Interactive Analysis of a Corpus of General Terms and Conditions for Variability Modeling

David Koller
Interaktive Analyse eines Korpus von Allgemeinen Geschäftsbedingungen zur Variabilitätsmodellierung

Interactive Analysis of a Corpus of General Terms and Conditions for Variability Modeling

Author: David Koller
Supervisor: Prof. Dr. rer. nat. Florian Matthes
Advisor: M.Sc. Ingo Glaser
Submission Date: 15.04.2019
I assure the single handed composition of this master's thesis only supported by declared resources

Garching b. München, 15.04.2019

__________________________

David Koller
Abstract

With worldwide E-commerce sales continuously growing, many companies entire the digital market. To conduct business online, most specify a tailor-made set of general terms and conditions for their company. Because of the optional nature and lack of formal regulations for these documents, creating reliable terms and conditions without legal domain knowledge is almost impossible. This thesis aims to provide an overview of common clause, relations and clauses that should be avoided in form of a variability model. For this task, we develop an interactive matching system to classify German language terms and conditions, by grouping semantically related clauses using text similarity and supervised machine learning approaches. We investigate to which degree the matching system can classify unknown terms and conditions and evaluate the emerging classes of clauses, to extract the most frequent types.
Zusammenfassung

# Contents

Abstract.................................................................................................................................III

Zusammenfassung......................................................................................................................IV

Contents.....................................................................................................................................V

List of Figures..........................................................................................................................VIII

List of Tables...........................................................................................................................X

List of Listings........................................................................................................................ XI

Abbreviations..........................................................................................................................XII

1. Introduction..........................................................................................................................1  
   1.1 Research Questions ..........................................................................................................2  
   1.2 Research Approach ..........................................................................................................2  
      1.2.1 Iterative Approach ....................................................................................................2  
      1.2.2 Ground Truth ............................................................................................................3  
      1.2.3 Data Set ....................................................................................................................4  
   1.3 Structure of this Thesis......................................................................................................5  

2. Related Work.......................................................................................................................6

3. Basic knowledge....................................................................................................................7  
   3.1 Natural Language Processing (NLP) ...............................................................................7  
      3.1.1 Semantic Text Matching (STM) ..............................................................................7  
      3.1.2 Text Preprocessing techniques ...............................................................................8  
      3.1.3 Text Similarity Methods ........................................................................................10  
   3.2 Interactive Machine learning ........................................................................................15  
   3.3 Variability Modeling .......................................................................................................17

4. Requirements and Design...................................................................................................19
# Contents

4.1 Requirements ........................................................................................................... 19  
  4.1.1 Functional Requirements ................................................................................ 19  
  4.1.2 Non-Functional Requirements ...................................................................... 20  

4.2 Design ....................................................................................................................... 20  
  4.2.1 Software Architecture ..................................................................................... 20  
  4.2.2 Interactive Process .......................................................................................... 21  

4.3 Data Model .............................................................................................................. 25  

5. Implementation Matching System .............................................................................. 26  
  5.1 User Interface ...................................................................................................... 26  
    5.1.1 AGBListView .................................................................................................. 27  
    5.1.2 AGBFormView ................................................................................................ 28  
    5.1.3 ParagraphMatchingView .............................................................................. 29  
    5.1.4 ClauseMatchingView .................................................................................... 29  
    5.1.5 AGBDetailView .............................................................................................. 30  
    5.1.6 ClassDetailView ............................................................................................ 31  
    5.1.7 DataView ......................................................................................................... 31  
    5.1.8 General Components ..................................................................................... 32  
  5.2 Back-end ................................................................................................................ 33  
    5.2.1 Data Model Implementation ............................................................................. 33  
    5.2.2 Workflow ....................................................................................................... 34  
    5.2.3 Routes for Classification process ................................................................. 41  
    5.2.4 Additional REST API Routes ........................................................................ 43  

6. Evaluation .................................................................................................................. 46  
  6.1 Quantitative Evaluation ......................................................................................... 46  
    6.1.1 Evaluating Classification Process ................................................................. 47  
    6.1.2 Validation and Comparison to Paragraph Vectors ........................................ 53  
  6.2 Qualitative Evaluation ........................................................................................... 56  
    6.2.1 Suitability of our Ground Truth .................................................................... 56  
    6.2.2 Feature Modeling for Terms and Conditions ................................................. 58
## Contents

7. Conclusion ........................................................................................................... 64

Bibliography ............................................................................................................. 66

Appendix A ............................................................................................................... 68

A.1 Ground Truth .................................................................................................. 68

A.2 References for illegal Clauses ......................................................................... 77
List of Figures

Figure 1.1: Iterative Research Approach .................................................................3
Figure 3.1: Illustration of Semantic Text Matching for our System ..............................8
Figure 3.2: Preprocessing Pipeline for Terms and Conditions ..................................10
Figure 3.3: Model architecture for CBOW and Skip-Gram ......................................13
Figure 3.4: The Distributed Memory model (PV-DM) left and the Distributed bag of words
(DBOW) model right ......................................................................................15
Figure 3.5: Difference between Traditional and Interactive Machine Learning ..............16
Figure 3.6: Feature model of a mobile phone product line ........................................18
Figure 4.1: UML Deployment Diagram for our Matching System ...............................21
Figure 4.2: UML Use Case Diagram for Matching System ......................................22
Figure 4.3: UML Sequence Diagram for adding Terms and Conditions ......................23
Figure 4.4: UML Sequence Diagram for matching Terms and Conditions ..................24
Figure 4.5: Data Model for the interactive matching system .....................................25
Figure 5.1: React view and components for front-end application ..............................26
Figure 5.2: Overview of all terms and conditions in our system ..................................27
Figure 5.3: Overview over model selection for predictions .......................................27
Figure 5.4: Text Input tab from AGBFormView ......................................................28
Figure 5.5: Check Format tab for AGBFormView ..................................................28
Figure 5.6: Currently dragged paragraph in ParagraphMatchingView .......................29
Figure 5.7: Predicted Classes for Clauses in ClauseMatchingView .............................30
Figure 5.8: Overview over paragraphs and clauses from a chosen online-shop .............30
Figure 5.9: ClassDetailView with paragraphs of a class .........................................31
Figure 5.10: Screenshot of the DataView ................................................................32
Figure 6.1: Scores for Paragraph Classification Process ..........................................49
Figure 6.2: Precision of all Methods for Paragraphs ...............................................50
Figure 6.3: Score for Claus Classification Process ....................................................51
List of Figures

Figure 6.4: Precision of all Methods for Clauses .................................................. 52
Figure 6.5: Cross Validation Scores for Paragraphs .............................................. 54
Figure 6.6: Comparison between Embedding Types for Paragraphs ..................... 54
Figure 6.7: Cross Validation Scores for Clauses .................................................... 55
Figure 6.8: Comparison between Embedding Types for Clauses .......................... 55
Figure 6.9: Paragraph Distribution ........................................................................ 56
Figure 6.10: Clause Distribution ........................................................................... 57
Figure 6.11: Number of Terms and Conditions with Paragraphs in each Class ...... 59
Figure 6.12: Number of Terms and Conditions with Clauses in each Class ........... 60
Figure 6.13: Feature Model of “Optional” Paragraphs and their Clauses ................. 61
Figure 6.14: Feature Model of Mandatory Paragraphs and their Clauses ............... 63
List of Tables

Table 6.1: “Most Unique Classes” Distribution for Paragraphs ........................................50

Table 6.2: “Most Unique Classes” Distribution for Clauses ........................................52
List of Listings

Listing 5.1: Excerpt of exemplary Class Definition ..........................................................33
Listing 5.2: Example JSON of a Paragraph Object ............................................................34
Listing 5.3: Excerpt for creating Agb, Paragraphs and Clauses ........................................35
Listing 5.4: Pre-processing of Paragraphs and Clauses ......................................................36
Listing 5.5: Creating ELMos for Paragraphs .......................................................................37
Listing 5.6: Excerpt of example Doc2Vec model creation ..................................................38
Listing 5.7: Excerpt for creating Document Embeddings for new and old Paragraphs .........38
Listing 5.8: Predictions for Paragraphs based on Cosine Similarity ....................................39
Listing 5.9: Machine Learning Models used for classification ............................................39
Listing 5.10: Model Training and most unique classes .........................................................40
Listing 5.11: Excerpt to provide Predictions to Front-End ..................................................41
Listing 5.12: Excerpt of REST API route to fetch paragraphs .............................................42
Listing 5.13: Excerpt to fetch Paragraphs of relevant Clauses ...........................................42
Listing 5.14: Excerpt to set true Class for Clauses ..............................................................43
Listing 5.15: Data for Bar Charts about Class Distribution of Clauses ...............................44
Listing 5.16: Excerpt of Back-End for ClassDetailView ....................................................45
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGB</td>
<td>Allgemeine Geschäftsbedingungen</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CBOB</td>
<td>Continuous Bag-of-Words</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>CS</td>
<td>Cosine Similarity</td>
</tr>
<tr>
<td>DT</td>
<td>Decision Tree</td>
</tr>
<tr>
<td>ELMo</td>
<td>Embeddings from Language Models</td>
</tr>
<tr>
<td>FODA</td>
<td>Feature-Oriented Domain Analysis</td>
</tr>
<tr>
<td>iML</td>
<td>Interactive Machine Learning</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>KNN</td>
<td>K-Nearest Neighbour</td>
</tr>
<tr>
<td>LDA</td>
<td>Linear Discriminant Analysis</td>
</tr>
<tr>
<td>LR</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long-Short-Term-Memory</td>
</tr>
<tr>
<td>NB</td>
<td>Gaussian Naïve Bayes</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>PV</td>
<td>Paragraph Vector</td>
</tr>
<tr>
<td>PV-DBOW</td>
<td>Paragraph Vector - Distributed Bag of Words</td>
</tr>
<tr>
<td>PV-DM</td>
<td>Paragraph Vector - Distributed Memory</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SMT</td>
<td>Semantic Text Matching</td>
</tr>
<tr>
<td>T&amp;C</td>
<td>Terms and Conditions</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>Term Frequency – Inverse Document Frequency</td>
</tr>
</tbody>
</table>
1. Introduction

The digitalization of knowledge and information transforms most industries, including the legal domain. With retail E-commerce sales continuously growing worldwide, many new digital marketplaces and online shops are created, all of which specify their own general terms and conditions to conduct business online. Unfortunately, there is no legally binding standard for general terms and conditions, which leads to high variety between retailers with regards to content, form and lawfulness of their terms and conditions. While some clauses commonly appearing in terms and conditions like information about the right to withdraw or rights of the data subject introduced with the General Data Protection Regulation (GDPR) are mandatory for companies conducting business online, most clauses defined in terms and conditions are completely optional with little restrictions about their content.

For new companies getting into the online market, the wide variety of clauses and legal requirements makes it almost impossible to create good and lawful terms and conditions for their online presence. The only way to guarantee reliable terms and conditions is the acquisition of expert knowledge by hiring legal domain experts, which can be done in two ways: either have a lawyer draft the terms and conditions for a one-time few, with additional cost if the terms need to be updated in the future, or purchasing a monthly subscription for a small few from one of the relevant providers like “IT-Rechts Kanzlei”.¹

Therefore, this thesis is concerned with the creation of an overview containing the common clauses, mapping relations and providing example for clauses, that must be avoided. Because annotated data sets with Germans language terms and conditions do not exist, we design and implement a system, that allows for the interactive classification of clauses from a corpus of terms and conditions, specifically collected for this thesis, to then evaluate and extract some of the most common clauses and create a model, displaying commonality and variability between terms and conditions.

¹ https://www.it-recht-kanzlei.de/Service/agb-online-shop.php, last accessed 14.04.2019
1.1 Research Questions

For the development of such a matching system and to generate the variability model for paragraphs and clauses from a corpus of general terms and conditions, this thesis will address the following research questions:

1. What is interactive Machine Learning and how can it support the classification of paragraphs and clauses from a corpus of general terms and conditions?
2. What are the functional and non-functional requirements for the matching system of terms and conditions?
3. How does a prototypical implementation allowing legal domain experts to interact with system look like?
4. How could a variability model displaying relations between clauses and paragraphs look like?

1.2 Research Approach

This section introduces our iterative research approach and presents our ground truth as well as the data set used for our interactive matching system.

1.2.1 Iterative Approach

As already mentioned, the main goal of this thesis is the design and implementation of a prototypical system to classify paragraphs and clauses from a corpus of terms and conditions based on a predefined ground truth. To successfully develop such a system, we first perform extensive literature research on topics related to our problem. This includes sub areas of natural language processing like the semantic text matching text, common preprocessing techniques and text similarity approaches, interactive machine learning approaches and variability modeling. Based on the acquired knowledge, we define requirements for our matching system, and design a fitting architecture as well as the interactive process between user and system. Afterwards, the system is implemented, and we begin the interactive process of classifying paragraphs and clauses from our collected corpus of terms and conditions. In the end, we quantitively evaluate our sys-
1. Introduction

tem and the interactive process, before building a model displaying commonality and variability between terms and conditions. Figure 1.1 illustrates this approach.

