Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät für Informatik und Wirtschaftsinformatik

Master-Thesis

Name:	Fabian Hoppe
Thema:	Comparative study on algorithms used for clustering of natural language text
Arbeitsplatz:	Robert Bosch Engineering and Business Solutions Limited , Bengaluru
Referent:	Prof. Dr.-Ing. Laubenheimer
Korreferent:	Prof. Dr. Link
Abgabetermin:	02.10.2018

Karlsruhe, 19.03.2018

Der Vorsitzende des
Prüfungsausschusses

Prof. Dr. Heiko Körner

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 18.10.2018

. . . . . .

(Fabian Hoppe)

Abstract

We are everyday surrounded by vast amounts of information. Consequently Information Retrieval (IR) systems like search engines have become a part of our everyday life. But most of that information is unstructured data like natural language. This creates a challenge for IR systems, because in order to retrieve relevant documents the IR system has to understand the texts.

This work identifies features which capture the meaning of natural language texts (NLU feature) and compares them regarding the detection of duplicates. The compared NLU feature are lexical approaches, Par2Vec and a transfer-learning task. Lexical approaches and Par2Vec are typical unsupervised features. The transfer-learning approach trains the extraction of features on the STS task.

The duplicate detection represents many IR tasks. The classification uses cosine similarity to compare the data collection and a simple threshold to classify them. All features are evaluated with two datasets. No NLU feature is able to outperform the baseline TF-IDF feature. Especially the transfer-learning feature fails due to the limited model complexity and training data to learn linguistic rules.

Zusammenfassung

Wir sind jeden Tag von einer Menge an Informationen umgeben, daher gehören Information Retrieval Systeme wie eine Suchmaschinen zum alltäglichen Leben. Die Mehrheit dieser Informationen sind unstrukturierte Daten wie Sprache. Diese stellen eine Herausforderung für IR Systeme dar. Die Suche nach relevanten Dokumenten muss diese verstehen.

Die vorliegende Arbeit untersucht Merkmale, welche die Bedeutung eines Textes repräsentieren. Es werden lexikalische Methoden, Par2Vec und ein transfer-learning Ansatz beschrieben. Der transfer-learning Ansatz lernt das Extrahieren relevanter Merkmale durch STS-Daten.

Die Merkmale sollen als exemplarische IR Aufgabe Duplikate erkennen können. Dazu wird eine Klassifikation von zwei Datensetzen über die Kosinus Ähnlichkeit und einem Schwellwert vorgenommen. Keines der vorgeschlagenen Merkmale erlaubt eine bessere Klassifikation als das TF-IDF Referenzmerkmal.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artifical Intelligence

ANN Artifical Neural Network

BoW Bag of Words

BiLSTM Bidirectional Long Short Term Memory

CBoW Continuous Bag of Words

CNN Convolutional Neural Network

CSV Comma-separated Values

DNN Deep Neural Network

GloVe Global Vector

GRU Gated Recurrent Unit

IDF Inverse Document Frequency

IR Information Retrieval

LSTM Long Short Term Memory

LDA Latent Dirichlet Allocation

LSA Latent Semantic Analysis

ML Machine Learning

NLP Natural Language Processing

NLU Natural Language Understanding

NL Natural Language

NER Named Entity Recognition

NLTK Natural Language Toolkit

NLI Natural Language Inference

OCR Optical Character Recognition

PCA Principal Component Analysis

POS Part Of Speech

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

SVD Singular Value Decomposition

STS Semantic Textual Similarity

SVM Support Vector Machine

TF Term Frequency

1. Introduction

Humans are able to exchange all kind of information through language. For instance, language is used to talk about activities, to discuss abstract concepts and even helps to determine the sentimental state of other human beings. Consequently, language is used frequently in all kind of communication, like e-mails, reports, conversations and scientific papers. This usage generates large language datasets, which contain a lot of information. Approximated 80% of all relevant business data is text-heavy unstructured data[10]. Unfortunately acquiring insights out of text is a major problem for big data approaches. Currently the main part of those information cannot be used for further automatically analysis. The lack of such methods makes it difficult to find relevant information in large text datasets. It is like searching a needle in a haystack and the observable exponential increase of data in the last years deteriorates this problem by adding even more hay.

Without being able to find relevant information it is likely that already existing information is reproduced. The reproduction of information causes regularly unnecessary work. One example for this is the reimplementation of an algorithm. In software development it is well-known, that this duplication creates additional problems for maintaining the implementations and even causes inconsistencies based on slight differences of the several implementations. Of course, the produced heterogeneous data is a problem in other areas as well and makes the search itself more difficult. Other consequences of information shortage are even more devastating. Managers could make wrong business decisions, lawyers would not be able to defend their clients, engineers could design buggy products and the research of scientists would be more difficult without accessing the results of colleagues. Information has become by far the most valuable resource for almost everyone. Therefore the required access through Information Retrieval systems to manage large amounts of data is one of today's key challenges.

Information Retrieval (IR) recognizes the implicit patterns contained in collections of unstructured data to find data that satisfies information needs[48]. This defines a variety of IR algorithms for a broad application field. Depending on the task particular datasets, patterns and formulations of information needs are considered. Popular examples for IR systems are search engines like *Google* or *Baidu* which are used on a daily basis to search the web with a query-based request by billions of people. Recently the virtual assistants *Siri*, *Alexa* and *Cortana* integrated simple question answering systems, which allow a more natural language oriented formulation of information needs. The *IBM Watson* project illustrates that Information Retrieval is also used in enterprise related contexts as well as scientific fields.

Consistently those examples and IR in general focus on the biggest part of unstructured data - natural language. This focus establishes IR as a subfield of Natural Language Processing (NLP). As already mentioned the mandatory analysis of language to find the relevant information is a tough challenge. Language is ambiguous in many aspects,

uses a compositional structure and allows the definition of new concepts within the language. Nevertheless a perfect IR system must understand the text on a human or even superhuman level to find information needs. The task of understanding natural language is a fundamental building block for IR models and many other NLP tasks. The corresponding subfield is called Natural Language Understanding (NLU) and is AI-complete, which means that solving the NLU problem makes all human knowledge available to machines and constitutes artificial general intelligence[49].

1.1. Goal

The principal goal of Information Retrieval (IR) is making information accessible through document search based on an information requirement. It organizes large amounts of data and structures the data according to implicit patterns contained in the data and the information requirement. IR tasks consists of two distinct sub goals: understanding the input data and analyzing it with respect to this understanding by classification or clustering methods.

The abstract task of *understanding language* requires a further clarification. Therefore, it is necessary to take a closer look at the concept of language. Ludwig Wittgenstein describes language as a tool to trigger *pictures* in our minds[56]. According to Wittgenstein those *pictures* represent the meaning of a text and depend on the knowledge of the initiator. In order to find the same *picture* based on the text and thereby understanding it the knowledge of the recipients has to align with the knowledge of the initiator. NLU requires specific knowledge. Assuming a constructivist learning model those *pictures* extend the knowledge of all recipients and change future *pictures*. This learning process establishes a bidirectional dependency between language and knowledge. Therefore NLU has to integrate the described *pictures* into its knowledge base, otherwise it would not be possible to receive the right *picture* in an ongoing text. Understanding a text according to Wittgenstein means creating a *picture* based on this text and a priori knowledge. Transferring this concept into the domain of machine learning the creation of a *picture* is the generation of a representation within a vector space model based on several features of the text. Current systems rely on domain specific feature and thereby understand certain aspects. The research moves towards general understanding of texts to create a task independent representation of the meaning similar to the *pictures* humans create in their mind.

After the NLU algorithms created a representation for all texts it is possible to analyze them regarding specific patterns. This analysis step provides a subset of documents which comply to information needs: the retrieved documents. Therefore, the analysis focuses on grouping and potentially ranking documents according to this need. Information needs can either be formulated directly by transforming it into a NLU feature and compare it to the document representations or implicitly by classifying all documents. This text classification determines information needs based on different classes and requires labeled data for each class. In contrast to grouping all documents according to labels the direct comparison of NLU features does not depend on labeled data. Instead the most similar documents compared to information needs are retrieved by clustering them. Clustering

gathers similar documents and information needs with a data-driven approach. It does not need a priori knowledge about the partitioning of the document collection. However, the main goal of retrieving documents which satisfy a specific information need is shared by both supervised and unsupervised models. The retrieval task determines whether classification or clustering is applied.

Figure 1.1 depicts the fundamental IR process with the described NLU and analysis steps. The state-of-the-art models for both steps are illustrated in the following section.

Figure 1.1.: Schematic overview of transformation of text into the feature space with NLU and the analysis by classification or clustering of the feature. Notable the supervised setting (dashed line) specifies the information needs by labeling a part of the collection. The structure of such labeled data is well-known and do not require NLU.

1.2. State of the art

The field of Information Retrieval is a cornerstone of the current information age and has a significant value for corporations and individuals. This value becomes apparent by looking at the success of web search engines like *Google* and *Baidu* which represent just one application of the whole IR field. The diverse field of IR systems has led to many publications and conferences around this topic. The depicted state of the art in the following sections focuses on the creation of a vector space model to represent the meaning of texts and the related classification and clustering analysis tasks. Other IR topics like document indexing, formulation of information needs through queries, Boolean retrieval strategies and scoring models are not explicitly mentioned and will not be part of the research provide by this thesis as well.

