listing 5.4. The cardinalities were stored in a dictionary of lists indexed by the respective model pairs. Finally, the mean and standard deviation were computed for each model pair and stored in a 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 non-reflexive combinations. As observed before, any two-element combination of Doc2Vec, TF-IDF and SBERT has the lowest mean portion ($18 - 20\%$) of shared query results.

To display the differences between the models exemplary, the five most similar documents to a random document were returned from the database and visualized. The text of the query document was encoded using the respective model and a kNN query was used to obtain the results from the local database containing 2048 documents. The query image, i.e. the one surrounded by a border, was omitted from the database query response. The query responses are listed according to descending similarity to the query document. The query results are displayed in Figure 5.10. 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.

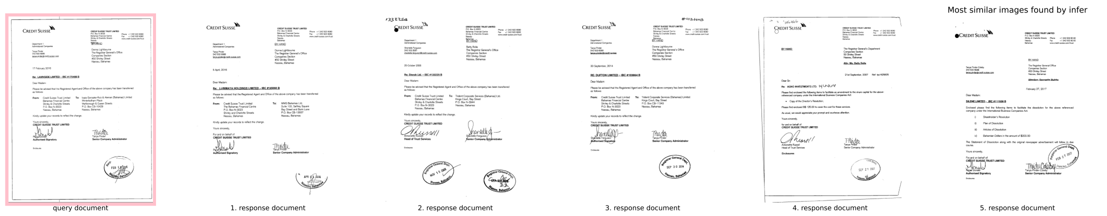
To further investigate the differences between the models, the query responses of the models are compared qualitatively on documents that were found to be unusual in terms of visual inspection. The first document in Figure 5.11 is a table consisting of little text compared



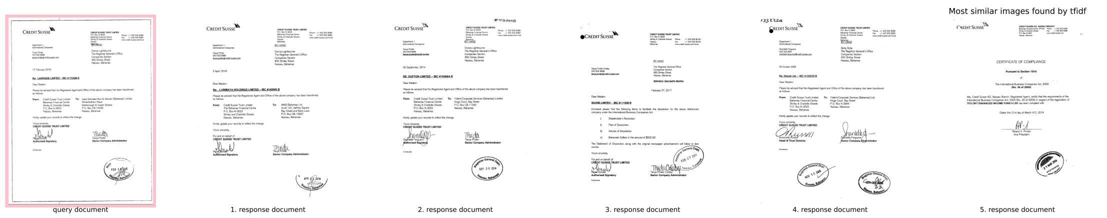
(a) Doc2Vec



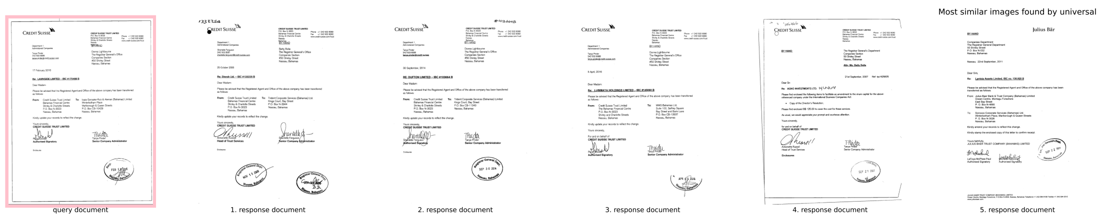
(b) SBERT



(c) InferSent



(d) TF-IDF



(e) USE

Figure 5.10: Exemplary query response for different embeddings on the same query document.

to other samples from the data corpus. The second document in Figure 5.11 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 5.11. 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.