![Figure 1.1: Iterative Research Approach](Source: Own Illustration)

1.2.2 Ground Truth

To classify terms and conditions, we first have to decide what these classes should be. We base them on a sample terms and conditions document specifically for online-shops provided by “Bundesverband E-Commerce und Versandhandel Deutschland e.V”.  

These terms and conditions originally have twelve paragraphs containing a total of 43 clauses. Since we can’t assume these terms and conditions to cover all possible paragraphs and clauses, miscellaneous classes called “Sonstige” were added, where every paragraph/clause without an equivalent in our ground truth would go. To make the qualitative evaluation discussed easier, instead of just adding on overall miscellaneous class for paragraphs and one for clauses, we added these classes to every paragraph, where clauses with thematic fit but no real equivalent in our ground truth would go.

For example, the clause “Die Lieferzeit beträgt, sofern nicht beim Angebot anders angegeben, 3-5 Tage.”, isn’t semantically related to any of the clauses in our ground truth, but it clearly thematically related to the paragraph “§ 6 Lieferung; Eigentumsvorbehalt”. Therefore, the clause would be sorted in the “Sonstige” class of the above-mentioned paragraph.

After adding those classes, we end up with a ground truth containing 13 paragraph and 56 clause classes (The complete ground truth can be found in Appendix A.1).

---

1.2.3 Data Set

There exists no publicly available data set of Germans general terms and conditions for online-shops. Consequently, the data set for this thesis has to be collected by us. Since our focus is on terms and conditions from online shops in this work, we used idealo\(^3\) as a hub to create our data set. Originally, a web scraper was used to collect direct links to terms and conditions of many online shops. With these links, the webpages were crawled, and the html pages downloaded. Unfortunately, we couldn’t just collect the terms and conditions from their websites, because there is no clearly defined start and end for them. Manually cleaning up and extracting the relevant information from the downloaded html pages turned out to more time consuming than simply going to all websites manually and copy and pasting the terms and conditions.

In total, terms and conditions of 98 online shops were collected this way. Including the ground truth, we now have 99 documents with a total of 1215 paragraphs containing 3471 clauses.

\(^3\) https://www.idealo.de/preisvergleich/AllePartner.html, last accessed 14.04.2019
1.3 Structure of this Thesis

The following chapters of this thesis are structured in the following way:

- Chapter 2 gives a brief insight into related work for this topic.
- Chapter 3 builds the knowledge foundation for several natural language processing tasks, interactive machine learning, and variability models.
- Chapter 4 discusses requirements and the architecture for our matching system and designs the interactive process between user and system.
- Chapter 5 presents the implementation of our interactive matching system by first introducing the user interface, before taking a closer look at the back-end services.
- Chapter 6 describes the evaluation for this thesis, divided into quantitative analysis of our interactive process and qualitative discussion to model variability and commonality in terms and conditions.
- Chapter 7 concludes this thesis and discusses potential future work.
2. Related Work

For German tenancy law, (Landthaler et al. 2018) present an approach to link clauses from tenancy contracts to legal comments and practitioner books using the text similarity measures tf-idf and word embeddings, with the goal to support lawyers in contract drafting, editing and analyzing by providing relevant information for any text passage of interest.

(Waltl et al. 2017) introduce an active machine learning approach to classify legal norms in German statutory texts. They used a labeled set of sentences from the German tenancy law and, by iteratively training the models, showed improvements compared to regular supervised machine learning approaches.

Interactive machine learning (iML) was first introduced in (Fails and Olsen 2003), where it was used for image classification. (Amershi et al. 2011) demonstrate, that, by balancing need of the system and the user, an interactive process can significantly improve end-user interactions with web image search.

In the legal field, (Avelka et al. 2015) propose a framework for the relevancy assessment of statutory analysis, where human experts interact with machine learning text classification algorithms, that show improves performance compared to current approaches.
3. Basic knowledge

In this chapter, we lay knowledge foundation for this thesis. First, we introduce several concepts from the field of natural language processing, that help us in the classification process of paragraphs and titles from our collection of terms and conditions. Then, the concept of interactive machine learning is established. Lastly, we introduce variability modeling and more specifically feature models.

3.1 Natural Language Processing (NLP)

Natural Language Processing is an interdisciplinary research discipline, originating from areas like linguistics, computer science and psychology, concerned with analyzing natural languages, like German or English, and processing them in human-like manner. Natural language is generally unstructured and comes in different forms like text or spoken words. The field of NLP can generally be divided into two distinct areas: processing and generation of natural languages (Liddy 2001).

This thesis focuses on the processing of natural languages provided as written text in form of terms and conditions of online-shops. NLP provides many tools to implement our interactive paragraph and clause matching system. In the following sections, we provide theoretical background for

- the semantic text matching task,
- several preprocessing techniques and finally
- text similarity methods like word embeddings.

3.1.1 Semantic Text Matching (STM)

Semantic text matching describes the task of identifying logical or semantic links between text elements of one or more different documents. These types of tasks are especially relevant for domains, where compliance with rules defined in text form is mandatory for certain documents (Jörg Landthaler et al. 2018). In this thesis, we investigate two of these problems. First, we match two types of text fragments – paragraphs and clauses – of one document type, general terms and conditions of online shops (Figure 3.1.a). Then, we try to establish semantic links between clauses from terms and conditions of online shops and illegal types of clauses according the German Civil Code
3. Basic knowledge

(“Bürgerliches Gesetzbuch”) and rulings made by the German court system (Figure 3.1.b). The illegal types of clauses appearing in this thesis were provided by legal experts.

Figure 3.1: Illustration of Semantic Text Matching for our System

Source: Based on (Landthaler et al. 2018)

According to (Jörg Landthaler et al. 2018), the STM task consist of two sub-problems:

- **Segmentation**: Documents often need to be divided into smaller parts before text elements can be matched.
- **Matching**: Text elements need to be matched with semantically related text elements.

In this thesis, both sub-problems will be discussed. While the matching problem is our focus, segmentation is necessary due to the inconsistent format of general terms and conditions and will be explained in more detail in the next section.

3.1.2 Text Preprocessing techniques

Generally, data preprocessing is used to transform raw data into a usable format or to improve the quality of data by cleaning, integrating form several databases, transforming or reducing the volume of data. For text preprocessing, the goal is to convert unstructured text into a structured or semi-structured format by reducing unnecessary information and normalize the remaining text.
Following, we take a look at current preprocessing techniques presented by (Vijayarani and Ilamathi, 2015) and (Kannan and Gurusamy 2014):

**Tokenization:** Tokenization is the process of splitting strings of text into individual tokens like words and punctuation. Certain tokens such as punctuation, digits and symbols are also commonly removed in this step. The remaining words are then converted to lowercase. Tokens are generally needed as input for further processing like several text similarity approaches explained in chapter 3.1.3.

**Stop Words elimination:** Stop words are words that contain little or no meaning for text elements, but appear frequently, such as articles, pronouns and prepositions. This preprocessing step aims to remove such words, since they contribute little to the context or content of documents and improve processing time and memory usage by decreasing the vocabulary and therefore necessary calculation.

**Stemming:** Stemming is the process of identifying a word’s root by cutting of the beginning or end of the word using a list of common pre- and suffixes. For example, “construction”, “constructed” and “deconstruction” all share the root “construct”.

In addition to the tokenization and stop words removal, which were implemented within our system, the documents were manually segmented and edited in the following way to better fit our purpose:

**Segmentation:** Segmentation, as previously introduced, is the process of dividing documents into smaller parts. In our case, we want to distinguish between actual clauses and paragraphs (or paragraph titles) containing the clauses. As there are no regulations about the format of terms and conditions, variations between companies are significant. This makes manual parsing to mark paragraph titles and actual clauses much quicker than defining a complicated ruleset for the machine. Specifically, we add this string of characters “---” to the beginning of every paragraph title, since such a string is very unlikely to appear naturally at the beginning of any sentence. With these characters, our system can easily distinguish between clauses and paragraphs.

**Further manual edits:** Some additional edits were necessary to normalize our corpus of terms and conditions to improve comparability between semantically linked text elements. Specifically, the following modifications were made:
3. Basic knowledge

- The explanation for the right of withdrawal ("Widerrufsbelehrung") is often considered to be its own document, independent of the general terms and conditions. Since this is more of a description of the customers rights, and less about contract stipulations anyway, we replaced the entire explanation with the term "Widerrufsbelehrung". But: This only concerns the explanation itself. If companies have clauses about the right of withdrawal, these were kept as is they are (Compare Appendix A.1)

- The template to exercise the right of withdrawal is also replaced by the single term “Muster-Formular” since this is more of a template letter the customers can use to exercise their right.

- Most companies name their address within their terms and conditions. Those were replaced simply with “Anschrift”, since we do not care about their address, but simply that they are mentioned in the document.

- Finally, the general format of clauses had to be adjusted. Often, clauses would be divided over several rows, including lists, for higher readability. With such a format, our system would have problems determining where one clause ends and a new one starts. Therefore, all clauses were formatted in a way to only occupy on line. (Compare Appendix A.1)

Figure 3.2 shows the preprocessing steps we take for all terms and conditions in our dataset.

![Preprocessing Pipeline for Terms and Conditions](source: Own Illustration)

3.1.3 Text Similarity Methods

Semantic text matching problems are often tackled by using text similarity methods based on TFIDF or word embeddings (Landthaler et al. 2018). These techniques create vectors representations of numbers for words or entire documents. To determine the semantic relation between two words or documents, the similarity has to be calculated using some text similarity measure, where high similarity indicates semantic links. (H.Gomaa and A. Fahmy 2013) provide an extensive summary over text similarity approaches.
In this thesis, we use the cosine similarity as our base-case to determine the quality of our machine learning models, by comparing common scoring measure like recall, precision and f1-score for our classification predictions made by our models and based on the cosine similarity of our text elements.

Several text similarity approaches are introduced, of which the contextual and paragraph embeddings are implemented in our system:

- First, we summarize the TF-IDF score, which is often used as a weighting scheme, for example in research-paper recommender systems (Beel et al. 2016), then
- word embeddings are introduced, both as context-independent (Word2Vec) and contextual embeddings (ELMo), and finally
- we explain one paragraph embedding method (Doc2Vec), that is based on context-independent Word2Vec algorithm

**Term Frequency - Inverse Document Frequency (TF-IDF)**

The term frequency – inverse document frequency is a score to measure the importance of a term for a document in a corpus (Leskovec, et al. 2011). The TFIDF score is calculated as the product of two functions term frequency and inverse document frequency.

The term frequency $tf(t,d)$ measures, how often a term appears within a document. Since terms can appear more often in longer documents, the term frequency is generally normalized by dividing through the document length (Equation 3.1).

$$tf(t,d) = \frac{\text{Number of times Term } t \text{ appears in Document } d}{\text{Total number of terms in Document } d} \quad (3.1)$$

The inverse document frequency $idf(t,D)$ measures, how much information the word contains. Terms appearing in many documents, like stop words, provide little information and should therefore be weighted down. The calculation of the ids score is shown in Equation 3.2.
3. Basic knowledge

\[
idf(t, D) = \log \frac{\text{Total number of Documents in Corpus } D}{\text{Number of Documents containing Term } t}
\]  

(3.2)

**Context-independent word embeddings (Word2Vec)**

Word embeddings are learned vector representation of words mapped into a predefined vector space, where words with the same meaning have similar vector representations. One benefit of word embeddings is the higher density and lower dimensionality of vectors compared to more basic approaches like one-hot encoded vectors, where the dimension of any vector equals the total number of words in the vocabulary. The main benefit is the ability to generalize: if certain words have a similar context, their representations should capture this similarity (Goldberg 2017).

With Word2Vec, (Mikolov et al. 2013a) introduce a machine learning approach, that learns these representations of words more efficient than previous methods by using a two-layer neural network: First, words vectors are created from a simply algorithm. Then, a feedforward neural net language model is trained on these word vectors. Word representations created this way have shown some success in capturing both syntactic (e.g. the comparative word relationship, where “colder” is similar to “cold” in the same way “bigger” is similar to “big”) and semantic relations between words, like the “head of state to country”-relationship. These relationships can be achieved by performing basic algebraic operations on word vectors, for example \(\text{Merkel} – \text{Germany} + \text{Macron} = \text{France}\).

Word2Vec provides two model architectures shown in Figure 3.3 to create word representations: the continuous bag-of-words (CBOW) and the continuous skip-gram model.
Both models learn word embeddings by considering their local context. The CBOW model learns the current words’ embedding based on its context, whereas the Skip-gram model predicts the context of the current word. Here, the context is defined as a window containing neighboring words. For CBOW, they recommend a window size of five, while the Skip-gram should have a context window twice as big. Overall, while the CBOW is faster, the Skip-Gram model produces better results.

It is important to note, that the here mentioned context is only relevant during the word embedding creation. Consider these sentences as our only data: “I’m sitting on a bank in the park.” and “I’m going to the bank to withdraw money”. The vector for the word bank is created based on the context in both sentences, but the result is only one representation for the word, even though it has a completely different meaning in both sentences. Contextual word embeddings try to solve this problem known as polysemy, by computing different word representations for one word based on its context.