1.2.1. Natural Language Understanding

As described the first step towards finding relevant information is understanding the document collection and the information needs. The field of NLU addresses this problem for general NLP tasks by extracting features which represent the meaning. Those features can be either extracted by rule-based systems or with data driven statistical approaches. Similar to other fields in AI recent algorithms of NLU focus on statistical deep learning models, which provide numerical features instead of simple nominal features provided by most rule-based approaches.

1.2.1.1. Rule-based approaches

First NLU approaches were published between 1960 and 1975. Examples of those early systems are solving algebra text problems from school books[4] or answering question about the position of building blocks in a model world[55] to prove the understanding of a text input. Those early systems use rule-based methods to understand language commands and perform actions according to the understanding. For example the block model of the SHRDLU system[55] moves blocks and answers questions based on English key words. Figure 1.2 shows an example input query and the corresponding action of the system. The rules used in NLP systems are described with context-free grammars and checked

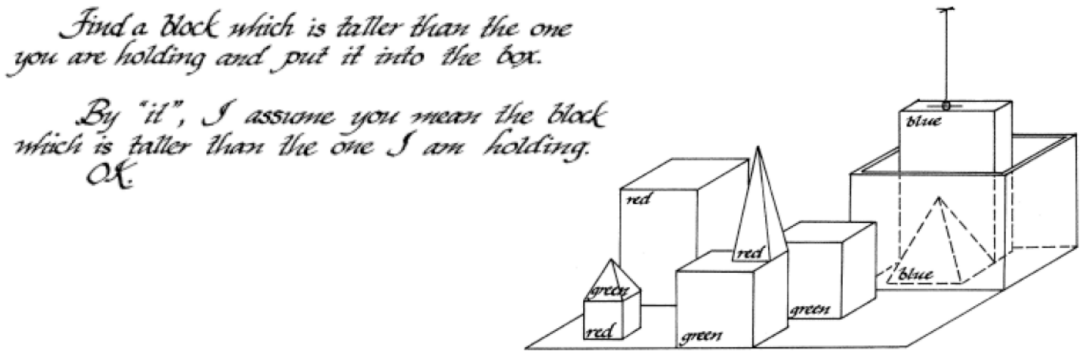


Figure 1.2.: Visualization of the SHRDLU model world together with a query input and the corresponding output[55].

by parsers. For example, SHRDLU uses a systemic functional grammar to process the language commands[55]. The idea of rule-based systems is supported by the linguistic concept of generativism. This concept was introduced by Noam Chomsky and claims that all NL is governed by general laws and principles, therefore NL can be described with a generative grammar[12].

Rule-based systems focus on a specific domain, like a restricted world of colored blocks. In this domain the possible inputs and outputs can be processed with few rules or a simple grammar and the system is able to fully understand the reduced language input. Improvements in the following years showed that those systems are very difficult to scale to a wider domain or adopt to another domain[34]. Rule-based systems, like SHRDLU require knowledge about the different utterances and how they can be connected. Therefore such systems grow exponentially with an increasing number of utterances. Nevertheless rule-based approaches are still considered in NLP applications with specific domains, especially data preparation relies on the simplistic approaches provided by rule-based systems. Examples are tokenization[24], stemming[44] and sentence boundary detection[24].

Nominal feature Early rule-based systems use words as nominal features. The rules test whether the given words or word phrases are elements of a set of words. The introduced historical rule-based systems [4, 55] use small word lists and check if the input is a part of them.

Instead of using distinct word sets with no additional information more recent approaches match the input words to big thesauri to find relations like synonyms or hyponyms (*is a* relation) between words. The word relations in a thesaurus like WordNet[28] are used to define a graph. This graph uses the graph distance between words to measure the semantic similarity. Budanitsky and Hirst provide an overview on five such distance measurements[7].

Other relations like stereotypical situations for the words also allow to understand the context of those words. For example the FrameNet[18] dataset lists *duration, place, student, teacher, subject, motivation,* etc. as stereotypical situations for the word *studying*. The situations could be used to add semantic annotations to the sentence[20]. By structuring those annotations, it is even possible to generate complex parse trees based on those frames[1]. Recently Liang proposed to train the semantic parsing process by question answer pairs to reduce the required amount of supervision to create a formal representation of the questions[33]. This approach uses statistical methods to generate a grammar and combines thereby nominal feature with statistical approaches.

1.2.1.2. Statistical approaches

The increase in computational power and available language data was responsible for the shift from rule-based methods to data-driven statistical methods in the early 1990s. The statistical revolution also marks the shift from generative grammars to more structural linguistics-based approaches. Structural linguistics focus on probabilities assembled over a large utterance corpus[36]. Instead of trying to find complex rules which describe the language, statistical methods analyze the occurrence of words and phrases and try to find patterns to understand the text. An example for this analysis task are language models. Those models try to predict a word with the help of given context words. Those so called word embeddings have been proven to be useful for many NLP task. The language models are used as inputs for supervised NLU tasks, like question answering[54], sentimental analysis[51] or Semantic Textual Similarity (STS)[8]. Those systems use CNNs or RNNs with large training datasets and hand-crafted additional features like POS tags. Other NLP tasks like machine translation, text summarization use similar encoder-decoder architectures to convert the text. Those tasks also require semantical understanding of the input text, therefore the encoder is also trained to produce an abstract representation of the meaning[9].

Statistic methods are able to process the complete corpus of a language. They are not restricted to specific domains with reduced vocabulary[34]. But statistical methods focus on specific subtasks of NLP, like language models instead of understanding the whole language. This yields the classification of statistical and rule-based systems shown in Figure 1.3. Statistic methods achieved compelling result over all application domains for subtask, but do not reach the deep language understanding of rule-based systems.

Recent research focuses on adding more structural components like dependency grammars to ANNs methods. Tai et al. use Long Short Term Memorys (LSTMs) to model the tree structure of language[53], Liang trains networks for semantic parsing[33]. Those approaches combine statistical methods and generative grammars. They are aiming to solve the *poverty of stimuli* problem[13]. Chomsky argues that a child is able to learn lan-

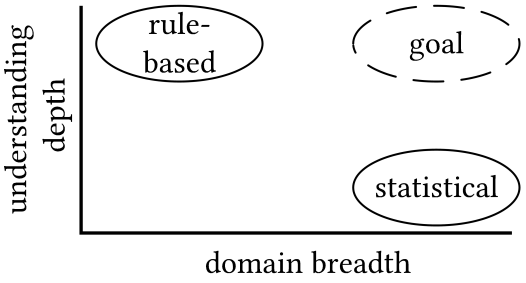


Figure 1.3.: Classification of rule-based and statistical approaches regarding the domain breadth and understanding depth dimension[34]. The optimal approach would archive a high value in both dimensions.

guage with a small amount of training examples, especially almost non-negative examples. According to Chomsky to solve this problem the human mind needs to use a language acquisition device, that contains a generative grammar[11]. Manning and Schütze state that at least some architectural particularity is required to understand a language on a human-level and that it is not possible to learn a language with a tabula rasa[36].

Numerical feature Statistical approaches generate numerical feature to represent words, sentences and paragraphs by counting and analyzing the occurrences over large corpora. The Bag of Words (BoW) model is the first step towards a numerical representation. It accumulates *one hot* encoded word vectors over a fix vocabulary of words[48]. Despite its simplicity it has been proven to work well for tasks like text categorization[38, 27]. The problem of different information entropy for words in languages are addressed by TF-IDF approaches[48]. But the main disadvantage of the BoW model is the assumption of same distances between different words. It does not involve any semantics.

In compliance with the famous quote "You shall know a word by the company it keeps."[19] from the linguist Firth more sophisticated features take the local context into account to address this problem. The local context of a word is its n surrounding words. Based on this concept LSA[16],LDA[3] and more recently word2vec[40] and GloVe[43] were proposed. Those word representations or word embeddings have been proven to accurately map the semantic of words into the vector space. Figure 1.4 shows that semantically close words are also mapped to nearby vectors. Especially the analogy task shows impressive results. For example produces the vector calculation $\text{woman} - \text{man} + \text{king}$ a result close to queen[40]. In contrast to the introduced thesauri-based feature the captured semantic entanglement of close features does not reflect a distinct relation like synonyms or hyponyms. If the opposite is frequently used in the same n-grams, the word embedding would be close to it[32].

Word embeddings do not address the problem of featurizing larger utterances. Straightforward algorithms use concatenation or (weighted) summation of the single word vectors to represent larger parts of the text[35]. Those approaches are facing the same problem as BoW-models. The word order is ignored. More sophisticated models consider the structure of sentences and paragraphs. Le and Mikolov extend the language model with

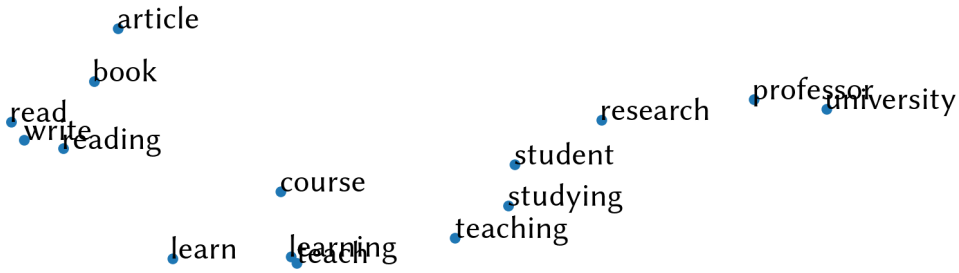


Figure 1.4.: A set of GloVe embeddings reduced to a two dimensional vector space with PCA. The visualization shows that related words are mapped to vectors with a small Euclidean norm.

an additional paragraph vector[30]. This model assumes that the local context of a word depends not only on the word itself but also on the paragraph.