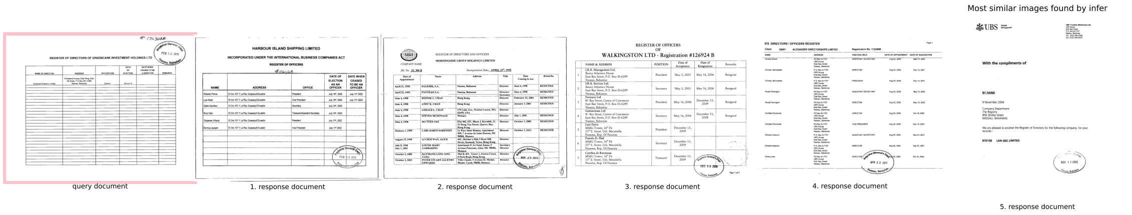
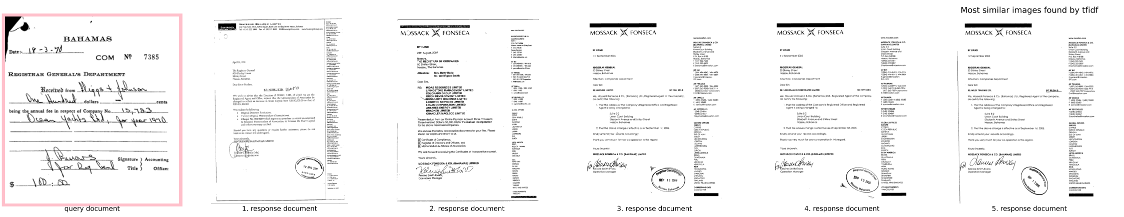
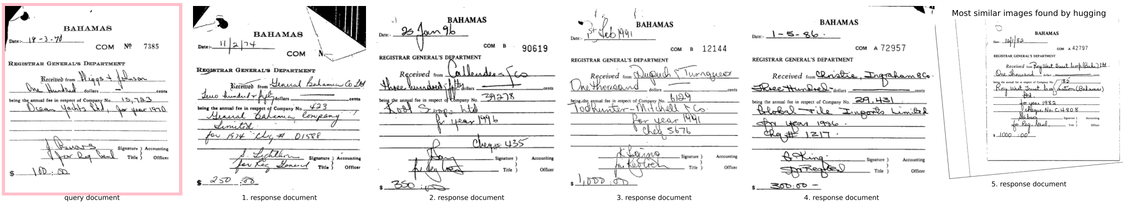


Figure 5.11: InferSent query responses on a query document consisting of little text.

There are query documents such as the one in Figure 5.11 that reveal the dissimilarity between certain models. Most models' response documents are similar to Figure 5.13(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 5.13(a) are not handwritten and cover a different content, i.e. requesting a payment and three documents concerning an address change. TF-IDF's results could thus be considered to be of poor quality.



(a) TF-IDF



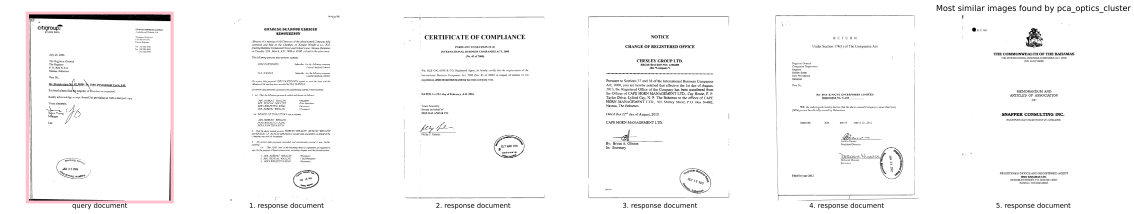
(b) SBERT

Figure 5.11: Qualitative comparison of query responses. The majority of the query document consists of handwritten text. The results of the TF-IDF model are not very similar to the query document and thus, are considered to be of poor quality. The other models, for example, SBERT, produce results that are more similar to the query document.

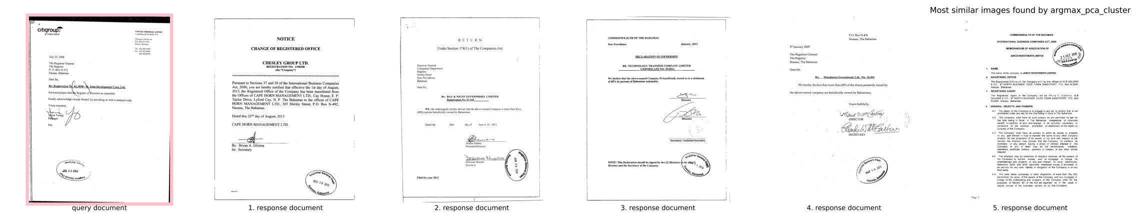
Since not only textual information but also visual information is encoded in the database, the next step is to compare the query responses of approaches that consider visual similarity. The query responses are clustered using OPTICS or `argmax` of the PCA compression.

The first exemplary query document in Figure 5.12 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 from the group. None of the result documents originate from the same company as the query document.

The second exemplary query document in Figure 5.13 is a certificate. 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.