**Contextual word embeddings (ELMo)**

(Peters et al. 2018) introduce the concept of deep contextual word representations, called *Embeddings form Language Models* (ELMos). More precisely, ELMos are learned by a pre-trained, multi-layer, bi-directional language model.
Language models (LM) are used to predict the next word of a sentence based on their context, like the CBOW architecture from Word2Vec. Bi-directional models are trained forward and backward on sentences. Let’s take part of a sentence from the previous section as input: For “I’m sitting on a “ the forward model should output high probabilities for potential word like “bank” or “chair”. For the backwards model, the sentence order will be reversed: “bank a on sitting ”. The model should predict a word like “am” or “was”. Additionally, multi-layer Long-Short-Term-Memory (LSTM) is used, where one LSTM-layer takes the output of the previous layer as input.

Specifically, (Peters et al. 2018) use 2 bi-directional LSTM layers with 4096 units and 512 dimension projections with a residual connection between both layers. In total, three representations for each word are created, the output of both layers as well as the original input. ELMos require context-independent word embeddings as input, which need to be created beforehand, which makes it easy to adapt existing systems using context-independent embeddings by using those as the input for the first ELMo layer. The authors use character convolutional neural network (CNN) to create the required input representations. The final contextual word embeddings are then created by combining the three representations applying weights that are learned to each.

The power of ELMos is shown by achieving new state-of-the-art results in the following six diverse NLP problems: Question answering, Text entailment, Semantic role labeling, Coreference resolution, Named entity extraction, Sentiment analysis.

In this thesis, contextual word embeddings are the basis for our interactive matching process. A pre-trained model provided by4 is used to create these word embeddings.

**Paragraph embeddings (Doc2Vec)**

Unlike word embeddings, paragraph embeddings are learned vector representations of text passages (or entire documents), where paragraphs with the same meaning have similar vector representations.

(Le and Mikolov 2014) first proposed the Paragraph Vector (PV) model, an unsupervised algorithm, that learns vector representations of paragraphs or documents regard-
3. Basic knowledge

less of their length. This method is called Doc2Vec and builds on the previously introduced word embeddings algorithm Word2Vec. Figure 3.4 shows the two proposed architecture models to learn paragraph vectors. Like word embeddings, paragraph vectors can achieve the similar results when performing basic algebraic operations (Dai et al. 2015).

Figure 3.4: The Distributed Memory model (PV-DM) left and the Distributed bag of words (DBOW) model right  
Source: (Le and Mikolov 2014)

The distributed memory (PV-DM) model extends CBOW by adding the paragraph vector $D$ to its context. It acts as the topic or memory about what is missing from the current context. As in Word2Vec, a sliding window is used for the local contexts. The paragraph vector $D$ is shared between all local contexts of one paragraph, whereas the word vectors $W$ are shared across paragraphs, which means word representations stay context independent.

The second introduced architecture is the distributed bag of words (PV-DBOW) model. This method is similar to the Skip-gram model, but instead of predicting the current context based on a word, it is predicted on the paragraph vector $D$. (Le and Mikolov 2014) recommend combining both methods for consistently strong results but say PV-DM usually performs well on its own for most tasks.

3.2 Interactive Machine learning

Interactive Machine Learning describes the collaboration between machine and human agent to achieve certain tasks like classification or clustering of data points or finding interesting data projections. Such an interactive machine learning system generally consists of an automated component, a user interface and a learning algorithm, where the human agent interacts with the automated service via user interface by providing itera-
Basic knowledge

tive feedback to the learning algorithm and receives feedback themselves after every step. According to (Boukhelifa et al. 2018), human feedback can be either explicit or implicit, where

- explicit feedback is provided through a suitable user interface with the human’s knowledge, that their assessment is used in further learning algorithms (which is the type of feedback we gather by developing our matching system),
- while by giving implicit feedback, the human does not realize the relevance of their feedback for the system only trying to complete their own task.

The reasons to for interactive machine learning are varied: domain expert knowledge, that is hard to encode, can be integrated into the system, Uncertainties that appear in regular machine learning can be resolved more easily, and, by involving humans in the learning loop, trust into the results may be improved (Boukhelifa et al. 2018). Another advantage is displayed in Figure 3.5, which shows a major difference between traditional and interactive machine learning processes: learning speed.

![Figure 3.5: Difference between Traditional and Interactive Machine Learning](source: Amershi et al. 2014)

While in traditional machine learning, the iteration cycles and therefore the model improvement is slow, an interactive approach reduces the length of iteration cycles drastically, which lets the model improve in smaller, but quicker steps. By constantly providing feedback to the system, the machine learning behavior can be directed through simple trial and error, which allows users even with little knowledge or background in machine learning to improve the model (Amershi et al. 2014).
3.3 Variability Modeling

Variability modeling describes the area of representing commonality and variability of products in product lines. It is an important tool for software product line engineering both in practice and research. Variability models are useful in many steps of a product line life cycle, such as planning and scoping of product lines, helping developers by summarizing available features and in marketing.

According to (Berger et al. 2013), in industrial practice variability models are mainly used to fit existing products into a product line, followed by evolving single products into a product line in a reactive manner. The proactive method of planning product lines before any product is developed is used more rarely in the industrial practice. Variability Models have many different notation forms, like Spreadsheets, UML-based approaches or simply free text descriptions. The most commonly used notation is feature models, which will be introduced in more detail.

Feature Modeling

Feature modeling was first introduced in (Kang et al. 1990) as feature-oriented domain analysis (FODA) and has since been widely accepted to model commonality and variability on software product line engineering. High abstraction of features has proven as helpful in communication between stakeholder of different backgrounds in a product line, were features are often described and capabilities, that are delivered to costumer, product configuration or product management for differing markets.

FODA is a simple model containing several elements: features, that are either mandatory, optional, alternative or mutually dependent on other features (inclusion/exclusion); relationships are either compositions, generalization or specialization and modeled by AND/OR graphs; and textual descriptions of feature attributes and selection reasons for optional features (Kang and Lee 2013). After feature modeling was introduced, many papers extended the FODA model by adding new elements and relationship types, such as cardinality (Czarnecki et al. 2005) and refining the generalization/specialization relationships to OR- and XOR-specializations (van Gurp et al. 2001). In Figure 3.6, a feature model for a mobile phone product line is shown containing all previously introduced elements, excluding the cardinality.
3. Basic knowledge

In Figure X, the mandatory features for all phones in this product line are the ability to make/take calls and a screen, with the alternatives of basic, colour and high resolution. These are XOR-specializations of the more general screen, where only one can be chosen. Optional features are GPS and media, with the OR-specializations camera and MP3. In order to choose the camera as a feature for this mobile phone, it requires a high-resolution screen. GPS on the other hand can only be used, if the screen is not the basic version.

As discussed, variability modeling is mostly used to model commonalities and variabilities of products in a software product line. In this thesis, we adapt the feature modeling techniques for legal contracts. Our goal is to display the most common paragraphs and clauses of German-language terms and conditions, outline alternative versions of clauses and show examples of unlawful clauses.
4. Requirements and Design

This chapter explains the preliminary steps for the development of a matching system to interactively analyze a corpus of terms and conditions, using embeddings and several supervised machine learning approaches to find semantic links between clauses and paragraphs from different terms and conditions.

Section 4.1 discusses the functional and non-functional requirements that shall be implemented in our matching system. Based on the defined requirements, the system is designed in Section 4.2. Lastly, the data model for our system is introduced in Section 4.3.

4.1 Requirements

In this section, the requirements for our interactive matching system are defined. They are divided into functional and non-functional requirements. Section 4.1.1 discusses the functional, Section 4.1.2 the non-functional requirements.

4.1.1 Functional Requirements

In the following section, the functional requirements for our interactive matching system are introduced:

1. The matching system shall provide a user interface to add new terms and conditions to our database.
2. The manual pre-processing steps for a new document shall be explained comprehensively.
3. The system shall predict the class for all paragraphs/clauses.
4. The system shall provide different methods to make these predictions.
5. The system shall provide the interface for a user to correctly classify paragraphs and clauses based on the predictions.
6. The user shall receive feedback over the quality of different methods of predictions creation.
7. The system shall provide an overview over paragraphs and clauses with semantic links.
4. Requirements and Design

4.1.2 Non-Functional Requirements
After the functional requirements are defined, this section discusses the non-functional requirements for our system:

1. The system's front-end shall be implemented using a modern web framework.
2. The back-end shall be implemented in a programming language providing advanced libraries for machine learning tasks.
3. For communication between front- and back-end, a REST API shall be implemented.

4.2 Design
In this section, the target software architecture is designed based on the previously defined requirements. Afterwards, an overview over the interactive process between the user and the system to classify paragraphs and clauses is provided. Finally, the REST API routes are introduced.

4.2.1 Software Architecture
The platform for our matching system is designed to run remotely, for two main reasons: constant availability and processing power. While availability is important to provide constant access to the user, powerful hardware may be required for computationally expensive machine learning algorithms, especially for the growing data set.

Figure 4.1 depicts the deployment diagram for our interactive matching system. It follows a 3-tier architecture common for web applications consisting of the back-end MatchingServer, front-end MatchingClient and database MatchingDB.
4. Requirements and Design

![Figure 4.1: UML Deployment Diagram for our Matching System](Source: Own Illustration)

The front-end *MatchingClient* is a React application and communicates to the back-end via REST API. The core views and components of our application are shown in Figure 5.1. The back-end *MatchingServer* is a Python based server using Flask and implements the business logic of our system. It is responsible for learning embeddings and training machine learning models, as well as providing data to the front-end through defined REST API routes. The *MatchingDB* is a relational SQLite database, that is connected to the back-end through Flask-SQLAlchemy, and stores are relevant data based on the structure of the data model discussed in section 4.3.

4.2.2 Interactive Process

This section discusses the interactive process between user and system in more detail. We model a use case diagram to discuss the general functionality of our system, before going into more detail about to process of adding and matching terms and condition in form of a sequence diagram.

General System Overview

As Figure 4.2 shows, our system consists of three main use cases, in which the user interacts with our system: adding terms and conditions, matching them, and evaluating the results.
4. Requirements and Design

The use case *Adding Terms and Conditions* is the starting point in our system and contains several smaller use cases. Since our system should help in classifying paragraphs and clauses, they need to be extracted from a given document. The extraction and creation of the different text fragments also requires applying several preprocessing techniques. Learning word embeddings for newly added paragraphs/clauses is included in this main use case as well. Finally, making predictions about the class of a paragraph or clause is extending the use case, based on the user’s decision for reasons like efficiency in training models predicting for several terms and conditions at once. The functional requirements 1 to 4 will be implemented in this use case.

The central use case of our system is the *Matching of Terms and Conditions*. The classification of paragraphs is part of this use case. The process if classifying clauses is an extension to this use case and can only be activated, when the *Classifying Paragraphs* case for the respective documents is already done. The use case aims to answer the functional requirement 5.
4. Requirements and Design

The final major use case is the evaluation of the interactive matching system in quantitative and qualitative manner. With this use case, the functional requirements 6 and 7 shall be implemented.

Workflow between User and System

This section provides a more detailed view at the interaction between user and system to semantically classify paragraphs and clauses from a set of terms and conditions. The workflow shall provide more insight into required methods to implement this system and present necessary REST API routes. Figures 4.3 and 4.4 illustrate the sequence of interactions between user and the three layers of our application for adding (Figure 4.3) and matching (Figure 4.4) new terms and conditions.

![UML Sequence Diagram for adding Terms and Conditions](image)

**Figure 4.3: UML Sequence Diagram for adding Terms and Conditions**

*Source: Own Illustration*

The process of *Adding Terms and Conditions* starts with the user inputting terms and conditions into the relevant field in the user interface. The front-end segments the input into several text fragments and sends them to the back-end server. The segments are sorted into paragraphs and clauses, and table entries in our database are created for all objects. Afterwards, embeddings for all previously created text object are learned and added to the database. In the end, text similarity is calculated, and models are trained to make class predictions for paragraphs and clauses, before the back-end notifies the front-end, that calculations are done, and the user is redirected to the main page.
The process of *Matching Terms and Conditions* begins with the user choosing the terms and conditions, that should be classified. The following steps are done twice, first for paragraphs and then clauses. The front-end asks for the underlying method for the predictions and, after getting an answer, redirects the user to the matching view. Then, the predictions are requested from the back-end that accesses the database for them and returns the objects. Afterwards, the text fragments, that the predictions were made for, are fetched through the REST API and from our database. The text fragments are then sorted into classes based on their respective predictions and displayed to the user. The user can move text between classes, until each text fragment is sorted into the correct class. When the user is finished with the matching process, the classes with their text elements are sent to the back-end and the objects in the database are updated based on the classification.
4. Requirements and Design

4.3 Data Model

This section introduces the underlying data model for the interactive paragraph and clause matching system, implemented in the thesis. The class diagram in Figure 4.5 provides the conceptual structure.

![Class Diagram]

**Figure 4.5: Data Model for the interactive matching system**
*Source: Own Illustration*

*Terms and Conditions* is the central class in our system, since their interactive analysis is the focal point of this thesis. This class contains information about the company. Each *Paragraph* only ever belongs to one instance of *Terms and Conditions*, and one *Clause* is always part of one *Paragraph*. Both inherit their attributes from the abstract base class *Text Element*. Whenever a new *Terms and Conditions* object is created, a *Vector Representation* in form of a paragraph embedding is created for each *Text Element*. *Methods* are certain similarity measure or machine learning algorithms used to create *Predictions* about the class a *Paragraph* or *Clause* belongs to.