Aside from those unsupervised features end-to-end approaches take the syntax of text into account as sequences for RNNs[26] and LSTMs[25] models or as concatenations for CNN[31] based approaches. Supervised NLU task like STS, which measures the degree of difference in meaning for two sentences, can produce features for further NLP tasks. Examples for STS systems are introduced by Tai et al. and Shao[53, 50]. Especially STS provides due to frequent tasks at the SemEval workshops many benchmark algorithms together with sufficient training data[8]. Recently transfer learning based on multiple NLP tasks emerged as an additional approach for general purpose sentence embeddings[9, 14].

1.2.2. Analysis

The second step for IR systems is analyzing the extracted features to group relevant documents. The grouping of NLU features can either use classification or clustering. The current state of the art for both methods is briefly discussed in the following paragraphs.

1.2.2.1. Classification

In IR every search can be seen as classification problem. The classification separates all documents into a group which matches an information need and another group of documents which do not match the information need[48]. Consequently, it groups the collection constantly into multiple groups. Each group satisfying one specific information need. This makes classification suitable for common search terms and splitting a collection into several known topics. This is also the reason why classification in IR is referred to as topic classification. Typical IR classification tasks are spam filtering or finding news stories about a certain topic. But tasks like sentimental analysis can be considered as retrieval tasks as well.

Most algorithms use word level feature to classify text. As already mentioned BoW feature are still popular for many text classification task, because of the simplicity. Often it is enough to just compare the occurrences of utterances to classify texts. Term frequency extensions are used to create feature invariant to text length and take the different information entropy of words into account. Typically naive Bayes[48, 38] or SVMs[27] are

used as classifier on the term frequency feature. Nevertheless the *curse of dimensionality* and the related sparse feature vectors are difficult for most analysis methods. Therefore the success of word embeddings in many NLP tasks is also notable in text classification. Recent approaches additionally encounter the sentence structure by using RNNs and CNNs. Socher et al. introduced tree structured RNNs according to dependency graphs to group the sentiment of movie reviews[51]. Aside from word level representation CNNs, which work for many tasks well on raw signals, are used to classify the collection on character level[58]. This approach could recognize the meaning of affixes, like *pre-* and *-ing* to derive the meaning of new words.

1.2.2.2. Clustering

In IR clustering is based on the so called *cluster hypothesis*. Schütze et al. describe the hypothesis as follows: "Documents in the same cluster behave similarly with respect to relevance to information needs"[48]. Accordingly, if one document of the cluster satisfies an information need, all other documents of the cluster most likely satisfy it as well. This hypothesis can be exploited for many use-cases in IR. One example is applying hierarchical clustering to stepwise refine the information need. At each level it is possible to select the cluster most suitable for the intended information need. This *scatter-gather* search avoids ambiguity of search queries[48]. Another example for clustering is speeding up the search process by dividing the collection into clusters and searching just the nearest clusters to the information need.

The groups are determined by the document collection itself, therefore it depends on features and the used clustering algorithm. Similar to text classification BoW and word embeddings are commonly used as features for clustering text. However more sophisticated text representations based on end-to-end deep learning approaches cannot be used in this unsupervised setting. Unsupervised features using transfer learning like the already mentioned universal sentence encoder proposed by Cer et al. are not commonly used either. Xu et al. describe another model explicitly for short text clustering[57]. They use pseudo labels obtained by simple dimensionality reduction to train a CNN and cluster this feature representations with k-means.

The actually used clustering algorithms is determined by the specific, demanded characteristics of the clustering task, therefore the clustering approaches are highly task specific. Commonly k-means models are used for topic clustering and *scatter-gather* searches often use single-linkage approaches[48]. But most applications require specialized clustering algorithms to take unknown cardinalities, partial or fuzzy cluster and prototype representations of clusters into account.

1.2.2.3. Use case: Duplicate detection

An application between clustering and classification is the detection of duplicates in the collection. This typical classification task, with two input texts and a Boolean *is-duplicate* label, also compares documents which is a common task for clustering algorithms. Accordingly, the duplicate can be defined as small cluster. In this case the behavior with respect to relevance to information needs within a cluster should not differ at all. The

duplicate detection for natural language is used in detecting plagiarism[59], filtering web crawls / search results[48] or to identify bug repots, which describe the same error[23]. Recent methods use mostly fingerprints, a set of substrings, or simple BoW based feature to compare documents. On one hand this is due to the performance restriction and on the other hand it is based on the need to detect only exact duplicates.

1.2.3. Current limitations

The summarized state of the art reveals a few limitations of NLU feature and IR. Based on the following depicted limitations the contributions of this work is going to be defined in the next section.

NLU feature The current research on the semantic feature space for IR tasks has two important shortcomings which will be addressed by this thesis. Firstly, the end-to-end approaches applied by many supervised NLP tasks like STS, sentiment analysis or question answering rely on training data and do not provide explicit sentence or even text representations. Therefore, unsupervised tasks have to fall back on simpler features. The recently proposed general purpose representations based on transfer learning tasks could solve this problem, but as of today those feature were not used for IR tasks. Secondly, feature taking the word ordering into account are trained and used mostly on grammatical correct sentences. In IR many documents do not contain just continuous text. Bullet points, keywords and other interruptions of text with a distinct grammatical structure are common. Consequentially the current research does not report results for those noise data.

Analysis Current text classification approaches use large dataset with a structure aligned to English grammar rules. Therefore, those models lack evaluation capabilities for less strictly structured text with smaller training datasets. Due to the predominantly unsupervised setting most IR approaches still rely on simple feature, which are obtained in an unsupervised fashion. Those term-frequency or word embeddings models fail to represent the combined meaning of words in a sentence. The lately introduced general purpose text representations from transfer-learning tasks are currently not investigated for the use within IR.

1.3. Focus

This work aims to structure the recent research efforts by comparing NLU concepts for featurizing semantics of text and applying it to IR. In accordance to this goal the main contribution of this thesis is a comparative study for semantic similarity measurements. The focus of this study is to identify useful patterns and heuristics for specific IR tasks with varying datasets.

The considered NLU features are unsupervised features and text representations generated by a transfer-learning task. The transfer feature will be based on a supervised deep learning approach which generates from the STS tasks a sentence representation. This

representation will be combined and normalized to create a feature for the whole text. The comparison of those feature to classical unsupervised feature aims to evaluate the potential use of transfer-learned features in traditionally unsupervised settings. Additionally, to this evaluation of supervised and unsupervised features this work will explore the effects of involving the linguistic structure. In many situations the structure of texts differs from strictly structured text according to English grammar. For example, the texts might contain bullet points. The influence of this noise on structural properties will be studied as well.

Information Retrieval uses a variety of methods and models as depicted in Section 1.2. This thesis will concentrate on the basic aspects of IR tasks. The feature vectors and the comparison between documents and information needs. Both aspects are required for IR clustering and classification systems. This restriction also reduces the required system components and fits the introduced use-case of duplicate detection.

The duplicate detection will use bug reports as test datasets. Large software projects are tested by many different users to ensure the functionality of the product. During testing multiple thousand bug reports are gathered. In order to manage those reports it is necessary to apply basic IR approaches to satisfy typical information needs that emerge while those errors are repaired. This use-case focuses on searching the collection based on an entity to retrieve similar errors. The knowledge of similar errors helps to find reports which describe the same bug. Those duplicates cause avoidable costs through multiple correction attempts. Furthermore, determine the search for similar reports who is best suited to fix this bug (the developer, who recently fixed similar bugs), what the potential cause of the problem could be (the same cause, which produced similar problems) and provides several other important insights.

The analysis of bug reports is also a project of Robert Bosch Engineering and Business Solutions Private Limited, which supports this work. One of their aims is to apply AI techniques, like IR to improve the efficiency of the overall software development process. This work contributes to this goal by comparing several unsupervised NLU features and improving the current duplicate detection for bug reports.

1.4. Outline

The remaining part of the thesis is organized in four chapters. The structure of those chapters are described as follows.

- In Chapter 2 required fundamentals are characterized. The chapter introduces basic ML algorithms, statistical NLP concepts and basic linguistic knowledge.
- The main part of this thesis delves into the following two key aspects of IR systems.

  NLU feature Chapter 3 looks at distributional features to quantify the meaning of texts. Unsupervised and transfer learning supervised feature representations are described.

  Information Retrieval The analysis in Chapter 4 focus on clustering algorithms in IR systems and the special case of duplicate detection.

- The detailed evaluation of the aforementioned algorithms is depicted in Chapter 5. This evaluation uses duplicate detection.

The concepts of the different approaches as well as the results of the evaluation are summarized in Chapter 6. This chapter also describes promising reference points for future work.

2. Fundamentals

This chapter covers the essentials used in consecutive chapters. With respect to this focus a selection of general machine learning algorithms (Section 2.2) and statistical NLP tasks are presented. The statistical NLP part in Section 2.3 is confined to word embeddings and STS as NLP benchmark task. In favor of increasing the understanding of language as the analyzed domain data before delving into the analysis part Section 2.1 introduces basic linguistic concepts.

2.1. Linguistics

Natural Language Processing is due to the focus on language data closely related to linguistics as the scientific field of studying languages. In order to provide the required data understanding basic linguistic knowledge is necessary.