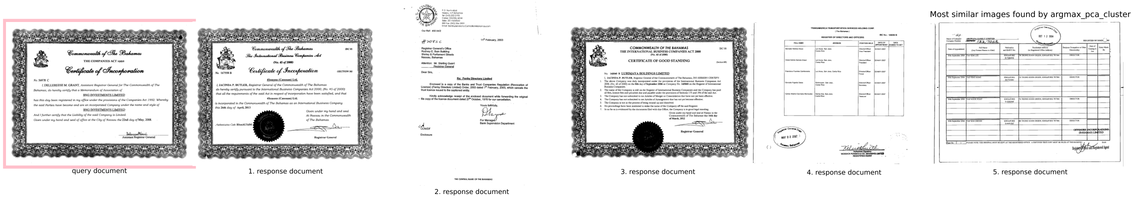
(a) OPTICS



(b) `argmax` of PCA compression

Figure 5.12: 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.

(a) OPTICS



(b) `argmax` of PCA compression

Figure 5.13: 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.

5.6 Comparison with baseline topic modelling approach

The baseline topic modeling 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 and visualization by the means of Word Clouds. 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.

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 component 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.

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.

6 Conclusion & Outlook

In Section 6.1 a discussion on the methods used and the contributions are presented. Section 6.2 concludes with an outlook on the shortcomings of this thesis and future work.

6.1 Conclusion

The techniques examined in this work are discussed in Subsection 6.1.1. This section gives a brief outline for which reasons the techniques have been chosen. Afterwards, the contributions of this work are summarized in Subsection 6.1.2.

6.1.1 Discussion

Before working with a model, the data has to be processed to a format that the model can work with. The requirements imposed on the data format depend on the model. Some models do not require any preprocessing, while others require a lot of preprocessing.

In this work, the TF-IDF configuration includes the definition of a custom preprocessor. Consequently, preprocessing played an important role in this embedding. The preprocessor was constructed with respect to the well-established preprocessing steps for NLP tasks [8], the TF-IDF documentation about the default preprocessor and the task. Since the data set can contain figures due to the financial background, the preprocessor encodes all numbers into discrete strings. Hence, the information is not lost but strongly reduced, which is vital due to the constraints imposed by the database's dense vector dimension limitation. The custom preprocessor was compared to the default preprocessor on multiple datasets. Since it consistently produced smaller vocabulary sizes, it was chosen for the task.

The other semantic embedding methods do not require any preprocessing. The visual embedding methods required more preprocessing. Originally, the images were matrices of `RGB` values. However, PCA and OPTICS require a vector as input. Therefore, the images were converted to greyscale and flattened to a vector. The images were resized to make them more comparable. Since OPTICS' performance deteriorates with increasing dimensionality, applying PCA to compress the data can be considered preprocessing.

As mentioned before, the database has a dense vector dimension limitation of 2048. This limitation is a problem for the visual embedding methods as well as InferSent and TF-IDF. The visual embedding methods were compressed with PCA, whereas the encoder of a trained AE was employed for textual embeddings. The PCA was fit on 1000 randomly selected images. The number of components was chosen beforehand in consideration of the explained variance and the reconstruction error. The architecture of the AE was chosen in consideration of the reconstruction error on 195 documents. Both compression techniques would benefit from evaluation on a larger data set.

As already stated this work examined both ways to encode semantic and visual information. The evaluation was conducted with respect to the task of finding similar documents. Since the data set is not labeled the evaluation is subjective and experimental. Qualitative evaluation was conducted by inspecting the responses of different models for several sample queries to the database storing the respective embeddings. The inspection was conducted manually, using Venn diagrams, heatmaps and statistical properties of the distribution of the cardinality of the shared query response sets.

The semantic responses were similar and often managed to find either documents with the same content or documents with the same company name. The visual responses were more dissimilar from each other and the query document. More precisely, the `argmax` approach returned poor results to some extent. Consequently, OPTICS is considered the superior visual embedding method for the database. Moreover, the textual embedding methods produced more meaningful responses than the visual embedding methods. For example, textual embedding methods managed to find documents with the same company name, while the visual embedding methods naturally returned visually similar documents.

When evaluating the semantic embedding methods, slight differences between the models became evident. Differences and similarities between the models were examined by comparing the responses to the sample queries. This evaluation included the calculation of the average portion of shared documents between the responses of several models and exemplary evaluation of actual queries. TF-IDF, Doc2Vec and SBERT were most dissimilar from each other. The TF-IDF approach performed rather poorly on unusual query documents.

The similarity metric used in this work is cosine similarity. Since Elasticsearch provides the similarity metrics Euclidean distance, dot product and cosine similarity, one of these metrics had to be chosen. Usually, cosine similarity is recommended for NLP tasks and thus, applied in this work. Soft cosine similarity is not used, since it is not available in Elasticsearch. However, the usage of soft cosine would likely improve the results.

Since the techniques discussed in this section have to be merged into a single system Word Cloud provides means to visualize the response documents containing the most similar documents for different models. This approach is well suited to describe topics as groups of predominant words [36]. The baseline topic modeling technique Top2Vec provides a Word Cloud implementation too.