Since there is no publicly available collection of general terms and conditions in the German language, especially not in labeled form for our needs, we had to collect our own dataset, as explained in Section 1.3.2.
5. Implementation Matching System

In chapter 4, we discussed the requirements and our design strategy for the implementation of our interactive matching system. In this chapter, we describe the actual implementation for our system. In section 5.1, the main features of our front-end will be presented, before we explain the development of our back-end in section 5.2.

5.1 User Interface

The development of our front-end is discussed in this section. Figure 5.1 provides an overview over all implemented views with their respective components. Overall, seven views were implemented with a total of thirteen components. The general elements are based on a provided template and were adjusted to fit the needs of this thesis. In the following sections, these views are described in more detail.

![React View and Components](source.png)

**Figure 5.1:** React view and components for front-end application

*Source: Own Illustration*
5. Implementation Matching System

5.1.1 AGBListView

The *AGBListView* is the entrance point to our application and provides an overview over all online shops, whose terms and conditions are part of our dataset. The user can see the current classification status for paragraphs and clauses for all terms and conditions and decide which document to classify next. Figure 5.2 shows the screenshot of that list.

![Figure 5.2: Overview of all terms and conditions in our system](Source: Own Screenshot)

By clicking on the arrow, the user moves on to the classification process of that document. He must choose a model to provide the predictions between several options. Re-classification based on previous labeling is also possible. Automatically, paragraphs must be classified before clauses, since they provide valuable information for the clause-matching process. Figures 5.3 provides the screenshot for the different options.

![Figure 5.3: Overview over model selection for predictions](Source: Own Screenshot)
5. Implementation Matching System

5.1.2 AGBFormView

This view allows the user to add terms and conditions for a new online-shop to our system. It provides a summary about the required format and necessary manual preprocessing steps described in chapter 3. The AGBFormView consists of two tabs: **Text Input** is shown in Figure 5.4 and provides the interface to add terms and conditions as well as reading the preprocessing summary by clicking on the question mark.

Figure 5.4: Text Input tab from AGBFormView  
*Source: Own Screenshot*

The **Check Format** tab allows for a format check, by providing a preview on how the whole text is segmented into independent paragraphs and clauses. Figure 5.5 shows a screenshot of segmented paragraphs and clauses.

Figure 5.5: Check Format tab for AGBFormView  
*Source: Own Screenshot*
5. Implementation Matching System

5.1.3 ParagraphMatchingView

The ParagraphMatchingView is one of the core features of this application. This view helps in matching paragraphs of terms and conditions with our ground truth. The left part of the screen contains all paragraphs from our ground truth as well as their class. On the right side, paragraphs from the current company are sorted based on the prediction of the chosen model in the AGBListView, as we can see in Figure 5.6.

![Figure 5.6: Currently dragged paragraph in ParagraphMatchingView](Source: Own Screenshot)

Each white box containing a paragraph is a draggable object. The slightly darker grey area surrounding the three paragraphs on the right is a drop area (droppable). One droppable exists for each class and can contain any number of draggable objects. Each draggable object can be moved between the droppables or to a different position in the current droppable. Moving any draggable object outside of a valid drop area will return it to its previous position. The internal state containing the position of all draggable objects is updated, whenever the user lets go of the object.

5.1.4 ClauseMatchingView

The ClauseMatchingView provides the same basic functionality as the ParagraphMatchingView, moving of Clauses to the semantic equivalent of the ground truth. This view also contains information obtained through the paragraph classification to simplify this process. Figure 5.7 shows a screenshot of the clauses belonging to the same terms and conditions as the paragraphs in Figure 5.6.
5. Implementation Matching System

Figure 5.7: Predicted Classes for Clauses in ClauseMatchingView
Source: Own Screenshot

The basic structure is the same as that one of the ParagraphMatchingView. The left side contains clauses from our ground truth and on the right clauses are sorted based on their predicted class. But here, both sides provide additional information besides the raw text. As seen in the data model in Figure 4.5, each clause belongs to exactly one paragraph. By matching paragraphs before clauses, we can display the true class of the parent paragraph for each clause. This helps the user to identify the correct class much quicker by providing the approximate topical area for any clause.

5.1.5 AGBDetailView

The AGBDetailView can be reached by clicking on any company name in the AGBListView. This view provides an overview over the terms and conditions of the current online-shop. When paragraphs and/or clauses for this company were already labeled, clicking on any paragraph title or the arrow next to a clause, the user is routed to the ClassDetailView. Figure 5.8 shows a screenshot for an already classified document.

Figure 5.8: Overview over paragraphs and clauses from a chosen online-shop
5. Implementation Matching System

5.1.6 ClassDetailView

The ClassDetailView displays all clauses or paragraphs belonging to a certain class. This view helps in finding potentially mismatched clauses, as well as providing valuable insight into which types clauses or paragraphs didn’t fit into the classes based on the ground truth, but were matched into one of our “miscellaneous” classes. Looking into these classes is an important part of the qualitative evaluation to build the variability model. Figure 5.9 shows a screenshot for this view.

![ClassDetailView](source: Own Screenshot)

Figure 5.9: ClassDetailView with paragraphs of a class
Source: Own Screenshot

5.1.7 DataView

The DataView provides diagrams with the class distribution for paragraphs and clauses over our entire dataset. Specifically, two diagram types are displayed: the total number of instances in each class, and the number of companies, that have at least one clause of a certain class in their terms and conditions. Hovering over any bar in the diagram will display the number of entries in that class, while clicking on it will lead to the ClassDetailView for the respective class. These diagrams build the foundation of our qualitative evaluation in chapter 6. Figure 5.10 shows a screenshot of this view.

![DataView](source: Own Screenshot)
5. Implementation Matching System

5.1.8 General Components

Several general components build the basic layout foundation of our web application. They are part of most views and contain the footer, header as well as the menu in the top right corner.

Figure 5.10: Screenshot of the DataView
Source: Own Screenshot
5. Implementation Matching System

5.2 Back-end

After explaining our front-end implementation, this section discusses the development of our back-end. The back-end accomplishes two main tasks: (a) providing relevant data to the front-end via REST API and (b) the evaluation of our models. Section 5.2.1 explains the implementation of the data model. The back-end implementation of the creation of terms and conditions and predictions is discussed in Section 5.2.2. Section 5.2.3 discusses the important routes for the actual classification process, before an overview over REST API routes implemented for the remaining views is provided in Section 5.2.4.

5.2.1 Data Model Implementation

This section deals with the implementation of the data model introduced in Figure 4.5. The code excerpt in Listing 5.1 exemplifies the definition of the two classes Terms and Conditions and Paragraph.

```python
1. class Agb(db.Model):
2.     id = db.Column(db.Integer, primary_key=True)
3.     name = db.Column(db.String(250))
4.     clauseIsLabeled = db.Column(db.Boolean)
5.     paragraphIsLabeled = db.Column(db.Boolean)
6.
7. class Paragraph(db.Model):
8.     id = db.Column(db.String, primary_key=True)
9.     title = db.Column(db.String)
10.    tokenText = db.Column(db.String)
11.    trueState = db.Column(db.Integer)
12.
13.    agb_id = db.Column(db.Integer, db.ForeignKey('agb.id'))
14.    agb = db.relationship('Agb', backref='paragraphs')
15.
16. class AgbSchema(ma.ModelSchema):
17.    class Meta:
18.        model = Agb
19.
20. class ParagraphSchema(ma.ModelSchema):
21.    class Meta:
22.        model = Paragraph
23.
24.    agb_schema = AgbSchema()
25.    agbs_schema = AgbSchema(many=True)
26.
27.    paragraph_schema = ParagraphSchema()
28.    paragraphs_schema = ParagraphSchema(many=True)
Listing 5.1: Excerpt of exemplary Class Definition
```

The Agb class implements the Terms and Conditions from our data model and could also be considered the company, we obtained the terms and conditions from. Each Agb has an ID, which represents the primary key of this class. The name is the company
name. The attributes `clausesLabeled` and `paragraphsLabeled` describe whether the clauses or paragraphs of this company have already gone through the matching process, where `paragraphsLabeled` has to be true before clauses can be classified. The `Paragraph` class is uniquely identified by an `ID` composed of their `Agb`'s `name` and a running number. `Title` contains the raw text, `tokenText` the tokenized form of that text and `trueState` represents the class, this `Paragraph` belongs to. Since each `Paragraph` belongs to one `Agb`, they also have their `Agb ID` as a foreign key. Line 15 establishes the relationship to `Agb` with the attribute `paragraphs` added to `Agb`. Not shown but defined in other classes are the additional attributes of `Paragraphs` like `Clauses`, `Vectors` and `Predictions`. Listing 5.2 shows an exemplary JSON of a `Paragraph` object with three clauses, several predictions made for the object and one vector representation next to the previously introduced attributes. To dump or load any of the objects, the schema class for each class must be defined. In the end, two instances of the schema are initialized to either handle one or many objects of the related class.

```json
1. {
2.   "agb": 18,
3.   "clauses": [
4.     "saturn_3",
5.     "saturn_4",
6.     "saturn_5"
7.   ],
8.   "id": "saturn_2",
9.   "predictions": [
10.  865,
11.  9037,
12.  10798,
13.  10812,
14.  10826,
15.  10840,
16.  10854
17. ],
18.   "title": "2 Vertragspartner",
19.   "tokenText": "Vertragspartner",
20.   "trueState": 0,
21.   "vectors": [
22.     15547
23.   ]
24. }
```

Listing 5.2: Example JSON of a Paragraph Object

### 5.2.2 Workflow

This section discusses the implementation of the workflow introduced in Figures 4.3 and 4.4 in three steps: (1) the creation of `Agb`, `Paragraph` and `Clause` objects and pre-processing steps is explained, (2) the implementation to learn vector representations through ELMo is discussed, and (3) several approaches to create our predictions are
shown. The focus, especially in steps (2) and (3), lies on the necessary steps in order to classify paragraphs through the user interface, but the process for clauses is comparable.

**Adding Terms and Conditions**

Listing 5.3 shows the REST API route, that creates new terms and conditions from the information send by the *AGBFormView*. The following code is only run, when the method from the http request is a POST. A new *Agb* object is created with the *name* defined in the body of the request. The *splitText* attribute contains an array of all text elements as seen in the *Check Format* tab in the view. Each text element is defined as either *Paragraph* or *Clause*, by checking if the string starts with “---”, the paragraph identifier added in the preprocessing steps.

```
1. @app.route("/newAGB", methods=["POST"])
2. def add_agb():
3.     name = request.json['name']
4.     clauses = request.json['splitText']
5.     new_agb = Agb(name = name, clauseIsLabeled=False, paragraphIsLabeled=False)
6.     db.session.add(new_agb)
7.     db.session.commit()
8.     for counter, clause in enumerate(clauses):
10.            new_ParagraphID = name + "_" + str(counter)
11.            new_paragraph = Paragraph(id= new_ParagraphID, title= clause[3:], agb_id = new_agb.id)
12.            db.session.add(new_paragraph)
13.            db.session.commit()
14.        else:
15.            new_ClauseID = name + "_" + str(counter)
16.            new_clause = Clause(id= new_ClauseID, rawText = clause, agb_id= new_agb.id, paragraph_id = new_paragraph.id)
17.            db.session.add(new_clause)
18.            db.session.commit()
19.            token_and_sim.tokenize_text(new_agb.id)
20.    return agb_schema.jsonify(new_agb)
```

Listing 5.3: Excerpt for creating *Agb*, *Paragraphs* and *Clauses*

After setting the attributes, the objects are added to the current data base session and then committed. The *paragraph_id* for clauses is always the last paragraph added, since the general structure of terms and conditions has paragraphs as the headline for upcoming clauses. Unlike the conceptual data model in chapter 4, *Clauses* also know about their parent *Terms and Conditions* and not just their *Paragraph* to make calling all claus-
es for one company more efficient. After committing the newly created objects, paragraphs and clauses are tokenized and cleaned up with the `tokenize_text` function. Listing 5.4 details the implementation.

First, the relevant `Agb` is queried. To tokenize text, a German language model provided by spaCy is used. spaCy is an advanced NLP library, developed for industrial use, with features like lemmatization, POS-tagging and stop word check. This implementation uses the tokenization, stop word and empty space removal feature of spaCy. Additionally, punctuation and digits are filtered out from paragraphs and clauses. Since the flask-sqlalchemy library doesn’t allow for arrays as data types, all token have to be joined as one string, before the object attributes can be updated.