The usage of a complex language distinguishes humans form all other species and is widely considered a trait of intelligence. The introduction chapter already established a close relation between our thinking process and the ability to express our thoughts through language. Therefore, language itself becomes a framework to order and structure our thoughts. Vice versa language is shaped by our brains (neurolinguistics), social influences (sociolinguistics) and of course the already existing language (historical linguistics). Those influencing factors emphasize the wide range of linguistics. The scope of this thesis is to understand the meaning of utterances, therefore the focus on this introduction lays on structural and overall semantical aspects of linguistics. Those two facets are required to categorize, understand and evaluate the NLU models that will be introduced within Chapter 3.

Structural and semantical aspects of language are often characterized as a stack of several different linguistic subfields based on a hierarchical view of language processing. Table 2.1 shows these different layers accompanied by a brief example sentence to illustrate the stepwise processing. Morphology and syntax analyze the structure of text. Those fields define rules, which are collected in grammars. Based on the structure semantics and pragmatics examine the meaning. The following sections depict grammatical rules and the encoded meaning.

2.1.1. Grammar

A grammar is a set of rules to construct proper words, phrases and sentences. Those rules are studied by the subfield's morphology and syntax. They focus on the word and text structure. Since the structure of a text is closely related to its meaning, we delve in the following paragraphs deeper into both subfields.

subfield		example sentence			
		The	*trash*	*is*	*full.*
Morphology		*is*: lemma of *be* (third person singular)			
Syntax	POS	determiner	noun	verb	adjective
	Dependency	*full* → *is*; *full* → *trash*; *trash* → *the*			
Semantics		object *trash* has property *full*			
Pragmatics		You should empty the trash.			

Table 2.1.: The different subfields of linguistics required to understand meaning of text. The example sentence emphasizes typical aspects of each subfield. It does not use any specific annotation.

Morphology This subfield is concerned about the structure of words. Words are constructed out of atomic units of meaning which are called morphemes. Most words consist of just one morpheme, like *dog* or *clean*. In order to alter the meaning, grammatical category (noun, verb, etc.) or other syntactical properties of words morphemes can be combined with affix morphemes[5]. The prefix *un-* can be added to *clean* to form the word *unclean*. This negates the meaning of the word *clean*. In general, the combination of morphemes can be classified into three classes.

- Inflections alter grammatical properties of words by adding affixes. A typical inflection is thereby the *-s* suffix to change a noun from singular to plural or *-ed* to change the tense of a verb. Inflections do not transform the grammatical category or general meaning of a word. All inflections are referred to as lemmas of one lexeme. This lexeme describes the principal meaning, therefore a lexeme is the root of a lemma and serves as a representative for the group of lemmas. The English language just contains eight inflections. All of them are suffixes.

- Derivations manipulate the meaning of a word with pre- or suffixes. In this process the grammatical category of the word is often changed. One example for this is the derivation *-er*, which transforms verbs into nouns. *Cleaner* describes a person who cleans for living. Another already briefly mentioned example is *unclean*. This derivation does not transform the grammatical category but changes the meaning more drastically than the derivation *-er* in *cleaner*.

- Finally compounding words are the last combination type. Inflections and derivations use affixes, which dependent on other morphemes to form a complete word. They cannot stand alone as words. Therefore, they are considered as bound morphemes. In opposition to these free morphemes can be used as separate words. Strictly speaking compounding words based on this property are not a combination of morphemes but a combination of words. A few English examples are *basketball* or *real estate*. Compounds refer to a new concept rather than a concatenation of the combined word meanings, therefore it is not possible to interpret just the separate morphemes. Especially for blank space separated open compounds, like *real estate* the interpretation becomes more challenging, because what appears to be two words is actually one.

Morphological analysis is used to derive the semantics of single words as well as to alter words to reflect details like the number or the temporal aspect of single words. The syntax of a language is built-up based on morpheme combinations.

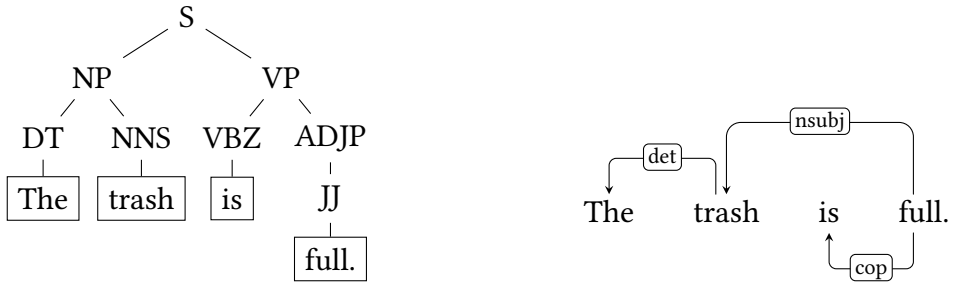
Syntax The ordering of words to form valid sentences is governed by syntax. Syntactical rules define phrases and sentences based on a grammatical categorization of words and the intended meaning.

Grammatical categorizations are a key concept of grammars. Similar behaving words with respect to phrase and sentence structures are grouped together[36]. Such groups are titled as Part Of Speech (POS). The major groups, like nouns, verbs, etc. are taught in elementary school, therefore the basic concept of POS tags is well-known. In linguistics a few additional POS tags are used, like *determiner* in Table 2.1. Words with same POS tags can be used as replacements for other words with the same tag without restructuring the phrase or sentence. This interchangeability indicates that the syntax, or grammatical correct structure is defined based on Part Of Speech.

The idea of generative grammars uses POS to define hierarchical rules to combine growing units. These set of rules generate for example out of the sequence (determiner, adjective, noun) a noun phrase. This phrase serve as a constituent and is combined with other constituents or POS until all constituents are combined into one single root. The root represents a sentence. The example sentence of Table 2.1 is depicted in Figure 2.1 as a tree according to this pattern. Combination rules are formalized as context-free grammars. Based on such a grammar it is not only possible to describe sentences, but also to generate sentences.

Another view on the structure of a sentence is given by dependency grammars. Instead of conflating the different parts a dependency grammar uses predefined, asymmetric relations between two phrases to determine the structure. One example of such a relation is the subject of a verb phrase. This relation defines who or what is described by the verb phrase. In Figure 2.1 subject relations are labeled with *nsubj*. In this case the verb phrase is the head and the noun is the dependent of the relation. In general, the dependent modifies, extends or just refers to the head[42]. Dependencies are more open for interpretation than constituency grammars, because it is not possible to define universal rules for them. In one sentence multiple dependencies are imaginable. It requires semantical knowledge to select the right dependency. Accordingly, data has to be manually labeled.

Both approaches establish a tree-hierarchy for sentences. Constituent structures store all words in the leaf-layer as opposed to dependency structure, where all nodes correspond to a word and edges are labeled. In the last years dependency grammars are frequently used for many NLP tasks, because it contains more semantical knowledge than constituency grammars, but both viewpoints are useful for specific tasks. For this thesis neither structure models are used explicitly, because the structure of a sentence will be learned with ANNs. Therefore, the details of both models are out-of-scope for this work and will not be discussed further.



(a) constituent structure (b) dependency structure

Figure 2.1.: An example of the constituent structure (a) and dependency structure (b). The Penn Treebank annotation-style is used to label the constituent structure[37]. The dependency structure uses Universal Dependencies[42]

2.1.2. Semantics

Semantics is the subfield of linguistics analyzing the meaning of words, sentences and utterances. As such it is an important part for NLU systems. According to the hierarchical structure understanding of text relies on morphology and syntax. Obviously, the meaning of words is closely related to morphology aspects. In addition to the simple word meaning utterances have to take syntax into account.

Lexical semantics Languages use a set of words. Those words refer to specific concepts, which define the meaning of a word. In order to understand a word, it is thereby necessary to know the corresponding concept. For concrete concepts like a *house* it is possible to connect word and concept by showing it, but this is not directly possible for abstract concepts. Those concepts have to refer to already existing concepts by defining relations to other words. Lexica or dictionaries describe a word in this fashion with varying, unspecific relations. Thesauri explicitly define the relations between words with metadata and are frequently used in formal systems to define words. Typical word relations are antonyms (*opposite* relation), hyponyms (*is-a* relation) or meronym (*part-of* relation)[36].

These relations are commonly used in morpheme combinations. The meaning of a word containing multiple morphemes can be derived from one morpheme altering the meaning of other morphemes. The word *unclean* refers to *clean* and states that it is *not* what is commonly associated with the word *clean* by adding the prefix *un-*. Adding the prefix *un-* to another morpheme, like *important* alters the meaning of this morpheme in a similar way. Many practical approaches spare analyzing the meaning on morpheme level and instead expand a thesaurus with important morpheme combinations. In this case words are the atomic meaning units.

Instead of explicitly defining word relations with a thesaurus it is also possible to find relations between words by analyzing the usage of words. This exploits implicit relations of co-occurring words and is called distributional semantics[19]. If two words occur frequently in the context of the same other words, it is reasonable to assume an affiliation between those words. For example, *student* and *professor* most likely occur often in the context of *university*. The obvious advantage of analyzing co-occurrences instead of

manually defining word relations is the omitted labeling effort. Distributional semantics requires only a large text corpus. At the same time those unspecific relations are not analyzed with respect to different types, like antonyms or hyponyms and therefore do not reach the annotational depth of thesauri approaches. Nevertheless are distributional semantics especially through word embeddings, as mentioned in the introductory chapter, regularly used in NLP tasks.

Since a word is defined by the context, in which it occurs, the same word can have different meanings in different contexts. This creates ambiguity. A *suit* can represent a *law suit* or a piece of cloth. Without knowing the context it is not possible to resolve this ambiguity, therefore the entire utterance contains more meaning than the single words. This open problem is word sense disambiguation and has to use compositional semantics to resolve the ambiguity.