The database used in this work is Elasticsearch. It was built to provide a fast and scalable search engine. Moreover, it is a document-oriented database, which is well-suited for flexible data. By default, several similarity measures and search strategies are provided. Moreover, the elastic stack offers a wide range of tools supporting embedding models which were not used in this work.

The libraries used for the implementation of the tool are Angular and Flask. The libraries were chosen after discussing options with this thesis' supervisor. However, the implementation of this tool is not the focus of this work.

The baseline topic modeling technique Top2Vec was chosen due to its simplicity and functionalities. The Top2Vec implementation is well documented and provides a Word Cloud implementation. Moreover, it enables the user to find similar topics for query terms and returns inherent topics in a data corpus. Hence, its functionality is similar to the functionality of the system developed in this work.

6.1.2 Contribution

An exhaustive literature review was conducted to discover suitable embedding methods for the task of finding similar documents. Initially, only semantic embedding methods were considered. However, when the first results were inspected the idea arose to group the documents by visual similarity. Thus, visual embedding methods were included in this work.

The semantic embeddings were generated by existing models. All models offer adequate documentation and were thus, mostly comfortable to use. However, some alterations had to be made to the models to make them suitable for the task. For instance, a custom preprocessor was implemented for the TF-IDF embedding method. The preprocessor was a better fit for the task than the default preprocessor. Moreover, a custom Word2Vec model was trained on the dataset. This Word2Vec model is used by InferSent reducing the time necessary to encode texts.

The Eigenfaces approach has been prevalent since 1991. In this work, the approach was adapted to the task of finding similar documents and thus, called Eigendocs. The idea of projecting items into a lower dimensional space was kept as well as the preprocessing steps. However, the preprocessing was extended by placing the document images onto a white canvas.

In order to find a suitable clustering algorithm for the task, literature research was conducted. The research revealed that the OPTICS algorithm is a suitable candidate for the task. In this work, two different preprocessing steps were implemented. The first was similar to the preprocessing steps used in [5]. The second one utilized Eigendocs to reduce the dimensionality of the data set. The parameters of the algorithm were chosen in consideration of the reachability plot.

The data was compressed using PCA or the encoder of an AE. Both the PCA and AE configurations were experimentally evaluated. The number of components of the PCA was chosen in consideration of the cumulative explained variance and an adaption of the reconstruction error RSME. The architecture of the AE was chosen in consideration of the RSME and the cosine similarity between the original and the reconstructed vector.

Before the data could be used for the task, it had to be stored in a database. Firstly, the type of database had to be chosen. It became clear that a document database would be the best fit for the task since it accepts documents of variable shape. Elasticsearch is a well-known document database that is used by well-established companies successfully handling big data. Secondly, the database had to be configured. This required the consideration of different field types and similarity measures. The dense vector seemed to be a native choice and cosine similarity was chosen among other reasons due to its popularity as a similarity metric in VSM. Afterwards pipelines to store the data in the database were implemented. The local database stores 2048 randomly chosen documents from the whole data set. The server database currently stores around 497504 incomplete documents.

Since the dataset is not labeled, the evaluation of the results is not trivial. Therefore, multiple evaluation methods were implemented. The first method is a Venn diagram that depicts the intersection of the query responses of the power set of different methods. The second method is a heatmap that shows the average portion of shared response documents between different methods. Lastly, the mean and standard deviation of the portion of shared response documents were calculated to further investigate the distribution of the results obtained above.

To ensure Word Clouds display only valid words the tokens were lemmatized before applying the Word Cloud implemented by Müller. The existing functionalities implemented by the library Top2Vec were bundled into a class which accepts inputs native to the task at hand. Hence, the functionalities were not extended but rather adapted to the task.

As initially mentioned, this thesis aims to provide computational means to facilitate the work with large unstructured text data. In the course of this work, multiple ML techniques were examined and evaluated on the task of finding similar documents. The semantic embedding methods TF-IDF, Doc2Vec, InferSent, USE and SBERT were examined. Eigendocs was implemented as a visual embedding method. It was possible to find a suitable database to store the data. A pipeline to preprocess, embed and store the data in the database was developed. The pipeline is scalable to large datasets. However, models such as TF-IDF yield less meaningful results on certain query documents and are prone to performance deterioation with increasing document corpus, since the embeddings' dimensionality correlates directly with the vocabulary size. When the number of terms in the data corpus increases, but the vocabulary size is static, it is likely that the vocabulary lacks important words.