Line 10 displays an example as to why this implementation introduced the direct relation between `Agb` and `Clause`. Instead of looping through all paragraphs to get all clauses, they can directly be accessed from the `Agb` class. This possibility is used commonly throughout the implementation.
Learning Vector Representations

Listing 5.5 show an excerpt for the learning of ELMo's. A pre-trained German language model is used to learn contextual word embeddings for clauses and paragraphs.

```python
1. def create_meanVector_cleanedText(id, e):
2.     e = Embedder('..\..\..\..\142')
3.     agb = server.AgB.query.get(id)
4.     create_vectors_for = [agb.clauses, agb.paragraphs]
5.     # Checking if we're currently working on clauses (True) or paragraphs(False)
6.     flag_forClauses = True
7.     for clause_or_paragraph in create_vectors_for:
8.         sentence = list(map(lambda paragraph : paragraph.tokenText.split(' '), clause_or_paragraph))
9.         result = e.sents2elmo(sentence)
10.        vectorList = []
11.       #create meanVector here
12.       for counter, wordVectors in enumerate(result):
13.           data = np.array(wordVectors)
14.           average = np.average(data, axis=0)
15.           vectorList.append(average)
16.       for counter, c_OR_p in enumerate(clause_or_paragraph):
17.           meanVector = list(map(str, vectorList[counter]))
18.           if flag_forClauses == False:  
19.               new_meanVector = server.Vector(vector=meanVector, paragraph_id=c_OR_p.id, meanVector=True)
20.               server.db.session.add(new_meanVector)
21.               server.db.session.commit()
22.         else:
23.             # The same for Clauses + saving every Word Vector for Clauses ###
24.             flag_forClauses = False
```

Listing 5.5: Creating ELMos for Paragraphs

First, the embedder and the current Agb are loaded. Since ELMo learns representations for each word from its current context, the required input is a tokenized sentence or list of tokenized sentences. In line 10, the token string for each Paragraph (or Clause) of this Agb is converted to a list with help of the `map()` function. The embedder returns 1024-dimensional word embeddings for every word of every sentence, calculated from the average of three word representations created by the LSTM layers. Afterwards, a vector representation for a Paragraph is created by averaging all word vectors of that Paragraph and commit to the database. For Clauses both, the single word vectors and one average representation were saved originally, but word vectors were later dismissed for several reasons discussed in the evaluation part of this thesis.

To later evaluate the quality of ELMo’s for the semantic text matching problem, document embeddings from Doc2Vec are implemented as comparison. In contrast to ELMo,
where a pre-trained model was used, our own language model is trained based on all paragraphs/clauses currently in the system. Listing 5.6 shows an excerpt of the implementation.

```python
def build_model(data):
    tagged_data = []
    for i, doc in enumerate(data):
        tagged_data.append(TaggedDocument(words=doc[0], tags=[doc[1]]))
    model = Doc2Vec(vector_size=256, dm=1)
    model.build_vocab(tagged_data)
    for epoch in range(100):
        model.train(tagged_data, total_examples=model.corpus_count, epochs=model.iter)
    model.save("d2v_p256.model")
```

Listing 5.6: Excerpt of example Doc2Vec model creation

Doc2Vec requires *TaggedDocuments* as input. They are a combination of tokenized sentences and a label, in this case the paragraph or clause ID. The label represents the paragraphs vector as explained in section 3.1.3. The above example trains document embeddings with 256 dimensions based on the PV-DM architecture over 100 iterations and is then saved to make it accessible for later evaluation. Listing 5.7 shows, how to get a document’s embedding of paragraphs used for training the model, as well as how to infer a representation for new paragraphs by providing the text in tokenized form.

```python
d2v_model = Doc2Vec.load("d2v_p256.model")
old_embedding = d2v_model.docvecs[old_paragraph.id]
new_embedding = d2v_model.infer_vector(new_paragraph.tokenText)
```

Listing 5.7: Excerpt for creating Document Embeddings for new and old Paragraphs

**Prediction Creation**

Predictions based on the commonly used cosine similarity measure provide the baseline for our application. Listing 5.8 shows an excerpt for calculating the similarity for paragraphs. This function is called whenever the REST API Route from Listing 5.3 is requested.

```python
def predict_similarity(paragraph1, paragraph2):
    similarity = cosine_similarity(paragraph1, paragraph2)
    return similarity
```
5. Implementation Matching System

```python
def highest_similarity_paragraphs(id):
    base_agb = server.Agb.query.get(1)
    agb = server.Agb.query.get(id)

    for paragraph in agb.paragraphs:
        similarityVector = []
        wordVector = server.Vector.query.filter_by(paragraph_id=paragraph.id).first()
        arrayVector = convert_to_Array(wordVector.vector)

        for base_paragraph in base_agb.paragraphs:
            base_paragraph_wordVector = server.Vector.query.filter_by(paragraph_id=base_paragraph.id).first()
            arrayBaseVector = convert_to_Array(base_paragraph_wordVector.vector)
            similarity = 1 - spatial.distance.cosine(arrayVector, arrayBaseVector)
            similarityVector.append(similarity)
        my_prediction = similarityVector.index(max(similarityVector))
        new_prediction = server.Prediction(predictedState=my_prediction, paragraph_id=paragraph.id, method_id=1, agb_id=id)
        server.db.session.add(new_prediction)
        server.db.session.commit()
```

Listing 5.8: Predictions for Paragraphs based on Cosine Similarity

The cosine similarity is always determined between new Paragraphs and Paragraphs of the ground truth, here called base_agb. For every new Paragraph the vector representation is fetched and the cosine similarity with every Paragraph from the ground truth is calculated. The class of the most similar vector is then set as the predictedState of the Prediction for our new Paragraph.

Besides the cosine similarity measures, several supervised machine learning approaches are used to predict the class of any Paragraph or Clause, to provide the user with more precise methods. Listing 5.9 shows the models used in this system. Because our system starts only with the ground truth as labeled data, several common models were tried and later evaluated.

```python
def set_models():
    models = []
    models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
    models.append(('LDA', LinearDiscriminantAnalysis()))
    models.append(('KNN', KNeighborsClassifier()))
    models.append(('CART', DecisionTreeClassifier()))
    models.append(('NB', GaussianNB()))
    return models
```

Listing 5.9: Machine Learning Models used for classification

The main approach during matching was the “Most Unique Classes” method, were higher distribution of Paragraphs or Clauses over the available classes was considered better. Listing 5.10 shows the implementation for training the models and predicting paragraph classes, as well as the definition of “Most Unique Classes”.
5. Implementation Matching System

First, a list containing the IDs of all Agbs with classified Paragraphs is mapped to paragraph_ids. The function create_training_set_for_paragraphs() then creates the training set from this list containing vector representations and the true class of all labeled Paragraphs. Afterwards, the Vectors for all Paragraphs from the new Agb are collected in create_test_set_for_paragraphs(). Each model is then trained to predict classes for new Paragraphs and those Predictions are committed to the database

In find_best_prediction() the approach – either one of the models or cosine similarity - predicting “Most Unique Classes” is determined. Two variables are introduced: most_unique holds the number of different predicted classes, max_instances the highest number of predictions for any class. First, we check if the current method has more unique classes than the current best. If True, the variables are updated accordingly. If these values are identical, most_instances are compared, where the smaller value determines the better method. The assumption is, that four classes being predicted twice is
more realistic for terms and conditions than one class being predicted five times with the three other classes only predicted once.

While all these predictions can be created for **Paragraphs** and **Clauses** whenever new terms and conditions are created, the actual process differs slightly and is explored more in depth in the evaluation chapter.

### 5.2.3 Routes for Classification process

In this section, we discuss how relevant data for both the paragraph and clause matching process are provided to the front-end. The section divides into two parts: (1) the relevant routes to fetch **Predictions** and **Paragraphs** are discussed, and (2) the **trueState** is set based on the matching done in the **ParagraphMatchingView**.

#### Providing requested Predictions

As seen in the **AGBListView**, the user can choose between one of several methods of prediction creation for the matching process. Listing 5.11 shows the implementation.

```python
1. @app.route("/predictions/<string:type>/<int:agb_id>/<int:method_id>", methods=['GET'])
2. def get_prediction_forAGB(type, agb_id, method_id):
3.     if type == "paragraph":
4.         all_predictions = Prediction.query.filter_by(method_id=method_id).filter_by(agb_id=agb_id).filter_by(clause_id=None)
5.     elif type == "clause":
6.         all_predictions = Prediction.query.filter_by(method_id=method_id).filter_by(agb_id=agb_id).filter_by(paragraph_id=None)
7.     return predictions_schema.jsonify(all_predictions)
Listing 5.11: Excerpt to provide Predictions to Front-End
```

This route is responsible for providing **Predictions** both for **Paragraph** and **Clause** matching depending on the defined **type**. It simply filters all predictions by the **agb_id** of the terms and conditions we want to classify and by the **method_id** of the model the user chose through the user interface and returns them jsonified to the front-end. To match any text elements, the back-end first needs to provide them to the front-end through the route shown in Listing 5.12. This route needs to be called twice, once to fetch the ground truth and once for the paragraphs/clauses of the current terms and conditions.
5. Implementation Matching System

Listing 5.12: Excerpt of REST API route to fetch paragraphs

This is a simple filtering of the Paragraph class for all paragraphs belonging to the relevant Agb. Only the attributes required for the matching process are loaded with the `with_entities()` function. Loading without the restriction of attributes always loads all related objects, which drastically increases the loading time. The big disadvantage of `with_entities()` is the inability to load foreign keys, which complicates the next step. The route to fetch Clauses is comparable.

These routes provide all relevant information to classify paragraphs, but the matching of clauses is supported by information about their respective paragraphs. The route in Listing 5.13 is requested for every clause of the ground truth and the to-be-classified clauses.

Listing 5.13: Excerpt to fetch Paragraphs of relevant Clauses

Since the paragraph_id couldn’t be sent in the previous step, the requesting of paragraphs cannot simply be done by using the `Paragraph.query.get(paragraph_id)` function but takes the additional step of first getting the Clause again and then using their paragraph_id.

Setting true Class for Paragraphs or Clauses

After the user has classified the Paragraphs/Clauses in the user interface, the trueState for each object is set and, if this was the first classification and not simply a correction, “true predictions” are created, that let the user easily correct classification mistakes without having to re-match all other elements based in some model. Listing 5.14 shows an excerpt of this implementation for Clauses.
5. Implementation Matching System

1. @app.route("/setTrueState/<string:type>", methods=["PUT"])
2. def set_trueState(type):
3.     classes = request.json['classes']
4.     agbid = request.json['agbid']
5.     agb = Agb.query.get(agbid)
6.     for counter, entries in enumerate(classes):
7.         for item in entries:
8.             if type == "clause":
9.                 entry = Clause.query.get(item['id'])
10.                if agb.clauseIsLabeled == True:
11.                     true_prediction = Prediction.query.filter_by(method_id=9).filter_by(clause_id = entry.id).first()
12.                     true_prediction.predictedState = counter
13.             else:
14.                 db.session.add(new_prediction)
15.                 db.session.commit()
16.     if type == "paragraph":
17.         entry.trueState = counter
18.     db.session.commit()
19.     if type == "clause":
20.         agb.clauseIsLabeled = True
21.     if type == "paragraph":
22.         agb.paragraphsLabeled = True
23.     db.session.commit()
24.     return paragraphs_schema.jsonify(classes)

Listing 5.14: Excerpt to set true Class for Clauses

The body of the http PUT-Request contains two elements: the ID of the Agb that was matched, and the distribution of Clause objects over all classes. We loop through every class. For every Clause, that was matched to one class, we get() it from our database and set the trueState to the class, here represented by the counter. If the Clauses of this Agb have been classified before, the respective Prediction representing the true class will be updated as well. Otherwise such a Prediction is newly created, to allow for later corrections. For Paragraphs this process functions the same. After all elements are updated, the labeling status of the Agb, either for clauses or paragraphs depending on the type, is set to True.

5.2.4 Additional REST API Routes

This section provides an overview over the REST API routes implemented to provide the front-end with relevant data. First, an excerpt of the back-end implementation for the DataView is discussed, before we shortly show the data collection for the ClassDetailView.
5. Implementation Matching System

DataView Route

This section discusses the back-end implementation for the DataView. Listing 5.15 shows an excerpt of the data collection to create bar charts about Clauses.

```
1. @app.route("/data/<string:whatData>", methods=['GET'])
2. def get_data(whatData):
3.     agb = Agb.query.get(1)
4.     result = []
5.     other_classes = [5,10,12,21,24,29,37,40,45,48,51,55]
6.     if whatData == "clauses":
7.         for counter, clause in enumerate(agb.clauses):
8.             all_clauses = Clause.query.filter_by(trueState = counter).all()
9.             if counter in other_classes:
10.                new_class = {"x": counter, "y": len(all_clauses), "color": "red"}
11.            else:
12.                new_class = {"x": counter, "y": len(all_clauses)}
13.             result.append(new_class)
14.     elif whatData == "uniqueClauses":
15.         for counter, clause in enumerate(agb.clauses):
16.             clauses_in_class_x = Clause.query.filter_by(trueState = counter).all()
17.             clause_ids = list(map(lambda my_clause: my_clause.agb_id, clauses_in_class_x))
18.             unique = numpy.unique(clause_ids)
19.             if counter in other_classes:
20.                 new_class = {"x": counter, "y": len(unique), "color": "red"}
21.            else:
22.                 new_class = {"x": counter, "y": len(unique)}
23.             result.append(new_class)
```

Listing 5.15: Data for Bar Charts about Class Distribution of Clauses

Depending on what diagram we want to create, different data must be prepared. The bar chart for "clauses" displays, how many total Clauses each class has. We simply count them for each class and set the x and y coordinates to the class and number of Clauses.