Compositional semantics Aside from lexical semantics utterances carry additional meaning. Compositional semantics analyze these utterances to resolve word ambiguity and to establish relations between individual word meanings. Manning and Schütze formulate this condition in the following way:

> The meaning of the whole is the sum of the meanings of the part plus some additional semantic component that cannot be predicted from the parts.[36]

The sum of the parts provides overall contextual information about the general topic or domain of the utterance. Humans use this when skimming through a text to determine the topic of it. In this case the meaning of a few keywords without any relation between them is sufficient to solve the task. Knowing the topic of a text can be used to disambiguate word meaning and to associate a priori knowledge with the text.

The additional component arises out of the order of words and sentences. Syntactical dependencies specify word relations, like the object of a verb, on sentence level. Compositional semantics derive the meaning of the sentence by connecting those dependencies with the meaning of individual words. But the syntactical rules do not define unique dependencies for all sentences. In general, multiple interpretations of word dependencies are possible. This results in ambiguity on sentence level. In order to resolve the ambiguity, it is necessary to identify the most likely dependency according to the domain and previous sentences.

Utterances with multiple sentences contain additionally relations between sentences. If a noun was already mentioned in a previous sentence, it is likely that a pronoun is used to refer to it. This anaphoric relation is one example of cross sentence references. Those references do not follow a nested structure, like it is commonly used for word dependencies.

Pragmatics Language is always embedded into a wider context, for instance the relation between transmitter and receiver, previous discussions or simply the body language of a speaker. All these influencing factors can alter the literal meaning of text. The example in Table 2.1 assumes that transmitter and receiver are flatmates and one of them is responsible

for emptying the trash. Additional context changes the simple statement into an imperative sentence.

Comprehensive NLU requires an understanding of any context to identify sarcasm and idioms. In human-to-human communication the pragmatics layer refines or alters the meaning of many utterances and is an important part of human communication. Some areas try to avoid encoding information into the pragmatics layer to prevent misunderstandings because recipients might not share the same context required to understand the pragmatic layer. Business reports, product documentation or other reports are examples for this. During this work we are going to focus on the literal meaning of a text, because our focus lays on formal reports.

2.2. Machine Learning

Apart from linguistic knowledge NLP and especially statistical NLP requires pattern recognition as well. These methods identify the underlying structures, similarities or other patterns of the text corpus. Typically, those text patterns are due to their complexity not explicitly programmed, but automatically learned. This constitutes Machine Learning (ML) as the second closely related field.

Most current approaches applied in NLP depend on ANNs. Using word embeddings and BiLSTMs this work is no exception. Consequently, fundamentals about neural networks are briefly mentioned.

2.2.1. Artifical Neural Network

Instead of a deep dive into the general basics of Artifical Neural Networks (ANNs) and the variety of different networks, which is provided by many books and articles like [22], this section focuses on the two networks actually applied during the remaining work. The feed-forward networks is applied to classify representations, which are partially generated by RNNs. Nevertheless, the basic elements of networks will also be introduced briefly at this point.

Perceptron Atomically all neural networks consist out of kind of layered perceptrons. The perceptron classifier defines a linear function, which separates the vector space into two distinct classes depending on whether it is above or below the plane described by the linear function. It is a binary classifier. Expressed in a mathematical fashion the perceptron calculates with $f(\vec{x})$ defined in Equation 2.1 the output $\hat{y}$.

$$f(\vec{x}) = \hat{y} = \begin{cases} 0 & \text{if } \vec{w}\vec{x} + b \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.1}$$

The possible result (0, 1) represents the two classes. The input vector $\vec{x} \in \mathbb{R}^n$ contains n features of one instance which should be classified. The input vector $\vec{x}$ and the weights $\vec{w}$ are elementwise multiplied and summed. This is the dot product. The parameters $\vec{w}$ and b define the linear function. Those values are determined by optimizing a cost function based on input vectors and the corresponding output.

2.2.1.1. Feed-forward networks

In order to classify more complex data a single linear function is not sufficient, therefore perceptrons are layered. The result of one is the input of another perceptron. All networks consist out of one input layer, several hidden layers and one output layer. The size of the input layer is determined by the size of the initial feature vector $\vec{x}$ and the output size is correlated to the required classification task. If this sequence of perceptrons does not contain any loops, meaning that the output of one perceptron just affects the following perceptrons and not a previous one in addition, those networks are feed-forward networks.

Logically the perceptron has to be modified to be used as a node in a neural network. The transformations of the features in the layers are not restricted to one single real value output. Accordingly the one dimensional result has to be generalized to a m dimensional result by exchanging the vector $\vec{w}$ with a matrix $W \in \mathbb{R}^{m \times n}$. Consequently, the bias b has to be expanded into a bias vector $\vec{b} \in \mathbb{R}^m$. The matrix indicates the different connections between each unit in a layer. A network with a valid weight on all entries is fully connected.

Furthermore, the step function has to be changed as well. The step function serves a special purpose in this concatenation of perceptrons. Linear functions are closed under the chaining of linear functions. Without the nonlinearity part represented by the step function for perceptrons the concatenation of them would be equally powerful than one linear function. Therefore, this nonlinearity part to determine the output is important for all networks. But for actual networks the step function is substituted by a better fitting approximation, because the derivation of it has some unfavorable properties. The best case for optimization would be a small change in the outputs whenever the weights and biases are slightly altered. This allows a gradually adaption of the parameter. The derivation of the step function is always zero except for $x = 0$, where it is infinitely large. Therefore, the output of a perceptron does not change gradually with better fitting parameters. Nonlinearities like the sigmoid function or the hyperbolic tangent function are frequently used approximations with better fitting derivations.

Optimization After summarizing the general structure of a neural network in a rather short way, the optimization aspect or how those parameters are adapted remains an open issue, which will be briefly addressed. The optimization has to measure the performance of the current parameters and gradually adjust those parameters to improve them. This requires a cost function, which measures the performance. Typically, the probability distribution q of the network is compared to the approximated original probability distribution p. The cross entropy, depict in Equation 2.2, is a common cost / loss / objective function to compare those probabilities.

$$J(\Theta) = -\sum p(x) \log q(x) \tag{2.2}$$

The partial derivative of the cost function $J(\Theta)$ with respect to each parameter Θ describes the change in the output value, if the corresponding parameter is slightly adjusted. In order to minimize the cost all parameters are adjusted stepwise in negative gradient direction till a local minimum is found. This approach is called gradient descent. The

Equation 2.3 and Equation 2.4 describe this method in mathematical terms.

$$\Delta C = \frac{\delta J(\Theta)}{\delta \Theta} \tag{2.3}$$

$$\Theta_{new} = \Theta - \eta \Delta C \tag{2.4}$$

Figuratively speaking determines ΔC the direction in which the parameter have to be adjusted. To avoid jumping over a minimum the parameters are combined with a learning rate η, which decreases the step size. The partial derivative is calculated based on back-propagation, which speeds up the calculation by reusing already calculated gradients. The error is propagated reversely through the whole network.

Vanilla gradient descent uses all training examples to calculate one step, for many cases this is to inefficient, therefore more elaborated algorithms are applied. Stochastic gradient descent for example just estimates the true probability function based on a few examples[45]. These examples are mini-batches. Other approaches like AdaGrad[17] try additionally to improve the step size.

2.2.1.2. Recurrent Neural Network

In opposition to feed-forward networks it is also possible to allow a node to influence previous nodes. Those are Recurrent Neural Networks (RNNs). The recurrent connection allows to take the previous activation state into account thereby it is particular useful for time-related features like a word sequence. The previous state is saved as a so-called hidden state. The basic approach is visualized in Figure 2.2.

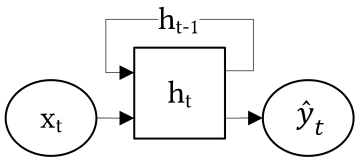


Figure 2.2.: The dependencies of output $\hat{y}_t$ to the hidden state h_t and the previous hidden state h_{t-1}.

The hidden state and the new input are frequently simply summed after multiplying the weight matrices. This is formulated in Equation 2.5. The actual output transforms the hidden state with the help of an additional weight matrix.

$$\begin{aligned} h(\vec{x}_t) &= \vec{h}_t = \sigma(W_h h(\vec{x}_{t-1}) + W_x \vec{x}_t) \\ f(\vec{x}_t) &= \hat{y}_t = \sigma(W_o \vec{h}_t) \end{aligned} \tag{2.5}$$

Those weight matrices are defined as $W_x \in \mathbb{R}^{l \times n}$, $W_h \in \mathbb{R}^{l \times l}$ and $W_o \in \mathbb{R}^{m \times l}$ with n denoting the input vector size, l the hidden vector size and m the output size. In contrast to the depicted equations the output and hidden state calculation does not need to use the same non-linearity σ. Instead other functions could be applied as well.

The training of RNNs simply unfolds the recurrent connection into a feed-forward network and applies backpropagation. This leads to deep virtual feed-forward networks

depending on the sequence length. With increasing network layers, the backpropagated gradients tend to vanish or explode depending on the nonlinearity component. If the gradients of this functions are commonly smaller than one the gradients vanish. Vice versa lead large gradients to exploding values[25]. This requires further improvements. Exploding gradients can be handled with cutting off large gradients[]. The vanishing gradients cannot be improved so easily, but LSTM provides a solution for this as well.