Moreover, a tool was implemented. The tool provides the possibility to conduct text queries. A document of interest can be examined in more detail: The detail component not only contains a PDF viewer, a Word Cloud of the most frequent words in the document, but also an option to examine the query responses for different embeddings. The file names of the ten most similar documents and a Word Cloud of the most frequent words among those documents is displayed.

An experimental evaluation of the embeddings was conducted. The visual embedding methods were found to be inferior to the semantic embedding methods on certain query documents. Moreover, the tool was compared to a baseline topic modeling approach called Top2Vec. The library Top2Vec implements similar functionalities to the tool developed in this work. However, the tool is not designed for a productive environment since the focus is on the comparison of different models rather than usability.

6.2 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. 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.

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 performed rather poorly on unusual query documents. There are multiple factors that could have contributed to this result. Firstly, the vocabulary was drastically reduced to satisfy the database's constraints for dense vector dimensionality. Thus, TF-IDF might 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 was carried out on small datasets of 195 and 2048 documents. This dataset may not be representative of the whole corpus.

In this work, the precomputed GloVe embeddings were 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 [54]. Hence, the quality of InferSent might have deteriorated due to the replacement of GloVe by Word2Vec.

When preprocessing the document images for Eigendocs, the images were placed on a white canvas assuming its dimensions were bigger or equal than 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.

The parameter selection for PCA is not representative of the whole dataset, due to the fact that the dataset used for the evaluation error was not drawn randomly from the data corpus and is too small. Moreover, the resulting plot was not optimal for conducting the "elbow method", since no significant change was evident in the slope.

Different AE architectures were experimentally evaluated on a selection of 195 documents. However, 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 thorough 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 data set 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. Thus, future work could include the weights of the query responses in the evaluation.

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 was used to split the workload across up to 100 processes, the embedding process has 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. 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. However, the tool developed in this work is not yet ready to be used in tax offices. The different embedding models to choose from are useful for research purposes but not in a productive environment.

Bibliography

[1] 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, 2015.

[4] Dimo Angelov. Top2vec: Distributed representations of topics. 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*, 40:100378, 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, 2018.

[10] Allison Chaney and David Blei. Visualizing topic models. 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, 2017.

[12] Qiuxing Chen, Lixiu Yao, and Jie Yang. Short text classification based on lda topic model. In *2016 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, 2022.

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

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

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

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

[18] download-glove. Glove: Global vectors for word representation. URL `https://nlp.stanford.edu/projects/glove/`. [Accessed 04.10.2023].

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

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

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

[22] 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].

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

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

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

[26] Elasticsearch-text-analyser. Text analysis overview. URL `https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-overview.html`. [Accessed 15.09.2023].

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

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

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

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

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

[32] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. 2022.

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

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

[35] Hari Krishna Kanagala and V.V. Jaya Rama Krishnaiah. A comparative study of k-means, dbscan and optics. In *2016 International Conference on Computer Commu-*

nication 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(24), 7 2019. doi: 10.4108/eai.13-7-2018.159623.

[37] Gunjan Khosla, Navin Rajpal, and Jasvinder Singh. Evaluation of euclidean and manhanttan metrics in content based image retrieval system. In *2015 2nd 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, 2014.

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

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

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

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

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

[44] 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. doi: https://doi.org/10.1016/j.procs.2020.03.219. International Conference on Computational Intelligence and Data Science.

[45] 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.

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

[47] 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.

[48] 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].

[49] 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 *2019 International Electronics Symposium (IES)*, pages 214–219, 2019.

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

[51] Li-Qiang Niu and Xin-Yu Dai. Topic2vec: Learning distributed representations of topics. 2015.

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

[53] Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok Choudhary. Scalable parallel optics data clustering using graph algorithmic techniques. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2013. Association for Computing Machinery.

[54] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation, 2014.

[55] 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, 2020.

[56] 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].

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

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

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

[60] Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model, 2014.

[61] Gerd Stumme, Robert Jäschke, and Christoph Scholz. Internet-suchmaschinen, 2011.

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

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

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

[65] Zhou Tong and Haiyi Zhang. A text mining research based on lda topic modelling. page 201–210, 2016.

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

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

[68] Ike Vayansky and Sathish A.P. Kumar. A review of topic modeling methods. *Information Systems*, 94, 2020.

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

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

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

[72] 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.

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

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

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

[76] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. Tfidf, lsi and multi-word in information retrieval and text categorization, 2008.

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

Declaration of authorship

I hereby declare that I am the sole author of the bachelor's thesis with the title "Identification of key information with topic analysis on large unstructured text data" and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

Kassel, November 7, 2023

Klara Maximiliane Gutekunst