If the current class is one of our collection classes “Sonstige […]”, we also set a special color, to make it easy for the user to distinguish between normal and collection classes.

For "uniqueClauses", instead of displaying the total count for each class, the user should see, how many Agbs have at least one Clause in a certain class. The agb_id of every Clause in a class is mapped to a list and all duplicates of agb_ids are filtered out. Then the class and number of unique agb_ids are mapped to x and y with collection classes specifically colored again.
ClassDetailView Route

The route for the ClassDetailView simply returns all Paragraphs or Clauses for a defined class. Listing 5.16 shows the implementation.

```python
1. @app.route("/dataFromClass/<string:poc>/<int:classID>", methods=['GET'])
2. def clausesFromClass(poc, classID):
3.   if poc == "clause":
4.     all_clauses = Clause.query.with_entities(Clause.id, Clause.rawText, Clause.trueState).filter_by(trueState = classID)
5.     return clauses_schema.jsonify(all_clauses)
6.   elif poc == "paragraph":
7.     all_paragraphs = Paragraph.query.with_entities(Paragraph.id, Paragraph.title, Paragraph.trueState).filter_by(trueState = classID)
8.     return paragraphs_schema.jsonify(all_paragraphs)
```

Listing 5.16: Excerpt of Back-End for ClassDetailView
6. Evaluation

In this chapter, we will evaluate our interactive matching system as well as the results of our classification of paragraphs and clauses to build our variability model.

This evaluation chapter is divided in two sections. First, a quantitative evaluation is performed, both on our interactive matching process as well as the different machine learning algorithms used in this process. The second part is concerned with the qualitative evaluation. We take a closer look at the resulting paragraph and clause distribution from our classification process and build a variability model as explained in chapter 3 for terms and conditions based on our findings.

6.1 Quantitative Evaluation

This section quantitatively evaluates our interactive matching system. We investigate how well contextual word embeddings support the semantic text matching task for paragraphs and clauses of terms and condition, compare them to document embeddings, and examine how well supervised machine learning approaches perform against our base case cosine similarity.

This evaluation is composed of three key sections: (1), we explain the actual workflow to classify paragraphs and clauses and evaluate our interactive matching process, (2), the overall quality based on the now existing labeled data is evaluated with state-of-the-art evaluation measures and. (3), compared across several vector representation type and machine learning models. But first, we want to introduce these measures and recap the different machine learning model, that are part of the implementation.

Evaluation Measures

Precision, as shown in Equation 6.1, displays the ratio of correctly classified paragraphs/clauses to all that were classified. It shows the ability of a system to not label negative data points as positive.

\[
\text{precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}
\]  

(6.1)
6. Evaluation

\[
\text{recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (6.2)
\]

Equation 6.2 shows recall, the ratio of correctly classified paragraphs/clauses to all, that should have been classified. It shows the systems’ ability to find all positive data points. The F1-Score (Equation 6.3) is a weighted average of precision and recall.

\[
F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (6.3)
\]

Machine Learning Models

Because no labeled data besides the ground truth was available before starting the matching process for terms and conditions, no assumption could be made about which models would perform best for our task. Therefore, common machine learning algorithms from different areas were used to predict classes and are now evaluated. The models were introduced in our implementation in Listing 5.9 and shall be recapped here:

- Logistic Regression (LR),
- Linear Discriminant Analysis (LDA),
- K-Nearest Neighbour (KNN),
- Decision Trees (DT), and
- Gaussian Naïve Bayes (NB).

Additionally, the cosine similarity (CS) between a new paragraph/clause and all paragraphs/clauses from the ground truth is calculated, with the highest similarity determining the potential class of new paragraphs/clauses.

6.1.1 Evaluating Classification Process

In this section, the actual process of classifying paragraphs and clauses is discussed in more detail. First, we explain our procedure and how it differs from the workflow intro-
duced in chapter 4. Then, the process is evaluated based on the introduced measures, separated for paragraphs and for clauses.

**Actual Workflow**

The actual workflow to classify terms and conditions differs slightly from the process introduced in Chapter 4. The original plan was to add and match terms and conditions one-at-a-time, thereby increasing our training data and improving the quality of our models in steps of size $n=1$. While segmenting a given document into paragraphs and clauses, applying preprocessing technique and learning word embeddings is computationally cheap, loading training data and fitting our models is not.

In practice, the complete dataset was collected and added to the database at once, before starting the matching process of paragraphs or clauses. Only when the data set was complete, did the preprocessing and learning embeddings take place. This way, instead of having a few minutes of processing time after adding each single terms and conditions (T&C), we could have the system do these processing steps at once without needing constant interaction of the user. The major disadvantage of first creating the complete data set was, that creating training sets for our models became more time consuming. If terms and conditions were added and matched one-at-a-time, whenever the models should predict classes for new data, we could simply load all vectors for either paragraphs or clauses, because all data currently in the system would be labeled. Instead, the system first needs to filter all terms and conditions that are already labeled, and then load all their paragraphs/clause vectors. While this took little time in the beginning, with a growing data set simply loading all labeled data to train the models takes several minutes.

Therefore, to avoid constantly having to reload the training set, and because our implemented models do not allow for re-fitting with new data, we decided to have our model predict for five sets of terms and conditions at once. While our interactive matching system learns in slower steps of size $n=5$, improvement in quality and therefore a faster matching process for the user makes it still fast enough, to outweigh constantly having to load our training sets.

As previously introduced, during the implementation we used the method of “most unique classes” as the basis of our matching process, where higher distribution of para-
graphs/clauses over the different classes is considered more desirable. We used this method for all, but the first four terms and conditions, where cosine similarity was used, to create a first small training set for future terms and conditions. Following, the evaluation of this method for both paragraphs and clauses is discussed.

**Paragraph Evaluation**

Figure 6.1 displays the evaluation measures *precision, recall and f1-score* for the classification process of paragraphs based on “most unique classes”.

![Scores for Paragraph Classification Process](image)

*Figure 6.1: Scores for Paragraph Classification Process*

*Source: Own Illustration*

As explained, classifying paragraphs was done in groups of five. The x-axis shows, how many terms and conditions were labeled to predict the current five paragraph sets: The scores at 35 for example show the average scores for paragraphs from T&C 36-40, predicted by the models trained on T&C 1-35.

While the scores are growing rapidly for the first 15 paragraph sets, they do not stabilize. They fluctuate between scores of ~60% with 40 labeled paragraphs and up to ~93% at 55. There are two potential explanations for this inconsistent behavior: either the new paragraphs are very different from previous ones, which leads to high inconsistency, or a model, that might not be the best, provided *most unique classes*. Table 6.1 displays all models including the cosine similarity and how often each model provided most unique classes for the paragraphs of one T&C.
6. Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>LR</th>
<th>LDA</th>
<th>KNN</th>
<th>DT</th>
<th>NB</th>
<th>CS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td># Chosen</td>
<td>31</td>
<td>38</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>3(+4)</td>
<td>98</td>
</tr>
</tbody>
</table>

Table 6.1: “Most Unique Classes” Distribution for Paragraphs

Source: Own Data

The linear discriminant analysis was most commonly used, closely followed by logistic regression. Our base case, the cosine similarity provided the most diverse predictions three times, with the other four being the first four T&Cs, to build a basic labeled set. Only Gaussian Naïve Bayes was never chosen.

If our initial assumption is, that higher diversity in predictions equals more desirable outcome, LR and LDA should provide the best results, because they were most commonly used. To confirm or reject the quality of our method, we compare all approaches. Figure 6.2 shows the precision of our approach (Unique) compared to all models and our base case, the cosine similarity.

![Figure 6.2: Precision of all Methods for Paragraphs](Source: Own Illustration)

As the graph shows, our approach of most unique classes performs average, with a few exceptions. Based on 20 paragraph sets, our approach performs better than any other model on its own, by combining predictions of several models, whereas at 95 it only uses the cosine similarity. In general, the graph for our approach is closely related to the LDA, which it used the most, but is pulled up 5-10% by LR and KNN in almost every step. Overall, LR and NB have the best and most consistent performance with a quick learning curve and up to ~98% precision, with KNN performing just as well, but a slower
6. Evaluation

learning curve. LR provides the best results in 15 of 19 steps, with the others being divided between KNN, NB and most unique classes.

The recommended machine learning model to semantically match terms and conditions for now is logistic regression.

Clause Evaluation

As for paragraphs, this section evaluates the matching process of clauses. In Figure 6.3, the results for the most unique classes approach are displayed.

![Scores for Clause Classification Process](source)

Compared to the paragraphs, the scores for clauses start about 20% worse, but have a similar climb of ~35% with only 15 labeled clause sets. After one deep dive that is 10-15% worse than even the starting predictions, most likely because T&C 21-25 contain very different or many new clauses, the scores slowly rise to ~90% and then fluctuating between ~65% and ~85%. Overall, the graphs develop similarly for clauses and paragraphs, which suggests, that whenever terms and conditions contain hard-to-classify (most likely because they are new) paragraphs, clauses are new or different, and therefore less precise to predict, as well.
6. Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>LR</th>
<th>LDA</th>
<th>KNN</th>
<th>DT</th>
<th>NB</th>
<th>CS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td># Winner</td>
<td>11</td>
<td>30</td>
<td>10</td>
<td>20</td>
<td>19</td>
<td>4(+4)</td>
<td>98</td>
</tr>
</tbody>
</table>

Table 6.2: “Most Unique Classes” Distribution for Clauses

Source: Own Data

For clauses, the *most unique classes* approach consumes a more even distribution of models for their predictions. While linear discriminant analysis still provides the most distributed predictions, decision trees and Gaussian Naïve Bayes have overtaken logistic regression (Table 6.2).

In Figure 6.4, we again compare our approach to purely using one of the underlying models with cosine similarity as the base case.

![Figure 6.4: Precision of all Methods for Clauses](Source: Own Illustration)

Several observations can be made for the clause matching process when comparing to the paragraph process. First, while the cosine similarity performs more consistent, it is also much worse and never even reaching the 40% mark. LR still outperforms all other models in 14 of the 19 steps, this time LDA performs second best leading all models the remaining five steps. The recommended model is still logistic regression.
6. Evaluation

6.1.2 Validation and Comparison to Paragraph Vectors

This section aims to validate the results from our matching process for clauses and paragraphs. While the quality of the models was already evaluated in the previous section, the results were dependent on the order we chose to classify our terms and conditions and might look completely different, when matched in a different order. This would allow the user to check all terms and conditions before matching and deciding on an order most beneficial to achieve high scores by fitting our models accordingly.

To validate, how the machine learning models perform for the semantic text matching task of terms and conditions independent of input order, k-fold cross-validation performed. This procedure randomly divides our data set into k groups of equal size, where each group is used once as a test set for the model trained on the remaining groups. The scores of all test sets are saved and averaged to obtain the overall evaluation measures (James et al. 2013, p.181). In this thesis, we perform cross-validation with the commonly recommended $k=10$ for paragraphs and clauses (Kuhn and Johnson 2013, p.70).

Afterwards, we compare the cross-validation results based on contextual word embeddings with paragraphs vectors created with the Doc2Vec approach, and a combined approach of concatenating both embedding types. It should be noted beforehand, that these comparisons are only valid to a limited extent, because of the different underlying language models. While the ELMos are learned based on a pre-trained German language model, we trained our own Doc2Vec models based on either all paragraphs or clauses from our dataset, with different vector sizes and architecture models. In the following section, only our best performing Doc2Vec models are used for comparison.

Paragraph Cross Validation

Figure 6.5 displays the results of the 10-fold cross-validation for paragraphs based on ELMos.
This validates the scores from the paragraph matching process. *Logistic Regression* is still the best performing model, closely followed by *Gaussian Naïve Bayes* and *K-Nearest Neighbour*.

The best performing Doc2Vec language model is based on the PV-DM architecture, with vector size of 256. Figure 6.6 compares contextual word embeddings, paragraph vectors and the concatenation of both only on the best performing logistic regression model.

Even though paragraph vectors were learned from a more specific language model, contextual embeddings still outperformed them by ~3%. But the best results are achieved by concatenating 1024-dimensional contextual embeddings with 256-dimensional paragraphs vectors.
6. Evaluation

Clause Cross Validation

Figure 6.7 displays the results of the 10-fold cross-validation for clauses based on ELMos. This again confirms the results from our clause matching process: LDA and LR outperform other models by at least 5%, with DT ranking last.

![Cross Validation Scores for Clauses](source: Own Illustration)

Unlike for paragraphs, the overall best performing Doc2Vec model is based on the PV-DBOW architecture with vector size 512. But like paragraphs, Doc2Vec alone scores worse than ELMos, while the concatenation slightly improves the scores. Figure 6.8 displays the comparison for the example of LR.

![Comparison between Embedding Types for Paragraphs](source: Own Illustration)
6.2 Qualitative Evaluation

After the quantitative evaluation gave insight into the performance of our terms and conditions matching system, we now want to take a closer look at the resulting classes for paragraphs and clauses. First, we look at the distribution of paragraphs and clauses, to discuss how suitable our chosen ground truth was for this problem. Afterwards, we explain, what constitutes as mandatory or optional paragraphs or clauses and create a feature model based on those.