Long Short Term Memory Hochreiter and Schmidhuber introduced the Long Short Term Memory as an extension to solve the vanishing gradient problem[25]. The basic idea is adding a memory cell, which controls whether the current state or the hidden state should be taken stronger into account. This mechanism allows preserving the state over a longer time period. LSTMs are commonly used for many NLP tasks, therefore many slightly altered definition exists. The following overview of a LSTM unit is in line with Tai et al.[53].

The memory cell $\vec{c}_t$ is naturally defined by its previous state which is weighted by the forget gate $\vec{f}_t$ and the current state calculated from the input gate $\vec{i}_t$ and $\vec{u}_t$. This adjusted memory state and the output gate $\vec{o}_t$ determine the new hidden state $\vec{h}_t$. The output $\hat{y}_t$ is not altered compared to the RNN. This interdependencies are illustrated in Figure 2.3 and formalized in Equation 2.6.

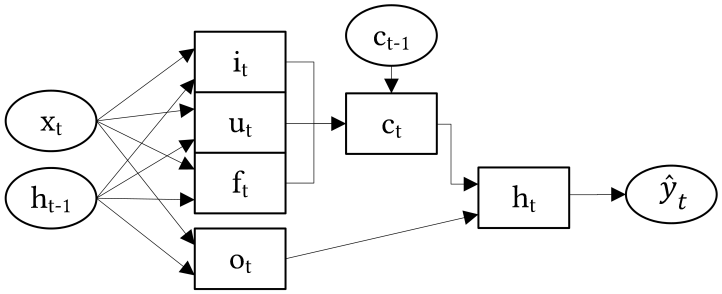


Figure 2.3.: Dependencies of the memory cell $\vec{c}_t$ and the hidden state $\vec{h}_t$.

The exact calculation of each part can differ in the applied nonlinearity and added biases. The equations are not discussed in detail, because of the rather limited benefit provided by understanding the details of a LSTM for this thesis. One exemplary definition of the different gates is:

$$\begin{aligned}
\vec{i}_t &= \sigma(W_{(hi)}\vec{h}_{t-1} + W_{(xi)}\vec{x}_t + \vec{b}_i) \\
\vec{f}_t &= \sigma(W_{(hf)}\vec{h}_{t-1} + W_{(xf)}\vec{x}_t + \vec{b}_f) \\
\vec{o}_t &= \sigma(W_{(ho)}\vec{h}_{t-1} + W_{(xo)}\vec{x}_t + \vec{b}_o) \\
\vec{u}_t &= \tanh(W_{(hu)}\vec{h}_{t-1} + W_{(xu)}\vec{x}_t + \vec{b}_u) \\
\vec{c}_t &= \vec{i}_t \odot \vec{u}_t + \vec{f}_t \odot \vec{c}_{t-1} \\
\vec{h}_t &= \vec{o}_t \odot \tanh(\vec{c}_t)
\end{aligned} \tag{2.6}$$

where $\odot$ denotes elementwise multiplication and the subscripts of the weights emphasize that for each gate a different weight matrix is multiplied. The functional expressions are due to the complexity and the minimal advantage of defining the input parameter left out.

The single LSTM units can be arranged in multiple layers or the signal can be processed in forward and backward direction (Bidirectional Long Short Term Memory). During the thesis such a BiLSTM calculates a sentence representation. Due to the complexity of the unit itself with current calculation power just small networks are trainable.

2.3. Statistical NLP

After introducing a few basic ML methods and conveying an impression of linguistic theories about semantics the fundamentals of NLP used in the following chapters can be introduced.

NLP is a large field ranging from simple text manipulation to complex understanding tasks like IR. As mentioned during the introductory chapter the focus of this work lays on text understanding. The basis for all analysis task to achieve understanding is the text representation. The representation of larger utterances, will be discussed in Chapter 3. As starting point for those more complex representations the encoding of words becomes a necessity. Those representations will be briefly explained in the following section, as well as non-semantic text representations and one evaluation task for sentence representations.

2.3.1. Word representations

The key factor for most understanding tasks is the representation of words. Consequently, the transformation of words into features and the applied enhancement methods for those features are fundamental.

A word is associated with a specific concept. Setting aside morphological aspects, like affixes, the concept is not related to any characteristic of this word other than the word itself. Due to this property just, the equality relation can be established for word comparisons. Words can be considered nominal data. The encoding of words into a vector must take this into account, therefore the mainly used initial representation for a word in vector space is a one-hot encoding. This representation associates each dimension of a vector with one word. The corresponding dimension of a word is set to one and all other vector values to zero to represent a word. The vector depends on the vocabulary associated with the vector's dimensions. This initial representation is used for basic text representations, like BoW as well as for processing more complex word representations.

The one-hot encoding has two main drawbacks. Firstly, typical languages require a large vocabulary, therefore those vectors have a high dimensionality and are sparse, which is challenging to handle for most ML methods. Secondly, the vectors do not allow the definition of any distance or similarity with respect to the lexical semantic between words. Word embeddings aim to annihilate those problems.

Word embeddings exploit distributional semantics to define a vector representation. As mentioned distributional semantics defines a word based on the co-occurring words. Expressed in a more mathematical way the representations of those words must be correlated.

If features are correlated it is possible to predict with one feature the other feature. Based on this reasoning word embeddings incorporate distributional semantics by predicting one word (center word) based on the surrounding words (context words) or vice versa. This prediction task is a language model. Multiple approaches were proposed based on this conceptual idea. Word2Vec and GloVe are due to their performance on the word analogy task (intrinsic evaluation task for lexical semantics) frequently used.

Word2Vec The Word2Vec approach, introduced by Mikolov et al. optimizes a neural network to predict context or center words.

- Skip-gram: Prediction of context words based on the center word.
- Continuous Bag of Words: Prediction of center word based on the sum of all context words.

Rare words are generally more unlikely to occur as center word, therefore the CBoW model is not able to train those word embeddings quite as accurate as the skip-gram model. Generally does skip-gram require less data, because it generates from n context words n training examples while the CBoW model obtains just one training example. But the CBoW model is less computational costly and generates a better representation for frequent words[39]. The basic process is quite similar between skip-gram and CBoW. Additionally, due to the smaller evaluation corpus just the skip-gram model is later on considered therefore the following description focuses on the skip-gram approach.

Predicting context words entails maximizing a probability distribution for all actual context words based on the complementary center word representations. The probability of each word is defined in Equation 2.7. Where o denotes the index of the context word, c the index of the center word and $\vec{u}_o$ respectively $\vec{v}_c$ the corresponding representations for context or center words. Those representations are stored in a weight matrix of a neural network. One-hot vector selects the column to produce $\vec{u}_o$ respectively $\vec{v}_c$ vectors. The final word representation is the sum of context and center representation of each word. The separated two internal matrices get lost, which creates a problem for training pretrained models further on because the internal weights are not published for the most common pretrained datasets.

$$p(o|c) = \frac{\exp(\vec{u_o}^T \vec{v}_c)}{\sum_{w=1}^{V} \exp(\vec{u_w}^T \vec{v}_c)} \tag{2.7}$$

Those representations are combined with a dot product and converted into a probability by the softmax function. The dot product generates high values for similar vectors. Therefore, the probability is high if context and center representations are similar or related. The softmax function is the default way in ML to generate a probability distribution.

The optimization of those weights requires a cost function. Word2Vec uses the cross entropy defined in Equation 2.8 for that. T denotes the length of the corpus, w_t is the word index of the word on position t. The actual probability is denoted by $q(\cdot)$. This cost function can be optimized by typical methods like gradient descent to train the involved

weight parameters.

$$J(\Theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m<j<m \\ j\neq 0}} q(w_{t+j}|w_t)\log p(w_{t+j}|w_t) \tag{2.8}$$

Nevertheless, due to the high number of parameters, a vocabulary of 10,000 words with 300 dimensional representations has 6,000,000 weight parameters, the training has to be faster to efficiently calculate representations. Mikolov et al. propose three improvements to achieve this.

- Subsampling: Frequent words in the corpus are trained unnecessarily often, especially considering the smaller information value of those words. Therefore, not all occurrences of those words have to be used for training. Word2Vec assigns a probability to each word whether it should be considered or not. This dropout probability for each word is defined in Equation 2.9.

$$p(w_i) = 1 - \sqrt{\frac{t}{|w_i|)}} \tag{2.9}$$

  Thereby $|w_i|$ denotes the overall occurrence of the word w_i. The hyperparameter t adjusts how fast this function converges to a probability of 1 to drop the word. A small value for t results in a high dropout probability for smaller counts.

- Negative sampling: Another training aspect is the large amount of negative prediction samples. For each training word pair, the center word weights together with all weights of context word representations are adjusted. For the earlier introduced small example those would already be 3,000,300 parameters for each step. The sum in the denominator in Equation 2.7 is responsible for this high number of adjusted parameters. Each step provides one positive example and $|V| - 1$ negative examples to train the context word representations. The information value that one word does not occur in the context of a specific center word is low but is trained significantly more often than any positive example. Those negative examples can be reduced to decrease the number of adjusted parameters. Negative sampling approximates the softmax function with a few negative examples, instead of all negative examples. The details are explained by Goldberg and Levy[21].

- Hierarchical softmax: Alternatively, to negative sampling the softmax function can be approximated by hierarchical softmax. The hierarchical softmax organizes the probability distribution into a tree-structure. The probability and embedding of one word is calculated by multiplying them along the path. Logically this acceleration procedure increases the memory complexity, because the intermediate representations have to be also stored. Due to the complexity of this approach a detailed description is avoided in this work. Further information is provided by Rong[47]. Mikolov et al. use a binary Huffman tree to generate a tree-structure for the embeddings. This not only minimizes the trained weights for frequent words, but also groups words with similar occurrence frequencies. The grouping

introduces a dependency between representation and tree position. Therefore, the grouping is an important aspect of this approximation. The Huffman tree showed compelling results. Other approaches train on negative sampling or the original softmax and cluster the embeddings to generate groups of approximated similar word embeddings[41].

GloVe The optimization on different local text windows instead of a single global co-occurrence matrix decreases the performance and is subsequently a disadvantage of the Word2Vec approach compared to LSA or other co-occurrence matrix based models. Pennington et al. propose the GloVe model to solve this problem[43].

They defined the cost function as shown in a simplified form in Equation 2.10[43]. The core of this objective function is the dot product as well. But instead of minimizing the occurrence step by step the overall co-occurrence $P_{i,j}$ of the word i in the context of j is taken into account. The logarithm of the count increases with larger co-occurrences. In order to minimize the function, the dot-product has to increase as well, which is the case for very similar vectors as previously mentioned.

$$J(\Theta) = \sum_{i,j=1}^{W} f(P_{i,j})(\vec{u}_i^{\,T}\vec{v}_j + b_i + b_j - \log P_{i,j})^2 \tag{2.10}$$

GloVe has the same drawbacks as Word2Vec for frequent words. To address this issue the squared error is weighted by function $f(\cdot)$. It incorporates the word occurrence and should be relatively small for large values. Pennington et al. propose the in Equation 2.11 presented function[43].

$$f(x) = \begin{cases} (\frac{x}{x_{\max}})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{2.11}$$

Identical to Word2Vec the GloVe model is widely used and described in more detail in many papers, therefore the details are of this model are not discussed further on at that point. More details can be found in the original paper[43].

Training Both models do not require labeled data but must be trained on corpus data. Logically the word usage of this corpus defines the representation and requires enough occurrences of each word to define the meaning. Accordingly, a large training corpus provides better embeddings. On the other side due to word ambiguity and text-specific word usage the training dataset must represent the word distribution of the actually analyzed text, too. Those two aspects often contradict each other. Depending on the task the training on embeddings based on a small domain specific corpus or the usage of pretrained embeddings using a dataset containing billions of words can be more efficient. Conceptually domain specific training performs better, if the word distribution differs substantially from the pretrained embedding corpora. Those are trained on Wikipedia articles, news datasets or any text found by a web crawler, therefore the encoded lexical semantic is rather general.

Another training aspect of neural network models, like the reviewed Word2Vec approach is the backpropagation to alter the word embeddings based on the actual classification

task. The representation calculated from the language model serves as initialization and the backpropagation used to train the classification model is appended to also train the embeddings. This emphasizes task specific aspects of the lexical semantics. This simple transfer learning approach for word embeddings requires more training data and computing power for the actual classification task.

N-gram model Compounding words or more specific open compounds, described in Section 2.1 are particularly challenging for word representation because the identification whether multiple words are actually a compound requires extensive world knowledge. For example *The Wall Street Journal* is a compound and should be encoded as just one word. However, this requires the knowledge that a newspaper with such a name exists. Instead of creating large dictionaries for those compounding words, the frequency of specific word combinations within a corpus can be used as a heuristic approach to identify those compounding words. Therefore, the frequency of a sequence of N words is analyzed.

N-grams are not considered during this thesis due to the additional time costs and the increase in complexity for the overall system (larger vocabulary, additional erroneous component), accordingly a detailed description is not necessary. Principally a reasonably well performing N-gram model should increase the overall performance, while a decrease would be highly unlikely.

2.3.2. Text representation: Bag of Words

Beyond representing single words larger utterances require a representation as well. Especially NLU features cannot be a simple combination of single word features due to the compositional aspect of semantics. The Bag of Words (BoW) model does not apply any semantics at all. Consequently it is together with derivations of the BoW model presented as fundamental model. Later the TF-IDF approach is used as a baseline.

The BoW model accumulates all one-hot vectors. The resulting representation obviously comprises the occurrence of each word in the vocabulary for this text. Such a word count provides especially for topic detection valuable results because according to linguistic concepts the pure word sum provides the general topic, similar to the human skim through process recognizing just single keywords to detect the general topic.

The major drawback of simple word counts as provided by BoW is the dependency on the text length. This problem can be avoided by using length invariant metrices, like the cosine similarity or by normalizing the vectors. The normalized vectors are defined in Equation 2.12 and coined Term Frequency (TF)[]. Where $t \in d$ denotes the terms/ words and $d = (t_1, t_2, t_3, \ldots, t_N)$ denotes a list of all terms within one document. The operator $\|\cdot\|$ used in Equation 2.9 for the word embeddings as well, denotes the number of occurrence of the term in d. Commonly both approaches are applied.

$$TF(t, d) = \frac{\|t\|}{\sum_{\hat{t} \in d} \left\|\hat{t}\right\|} \tag{2.12}$$

From the perspective of Shannon's information theory different words do not carry an equal amount of information. Some words, like stop words (*the*, *a*) occur more frequently

over all texts of the corpus and on the other hand some words are unique for one specific topic. The comparison between two representations has to take this into account. Therefore, each term has to be weighted depending on the entropy of it. The weights are the Inverse Document Frequency (IDF). The Equation 2.13 depicts the calculation of this weighting factor. Hereby denotes D the overall corpus, consequently is $\|D\|$ the number of all texts d and the denominator counts the actual documents containing the term t.

$$IDF(t, D) = \log \frac{\|D\|}{\|d \in D : t \in d\|} \tag{2.13}$$

Applying IDF as a factor for each dimension of the TF feature magnifies the differences for important words and diminishes unimportant words. This combination is intuitively named TF-IDF.

2.3.3. Semantic Textual Similarity

Semantic Textual Similarity (STS) is an intrinsic evaluation task to validate the semantic understanding of a sentence. Intrinsic tasks are supposed to focus on one subtask of more complex models. For STS this subtask is the semantic representation of sentences. Another example for intrinsic evaluations are word analogy tasks which are frequently used to evaluate word embeddings[39]. From a linguistic perspective word analogy tasks evaluate lexical semantics and STS is one of multiple intrinsic task to evaluate compositional semantics on sentence level. As such it is correlated to many different actual NLP tasks, like machine translation or text classification and can subsequently be used as intrinsic evaluation task for sentence representations of those systems. During the course of this thesis STS is not considered as evaluation task, but to exploit the entangled semantic aspects to train sentence embeddings for one NLU feature.

The STS task compares two sentences regarding the semantics of each sentence and classifies the relatedness of the sentences. This classification requires the understanding of the sentences. In opposition of the natural language interference task, which classifies two sentences in *entailment*, *neutral* and *contradiction*[6]. STS captures the degree of similarity. Those six distinguished levels of sentence similarity are presented in Table 2.2 together with one example for each level.

Dataset Due to the importance of semantic sentence representation many STS datasets exists. Especially the SemEval workshops contributed in the last years many datasets. One recently introduced dataset is STS Benchmark[8]. The authors Cer et al. combined several STS datasets from SemEval workshops of the last years. It contains 8,628 labeled sentence pairs. The sentences are gathered from news agencies, online forums and image/video captions. Those sentence pairs are labeled by humans. The similarity levels allow interpretation, therefore human judgment is not unique either. To avoid any bias each pair is labeled multiple times. The gold standard labels are the average of those multiple labels. Due to this process the actual label within the dataset is continuous.

Benchmarks As intrinsic evaluation task STS is suited to benchmark different approaches. The mentioned STS Benchmark focuses on as the name already suggests providing bench-

Class	Description	Example
0	Sentences are not similar at all.	Someone is slicing tortila's.
		Someone is riding a horse.
1	Sentences are not similar, but have the same topic.	A man is playing the piano.
		A woman is playing the violin.
2	Sentences are not similar, but have some details in common.	A woman is peeling a potato.
		A woman is peeling an apple.
3	Sentences are mostly similar, differing in an important aspect.	The turtle followed the fish.
		A sea turtle is hunting for fish.
4	Sentences are mostly similar, differing in an unimportant aspect.	A man is eating a banana by a tree.
		A man is eating a banana.
5	Sentences are completly similar.	A plane is taking off.
		An air plane is taking off.

Table 2.2.: Description of the six different similarity classes used for the STS task. The descriptions are taken over from Cer et al.[8].

marks. Cer et al. published the performance of most recent approaches for sentence embeddings on this dataset[8]. Additionally, the current state of the art benchmarks are presented at the online wiki[1]. The performance is measured and compared by linear correlation of labels and predictions with Pearson's r, typically scaled by 100.

Avoiding a deeper look into particular benchmarks generalizing the currently best performing approaches depend on handcrafted features combined with ANNs. Currently the best approach scores a Pearsons's r value of 0.81[8]. Recently pure ANN methods, especially models using transfer learning approach similar values. In Section 3.5 the BiLSTM model proposed by Tai et al. for the STS task is used to train general sentence embeddings. This method is supervised and achieves a Pearsons's r value of 0.711 on the test dataset of STS Benchmark[8].

[1] http://ixa2.si.ehu.es/stswiki/index.php/STSbenchmark

3. NLU feature

In order to analyze text with regard to the inherent meaning it is necessary to quantify text attributes which denote the meaning of text. The quantified attributes are NLU features. This chapter gives in Section 3.1 an overview of such features and introduces a brief taxonomy of common feature extraction methods for natural language. Afterwards the chapter dwells on details of three specific algorithms for creating a NLU feature space in sections 3.2 to 3.5.