6.2.1 Suitability of our Ground Truth

In this section, we discuss the resulting class distributions for paragraphs and clauses, to conclude, to what degree our ground truth covered real world terms and conditions.

Figure 6.9 displays the distribution of paragraphs over our classes, with the red bar representing “Sonstige”. 345 of a total of 1215 paragraphs didn’t fit into any of the twelve main classes, which is about 28.4%.

Figure 6.9: Paragraph Distribution
Source: Own Illustration

The most common ones, not covered by the ground truth, are “Anwendbares Recht” and “Gerichtsstand” with 42 and 47 appearances respectively, and “Vertragssprache” and “Vertragstextspeicherung” with 30 and 35 entries. While these paragraphs itself are not part of our ground truth, all their clauses, with the exception of clauses about “Vertragstextspeicherung”, are found within the ground truth. Clauses about “Vertragssprache” are represented in class three of the clauses (compare Figure 6.10), while clauses about
6. Evaluation

“Anwendbares Recht” and “Gerichtsstand” are covered in classes 53 and 54. Other paragraphs without a semantic match in our ground truth, that appear several times within our dataset are about “Datenschutz”, “Einlösung von Gutscheinen” and “Verhaltenkodizes”. None of their clauses have semantic links to any of our classes and are mostly part of clause class 55.

The distribution of clauses, as seen in Figure 6.10, shows, that many clauses from our data set don’t have semantic links to any of our clause classes. In total, 1452 of 3471 clauses, or around 41.8%, were sorted into one of the twelve miscellaneous classes. To check, whether our ground truth misses important or regularly appearing clauses, we shortly examine, what clause were sorted into the miscellaneous classes:

- **Class 5** contains clauses generally related to the paragraph “Geltungsbereich” and is mostly about the subject matter of the contract,
- **class 10** mostly provides clauses describing alternative ways for the conclusion of a contract between customer and the respective online shop, specific to that store,
- **class 12** has many clauses about additional cost, when the customer resides outside of Germany or the European union,
- **in class 21** most clauses are about the due date of the payment, with several clauses about certain payment methods like “SOFORT” or “RatePAY”,
- **class 24** only contains the miscellaneous clause from our ground truth,
- **class 29**, the paragraph specific class for “Lieferung und Eigentumsvorbehalten”, contains several frequently appearing clauses: rules about collection of the customer, delivery time, cost for the customer if they refuse the delivery
and the passing of accidental loss and the accidental deterioration to the cus-
tomer,

- **in class 37**, the most common clauses are about customers living outside the European unions and how they can’t make use of the right to withdraw, and who covers the return cost,

- **class 40** contains different additional clauses about “Transportschäden”,

- **class 45** has several clauses about the manufacturer’s warranty and subsequent improvement or replacement in case of defects,

- **class 48** mainly consists of two type of clauses: excluding further liability, and liability of the vendor and its vicarious agents or legal representatives,

- **class 51** only contains different clauses, and

- **in class 55** the biggest group of clauses is about vouchers with ~200 clauses originating from ~20 terms and conditions, with the second most common clause type being about saving the contract texts. Additionally, several clauses about following certain quality and data protection guidelines are part of this class. The remaining clauses only appear a handful of times.

Overall, while our ground truth covers most common paragraphs, many common clauses, especially for shipping are not covered. To get a better overview, over the most relevant paragraphs and clauses, the following section will design a feature model for general terms and conditions.

### 6.2.2 Feature Modeling for Terms and Conditions

As we explained in the beginning of this thesis, there are no strict rules about the format and content of terms and conditions. If a company doesn’t define a set of clauses, the underlying law is applied in cases in conflict. Therefore, Terms and conditions are an optional tool, to define rights and obligations for contracts, by “overwriting” existing unfavorable legal provisions (under certain restrictions).

For our feature model, this means there are no real “mandatory” clauses, since clauses are always optional: if we were to consider every type of clause from our data set, there would be not feasible way to display them in clear manner. Therefore, we define our own rules about what constitutes “mandatory” and “optional” paragraphs and clauses, based on our dataset:
6. Evaluation

- A class (both for paragraphs and clauses) is considered **mandatory**, when at least 50 online shops have at least one paragraph/clause in that class within their terms and conditions,
- A class is considered **optional**, when at least when as least 25 online shops fulfill the above rule.

Figure 6.11 displays, which paragraph classes fulfill the requirements for “mandatory”.

![Bar chart showing number of Terms and Conditions with Paragraphs in each Class](image)

**Figure 6.11: Number of Terms and Conditions with Paragraphs in each Class**

*Source: Own Illustration*

As the graph shows, only the paragraphs class 4 is neither optional or mandatory, eight classes should be mandatory and three optional. Looking deeper into the actual paragraphs for each class, the results are slightly different. Because we didn’t allow for multiclassing during the matching process, we sometimes had to choose between two possible classes for a paragraph. This problem appears for class 4, which is called “Aufrechnung; Zurückbehaltungsrecht” in our ground truth. Most paragraphs in the class are actually called “Zurückbehaltungsrecht, Eigentumsvorbehalt”, which would make them eligible for class 5 “Lieferung; Eigentumsvorbehalt” as well. The same problem is visible for class 2, where the ground truth is simply “Preise”, but many paragraphs are named “Preise und Zahlungsbedingungen”, which would qualify them for class 3. If both paragraph types appear alone more often than they do together, we consider them as separate classes, if they are more commonly combined, they will be considered as one class.
6. Evaluation

A different problem is visible in class 5 called “Lieferung; Eigentumsvorbehalt”, because most paragraphs in this class are either “Lieferung” or “Eigentumsvorbehalt. Therefore, this class should be split into two for our model.

After evaluating the different paragraph classes, we determine 9 paragraphs as mandatory. The optional paragraphs are based on the remaining classes 7 and 11, and paragraphs that appear at least 25 times in our miscellaneous class 12. This results in four different optional paragraph types. The same categorization into mandatory and optional is now done for clauses, based on Figure 6.12.

![Figure 6.12: Number of Terms and Conditions with Clauses in each Class](Source: Own Illustration)

Of all regular classes, 19 are represented in at least 50 terms and conditions and therefore mandatory. The only mandatory clause type, that isn’t represented by the ground truth, is about saving the text of contracts, for a total of 20 mandatory clauses. Nine regular classes appear in at least 25 terms and conditions, with another five clause types mainly about delivery, for a total of 14 optional clauses.

Before we finally introduce our feature model for terms and conditions, our definition of mandatory paragraphs needs to be extended slightly, because several of our mandatory clauses don’t fit into any of the current mandatory paragraphs. There are two possible options two resolve this: either one miscellaneous paragraph class taking in the remaining clauses, or upgrading the optional paragraphs, were the relevant clauses appear regularly, to mandatory. Because the second option provides a more distinct overview, we choose to upgrade the optional paragraphs.
6. Evaluation

For visibility, our feature model is divided into two parts: the “real” mandatory paragraphs with all their clauses are shown in Figure 6.14, while the “upgraded” optional paragraphs are shown in Figure 6.13.

![Feature Model of “Optional” Paragraphs and their Clauses](source: Own Illustration)

Here, we see the four “optional” paragraphs in light grey, with a total of five mandatory and one optional clause. For two of these clauses, there exists a certain formulation that is forbidden based on some court ruling (references to the rulings are summarized in Appendix A.2). While this graphic should be self-explanatory, we want to discuss some features of the next Figure in more detail. As Figure 6.14 shows, we have nine mandatory paragraphs containing 15 mandatory and 13 optional clauses. Four of our clauses have two or more options to choose from, while two clauses exclude each other, and two clauses require are previous clause to exist.

The two clauses excluding each other are one of the alternatives (XOR) of the customer definition and special rules regarding warranty for organizations. If an online shop determines, that only consumers are customer, there is no need to define special rules for organizations. Besides the above choice for the customer group, we also found two alternative for the conclusion of a contract: either the shop sends an order confirmation, or the conclusion depend on the chosen payment method. The payment methods also provide options (OR) for the online shop, where they can decide which payment methods
they offer to their customers. The final alternative is, whether an online shop agrees to partake in dispute settlement proceeding or not. We also have requirements between clauses, because the three relevant clauses appear together almost exclusively. Products displayed in a shop are non-binding offers, submitting a binding offer and conclusion of a contract always appear in a group, and therefore all require each other. Finally, we have several formulations for clauses, that are forbidden by law. “Voraussichtliche Lieferfrist [..]” for example is not a valid way to describe the delivery time and replacing a non-shippable product with one, that is qualitatively and base on cost similar, is not allowed either.

Concluding, with our feature model we provide an overview over paragraphs and clauses, defined as either mandatory or optional, appearing regularly in terms and conditions for online shops. We have shown relations between and variant of clauses, as well as exemplary clauses, that are against the law. Our model provides a first checklist for important clauses and decisions a new shop owner has to decide on but is by no means a complete list and should therefore only be used as a starting point give a short summary over important points.
Figure 6.14: Feature Model of Mandatory Paragraphs and their Clauses
Source: Own Illustration
7. Conclusion

In this thesis, an interactive process to successfully link semantically related paragraphs and clauses from a corpus of general terms and conditions was presented.

First, we briefly introduced the motivation for this thesis. Then, our research approach and relevant research questions for this thesis were described, before quickly discussing the selection process for our ground truth and the collection of general terms and conditions required for this thesis.

Afterwards, important topics were introduced, after performing a comprehensive literature review. First, concepts related to natural language processing, like the semantic text matching task, different commonly used preprocessing techniques, and several methods to create word and documents embeddings were discussed. Then, the concept of interactive machine learning introduced, before the topic of feature modeling was discussed.

After laying the knowledge foundation, the design process for our matching system started. First, functional and non-functional requirement were introduced. Based on the requirements, we designed relevant use cases and discussed the potential workflow for adding and matching terms and conditions in our system. We introduced our 3-tiered software architecture, that is commonly used in the development of web applications and defined the underlying data model for our matching system.

During the implementation chapter, we discussed the user interface with all its view and components and the back-end implementation, that functioned as a REST API to facilitate communication between our front-end end back-end.

In the quantitative evaluation, we first discussed our actual workflow for classifying paragraphs and clauses from terms and conditions and discussed the results of our process. The power of contextual word embeddings in combination with several supervised machine learning algorithms was shown and improvements by combing them with context-independent embeddings were discussed.

In the final part of this thesis, we evaluated the resulting classes from our matching process, defined “mandatory” and “optional” paragraphs and clauses, and creature a feature modeling containing the most common items within our data set.
In future work, to further improve the quality of our matching system, extensive grid searching methods can be applied to further improve the predictions quality of our machine learning models. A more extensive search into unlawful clauses can be conducted to build a more rounded variability model. Additionally, more feedback methods can be implemented, to improve the understanding and knowledge of the user for our system.

Finally, this system could be extended to support the matching of other forms of legal contracts, where labeled data is not readily available.
Bibliography


Kannan, Dr S, and Vairaprapaksh Gurusamy. 2014. “Preprocessing Techniques for Text Mining.” *6*.

Appendix A

A.1 Ground Truth

1 Geltungsbereich und Anbieter

(1) Diese Allgemeinen Geschäftsbedingungen gelten für alle Bestellungen, die Sie bei dem Online-Shop der Anschrift tätigen.

(2) Das Warenangebot in unserem Online-Shop richtet sich ausschließlich an Käufer, die das 18. Lebensjahr vollendet haben.


(4) Vertragssprache ist ausschließlich deutsch.

(5) Sie können die derzeit gültigen Allgemeinen Geschäftsbedingungen auf der Website [Link angeben] abrufen und ausdrucken.

(6) Sonstige Klauseln Geltungsbereich und Anbieter

2 Vertragsschluss


(2) Mit Anklicken des Buttons „Jetzt zahlungspflichtig bestellen“/„kaufen“ geben Sie ein verbindliches Kaufangebot ab (§ 145 BGB). Unmittelbar vor Abgabe dieser Bestellung können Sie die Bestellung noch einmal überprüfen und ggf. korrigieren.

(4) Ein Kaufvertrag über die Ware kommt erst zustande, wenn wir ausdrücklich die Annahme des Kaufangebots erklären (Auftragsbestätigung) oder wenn wir die Ware - ohne vorherige ausdrückliche Annahmeerklärung - an Sie versenden. Ausnahme: bei Zahlung mit Vorkasse und PayPal erfolgt die Annahme der Bestellung unmittelbar mit Ihrer Bestellung.

(5) Sonstige Klauseln Vertragsschluss

3 Preise

Die auf den Produktseiten genannten Preise enthalten die gesetzliche Mehrwertsteuer und sonstige Preisbestandteile und verstehen sich zzgl. der jeweiligen Versandkosten. Weitere Informationen zu den Versandkosten erhalten Sie auf unserer Internetseite unter [„Versandinformationen“ / „Lieferbedingungen“].

Sonstige Klauseln Preise

4 Zahlungsbedingungen; Verzug

(1) Die Zahlung erfolgt wahlweise per: Rechnung per Vorkasse, Nachnahme, Kreditkarte, Paypal oder Lastschrift.

(2) Die Auswahl der jeweils verfügbaren Bezahldenken obliegt uns. Wir behalten uns insbesondere vor, Ihnen für die Bezahlung nur ausgewählte Bezahldenken anzubieten, beispielweise zur Absicherung unseres Kreditrisikos nur Vorkasse.

(3) Bei Auswahl der Zahlungsart Vorkasse nennen wir Ihnen unsere Bankverbindung in der Auftragsbestätigung. Der Rechnungsbetrag ist innerhalb von 10 Tagen nach Erhalt der Auftragsbestätigung auf unser Konto zu überweisen.
Appendix A


(5) Bei Zahlung per Kreditkartewirdder Kaufpreis zum Zeitpunkt der Bestellung auf Ihrer Kreditkarte reserviert (Autorisierung). Die tatsächliche Belastung Ihres Kreditkartenkontos erfolgt in dem Zeitpunkt, in dem wir die Ware an Sie versenden.


(7) Bei Zahlung per Lastschrift haben Sie ggf. jene Kosten zu tragen, die infolge einer Rückbuchung einer Zahlungstransaktion mangels Kontodeckung oder aufgrund von Ihnen falsch übermittelter Daten der Bankverbindung entstehen.


(9) Sonstige Klauseln Zahlungsbedingungen und Verzug

5 Aufrechnung; Zurückbehaltungsrecht

(1) Ein Recht zur Aufrechnung steht Ihnen nur dann zu, wenn Ihre Gegenforderung rechtskräftig festgestellt worden ist, von uns nicht bestritten oder anerkannt wird oder in einem engen synallagmatischen Verhältnis zu unserer Forderung steht.

(2) Sie können ein Zurückbehaltungsrecht nur ausüben, soweit Ihre Gegenforderung auf demselben Vertragsverhältnis beruht.
6 Lieferung; Eigentumsvorbehalt

(1) Sofern nicht anders vereinbart, erfolgt die Lieferung der Ware von unserem Lager an die von Ihnen angegebene Adresse.

(2) Die Ware bleibt bis zur vollständigen Zahlung des Kaufpreises unser Eigentum.

(3) Wir sind ausnahmsweise nicht zur Lieferung der bestellten Ware verpflichtet, wenn wir die Ware unsererseits ordnungsgemäß bestellt haben, jedoch nicht richtig oder rechtzeitig beliefert wurden (kongruentes Deckungsgeschäft). Voraussetzung ist, dass wir die fehlende Warenverfügbarkeit nicht zu vertreten haben und Sie über diesen Umstand unverzüglich informiert haben. Zudem dürfen wir nicht das Risiko der Beschaffung der bestellten Ware übernommen haben. Bei entsprechender Nichtverfügbarkeit der Ware werden wir Ihnen bereits geleistete Zahlungen unverzüglich erstatten. Das Risiko, eine bestellte Ware besorgen zu müssen (Beschaffungsrisiko), übernehmen wir nicht. Dies gilt auch bei der Bestellung von Waren, die nur ihrer Art und ihren Merkmalen nach beschrieben ist (Gattungswaren). Wir sind nur zur Lieferung aus unserem Warenvorrat und der von uns bei unseren Lieferanten bestellten Waren verpflichtet.

(4) Wenn Sie Unternehmer im Sinne des § 14 BGB sind, gilt ergänzend Folgendes: Wir behalten uns das Eigentum an der Ware bis zum vollständigen Ausgleich aller Forderungen aus der laufenden Geschäftsbeziehung vor. Vor Übergang des Eigentums an der Vorbehaltsware ist eine Verpfändung oder Sicherheitsübereignung nicht zulässig. Sie dürfen die Ware im ordentlichen Geschäftsgang weiterverkaufen. Für diesen Fall treten Sie bereits jetzt alle Forderungen in Höhe des Rechnungsbetrages, die Ihnen aus dem Weiterverkauf erwachsen, an uns ab. Wir nehmen die Abtretung an, Sie sind jedoch zur Einziehung der Forderungen ermächtigt. Soweit Sie Ihren Zahlungsverpflichtungen nicht ordnungsgemäß nachkommen, behalten wir uns das Recht vor, Forderungen selbst einzuziehen. Bei Verbindung und Vermischung der Vorbehaltsware erwerben wir Miteigentum an der neuen Sache im Verhältnis des Rechnungswertes der Vorbehaltsware zu den anderen verarbeiteten Gegenständen zum Zeitpunkt der Verarbeitung. Wir verpflichteten uns, die uns zustehenden Sicherheiten auf Verlangen insoweit freizugeben, als der realisierbare Wert unserer Sicher-
heiten die zu sichernden Forderungen um mehr als 10 % übersteigt. Die Auswahl der freizu-
gebenden Sicherheiten obliegt uns.

(5) Sonstige Klauseln Lieferung und Eigentumsvorbehalt

7 Widerrufsbelehrung

Für den Fall, dass Sie Verbraucher im Sinne des § 13 BGB sind, also den Kauf zu Zwecken
tätigen, die überwiegend weder Ihrer gewerblichen noch Ihrer selbständigen beruflichen Tä-
tigkeit zugerechnet werden können, haben Sie ein Widerrufsrecht nach Maßgabe der folgen-
den Bestimmungen.

Widerrufsbelehrung

Musterformular

(1) Das Widerrufsrecht besteht nicht bei der Lieferung von Waren, die nicht vorgefertigt sind
und für deren Herstellung eine individuelle Auswahl oder Bestimmung durch den Verbrau-
cher maßgeblich ist oder die eindeutig auf die persönlichen Bedürfnisse des Verbrauchers
zugeschnitten sind(z.B. T-Shirts mit Ihrem Foto und Ihrem Namen), versiegelter Waren, die
aus Gründen des Gesundheitsschutzes oder der Hygiene nicht zur Rückgabe geeignet sind,
ennen ihre Versiegelung nach der Lieferung entfernt wurde, von Waren, wenn diese nach der
Lieferung auf Grund ihrer Beschaffenheit untrennbar mit anderen Gütern vermischt wurden,
von Ton- oder Videoaufnahmen oder Computersoftware in einer versiegelten Packung, wenn
die Versiegelung nach der Lieferung entfernt wurde, von Zeitungen, Zeitschriften oder Illus-
strierten mit Ausnahme von Abonnement-Verträgen.

(2) Bitte vermeiden Sie Beschädigungen und Verunreinigungen. Senden Sie die Ware bitte
möglichst in Originalverpackung mit sämtlichem Zubehör und mit allen Verpackungsbestand-
teilen an uns zurück. Verwenden Sie ggf. eine schützende Umverpackung. Wenn Sie die
Originalverpackung nicht mehr besitzen, sorgen Sie bitte mit einer geeigneten Verpackung
für einen ausreichenden Schutz vor Transportschäden, um Schadensersatzansprüche we-
gen Beschädigungen infolge mangelhafter Verpackung zu vermeiden.
(3) Bitte rufen Sie vor Rücksendung unter der Telefonnummer bei uns an, um die Rücksendung anzukündigen. Auf diese Weise ermöglichen Sie uns eine schnellstmögliche Zuordnung der Produkte.

(4) Bitte beachten Sie, dass die in den vorstehenden Absätzen 2 und 3 genannten Modalitäten nicht Voraussetzung für die wirksame Ausübung des Widerrufsrechts sind.

(5) Sonstige Klauseln Widerrufsbelehrung

8 Transportschäden

(1) Werden Waren mit offensichtlichen Transportschäden angeliefert, so reklamieren Sie solche Fehler bitte sofort bei dem Zusteller und nehmen Sie bitte schnellstmöglich Kontakt zu uns auf.

(2) Die Versäumung einer Reklamation oder Kontaktaufnahme hat für Ihre gesetzlichen Gewährleistungsrechte keine Konsequenzen. Sie helfen uns aber, unsere eigenen Ansprüche gegenüber dem Frachtführer bzw. der Transportversicherung geltend machen zu können.

(3) Sonstige Klauseln Transportschäden

9 Gewährleistung

(1) Soweit nicht ausdrücklich etwas anderes vereinbart ist, richten sich Ihre Gewährleistungsansprüche nach den gesetzlichen Bestimmungen des Kaufrechts (§§ 433ff. BGB).

(2) Wenn Sie Verbraucher im Sinne des § 13 BGB sind, beträgt die Haftungsdauer für Gewährleistungsansprüche bei gebrauchten Sachen - abweichend von den gesetzlichen Best-

(3) Im Übrigen gelten für die Gewährleistung die gesetzlichen Bestimmungen, insbesondere die zweijährige Verjährungsfrist gem. § 438 Abs. 1 Nr. 3 BGB.

(4) Wenn Sie Unternehmer im Sinne des § 14 BGB sind, gelten die gesetzlichen Bestimmungen mit folgenden Modifikationen: Für die Beschaffenheit der Ware sind nur unsere eigenen Angaben und die Produktbeschreibung des Herstellers verbindlich, nicht jedoch öffentliche Anpreisungen und Äußerungen und sonstige Werbung des Herstellers. Sie sind verpflichtet, die Ware unverzüglich und mit der gebotenen Sorgfalt auf Qualitäts- und Mengenabweichungen zu untersuchen und uns offensichtliche Mängel binnen 7 Tagen ab Empfang der Ware anzuzeigen. Zur Fristwahrung reicht die rechtzeitige Absendung. Dies gilt auch für später festgestellte verdeckte Mängel ab Entdeckung. Bei Verletzung der Untersuchungs- und Rügepflicht ist die Geltendmachung der Gewährleistungsansprüche ausgeschlossen. Bei Mängeln leisten wir nach unserer Wahl Gewähr durch Nachbesserung oder Ersatzlieferung (Nacherfüllung). Im Falle der Nachbesserung müssen wir nicht die erhöhten Kosten tragen, die durch die Verbringung der Ware an einen anderen Ort als den Erfüllungs- ort entstehen, sofern die Verbringung nicht dem bestimmungsgemäßen Gebrauch der Ware entspricht.-Schlägt die Nacherfüllung zweimal fehl, können Sie nach Ihrer Wahl Minderung verlangen oder vom Vertrag zurücktreten. Die Gewährleistungsfrist beträgt ein Jahr ab Ablieferung der Ware.

(5) Sonstige Klauseln Gewährleistung
10 Haftung


(2) Im Übrigen gilt folgende beschränkte Haftung: Bei leichter Fahrlässigkeit haften wir nur im Falle der Verletzung einer wesentlichen Vertragspflicht, Erfüllung die ordnungsgemäße Durchführung des Vertrags überhaupt erst ermöglicht und auf deren Einhaltung Sie regelmäßig vertrauen dürfen (Kardinalpflicht). Die Haftung für leichte Fahrlässigkeit ist der Höhe nach beschränkt auf die bei Vertragsschluss vorhersehbaren Schäden, mit deren Entstehung typischerweise gerechnet werden muss. Diese Haftungs-beschränkung gilt auch zugunsten unserer Erfüllungsgehilfen.

(3) Sonstige Klauseln Haftung

11 Alternative Streitbeilegung


Wir sind bemüht, eventuelle Meinungsverschiedenheiten aus unserem Vertrag einvernehmlich beizulegen. Darüber hinaus sind wir zu einer Teilnahme an einem Schlichtungsverfahren nicht verpflichtet und können Ihnen die Teilnahme an einem solchen Verfahren leider auch nicht anbieten.

Sonstige Klauseln Alternative Streitbeilegung

12 Schlussbestimmungen
(1) Sollten eine oder mehrere Bestimmungen dieser AGB unwirksam sein oder werden, wird dadurch die Wirksamkeit der anderen Bestimmungen im Übrigen nicht berührt.


(3) Sind Sie Kaufmann, juristische Person des öffentlichen Rechts oder öffentlich-rechtliches Sondervermögen, so ist unser Geschäftssitz Gerichtsstand für alle Streitigkeiten aus oder im Zusammenhang mit Verträgen zwischen uns und Ihnen

13 Sonstige

Sonstige Klauseln
<table>
<thead>
<tr>
<th>Reference for illegal Clauses</th>
<th>Court and Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Die AGB gelten auch für alle künftigen Geschäftsbeziehungen, auch wenn sie nicht erneut ausdrücklich vereinbart werden”</td>
<td>LG München I (Urteil v. 14.08.2003, 12 O 2393/03)</td>
</tr>
<tr>
<td>“Versand auf Risiko des Käufers”</td>
<td>LG Landau (Urteil v. 17.02.2006, HK O 977/05)</td>
</tr>
<tr>
<td>“Fehllieferungen oder offensichtliche Mängel sind durch den Kunden innerhalb von 2 Wochen nach Anlieferung der Ware zu rügen.”</td>
<td>OLG Koblenz (Urteil v. 03.12.2008, 4 W 681/08)</td>
</tr>
<tr>
<td>“Die Parteien verpflichten sich für den Fall der Unwirksamkeit einer Bestimmung, sie durch eine andere zu ersetzen, die dem wirtschaftlichen Zweck der unwirksamen Bestimmung am nächsten kommt.”</td>
<td>LG Hamburg (U. v. 14.09.2006, 327 O 441/06)</td>
</tr>
<tr>
<td>„Voraussichtliche Versanddauer: 1 -3 Werktage„</td>
<td>OLG Bremen (Urteil v. 05.10.2012, Az. 2 U 49/12)</td>
</tr>
</tbody>
</table>