3.1. Overview

Natural Language Understanding is a large research field with an even wider range of tasks, each of them with an unique set of requirements and varying definitions of *understanding*. Translating a text requires another form of understanding than identifying the overall topic of a text or answering questions. Based on those differences each task also requires specific NLU features. This is the reason why many features with divided characteristics exist. In order to select, evaluate and even understand those features it is useful to categorize them. Consequently, a brief taxonomy based on linguistic fundamentals and the required training effort is in the following section introduced. Similar to the taxonomy linguistics allows to define the principal structure of extracting NLU features which is illustrated below.

3.1.1. Taxonomy

The introduced taxonomy relies on three main aspects: way of assigning meaning, type of exploited semantics and the required input data. The first two aspects engage a lexical point of view. The third aspect is a typical ML property to categorize models. In Figure 1.1 the classification with a selection of algorithms for each category is shown. The different aspects are described in the following paragraphs. The taxonomy constitutes the selection of the three in detail analyzed algorithms, which is discussed in the last paragraph.

Definition of Meaning The used descriptions to define the meaning is a key distinguishing characteristic. It is displayed on the first level of Figure 3.1. The meaning can be explicitly denoted by a thesaurus which requires a large collection or by implicitly exploiting the distributional hypothesis. Distributional models require instead of a priori defined relations between certain words a large text corpus to obtain the implicit relations. Thesauri-based approaches formulate specific rules based on the a priori relations. The features are nominal and it is only possible to analyze words within a thesaurus with predefined relations. Naturally words are nominal. Distributional semantics introduces a notion of

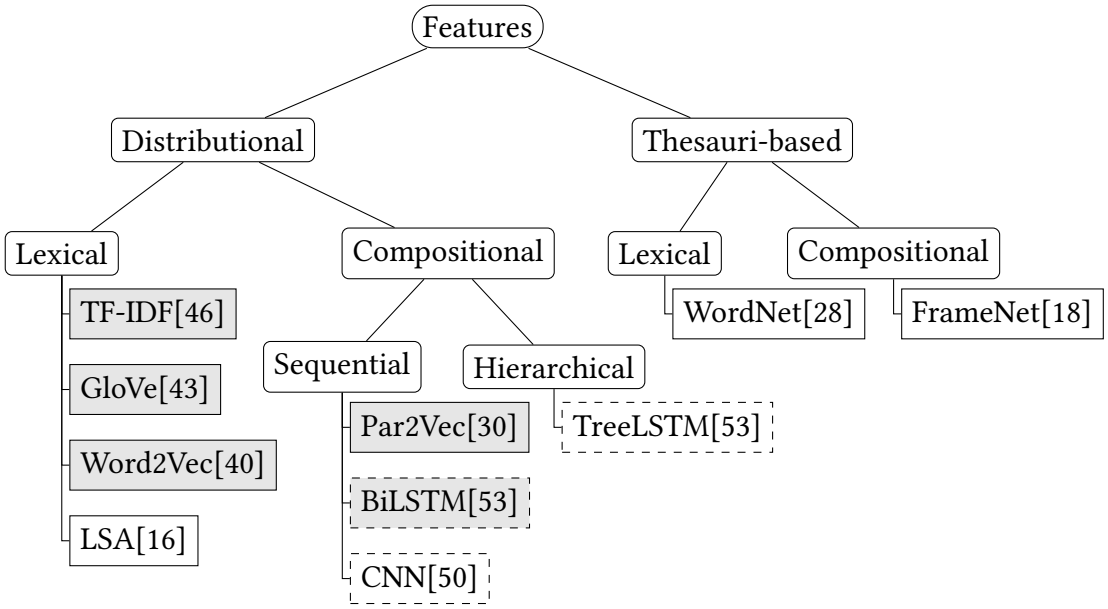


Figure 3.1.: Hierarchical overview of NLU features. The dashed boxes indicate supervised models, all other models are unsupervised. Models which are investigated in this work are highlighted in grey.

similarity based on the differences of the word co-occurrences. This is used by distributional models to generate numerical features. Co-occurrences are not able to distinguish word relations and therefore use vague relations.

Context Aside from the principal gathering of meaning certain feature also take varying context into account. Some methods just use lexical semantics and other analyze additionally the structure to obtain compositional semantics as well. The benefit of compositional semantics depends on the structure of input data and the task. Tasks like topic detection do not rely on details obtained by a larger context. In general, distributional and thesauri-based approaches can observe compositional semantics.

Thesauri-based approaches append the relations used to compare different words with relations of the context in which the word is used. Those annotations are frames based on frame semantics. For example, a sentence containing the word *studying* often contains the *duration* or the *subject* in the same sentence. Based on the frame it is possible to connect both words and capture the meaning of the whole sentence.

Distributional features encode the lexical meaning to vectors. The compositional meaning is than calculated out of a sequence of vectors for the lexical meaning. The typical dependency-structure of sentences makes it possible to divide this sequence analysis into the following two categories:

- Based on the perception of humans the vectors are read in sequential order. The linguistic structure of a sentence is either ignored or must be learned by the model.

- Instead of the sequential order the syntactical structure of a sentence is used to extract the features. The increased structure of the input data reduces the required training data in cases where the NLP task requires syntax.

Training data Apart from the linguistic categorization one important factor for all features is the required data to learn them. Initially all methods use *one-hot* encoded, concatenated vectors or conceptually similar representation as feature. The feature learning transforms those simple features into more sophisticated ones with lower dimensionality to avoid overfitting and to reduce training time. Those transformations could either be unsupervised or supervised. In Figure 3.1 supervised models are marked with dashed rectangles.

The thesauri-based approaches mostly rely on manual feature engineering. Therefore, it does not use feature learning at all. Subsequently no additional training data is required. The landscape of distributional features is more divided. Traditional models use unsupervised methods, like PCA or SVD to reduce the dimensionality. Word embeddings use language models and thereby are basically also unsupervised. Finally typical deep learning approaches, like CNNs or RNNs integrate the feature learning into the classification task. Essentially each layer of those networks represents features. To learn those features labeled data is required. The features are closely related to the classification task they are trained on and do not provide any theoretical insight.

Selected methods The in-depth described models are marked with grey filled rectangles in Figure 3.1. TF-IDF is used as baseline model. It compares the plain word occurrences. As such it does not make use of any semantical knowledge.

Thesauri-based approaches are not considered, because differentiating between special word relations is not required and distributional models evolved as quasi-standard for the vast majority of NLP tasks. In addition, creating a suitable thesaurus based on the specific domain data is time-consuming, cost expensive and would require extensive domain and linguistic knowledge. As a consequence, the subsequently presented methods are restricted to distributional semantics.

The first algorithm takes plain lexical semantics into account. It uses simple word embeddings to be more specific Word2Vec and GloVe as features. This model is described in Section 3.3. Word embeddings are frequently used as word representations and are the basis of many compositional features as well. Hence, they are suitable for comparing lexical and compositional models. The remaining two algorithms Par2Vec (Section 3.4) and BiLSTM (Section 3.5) use compositional semantics based on word embeddings. Additionally Par2Vec is an unsupervised method and BiLSTM uses a supervised task to learn features. This difference gives insights in the applicability of transfer features compared to unsupervised features. Due to the close similarity between sequential and hierarchical features, especially comparing BiLSTMs and TreeLSTMs and the time limitation of this thesis hierarchical feature are not considered. The TreeLSTM algorithm shows for the STS task an improvement of 1% in the Person's r correlation compared to BiLSTMs models[53]. Therefore, it is a reasonable assumption that in the given setting both models will provide similar results.

3.1.2. Process pipeline

The extraction of NLU features has to follow a general structure that is based on the linguistic notion of semantics. Compositional semantics requires lexical semantics to convey the overall meaning. Additionally, based on the strong grammatical dependencies within a sentence and more loose relations across sentences compositional semantics can be distinguished between sentence and paragraph, article, etc. This generates the intuitive structure shown in Figure 3.2.

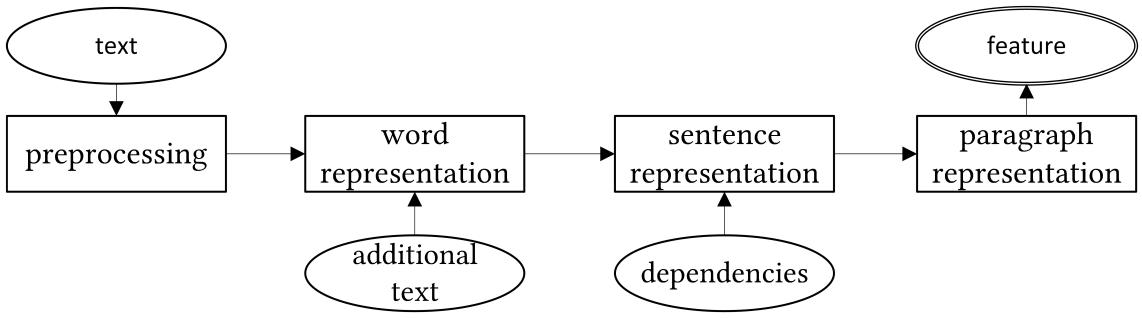


Figure 3.2.: The basic feature extraction pipeline for all described NLU features of this thesis.

Preprocessing Before actually extracting features the text has to be divided into sentences and words. Additionally, data cleaning, like removing unimportant characters from the text is applied in many cases as well.

Word representation The word representations encodes based on the separated words the aspects of lexical semantics. The result of this step is one feature vector for each word. Distributional semantics requires a large text corpus to generate an accurate representation. Therefore additional text similar to the input text is commonly used to improve those feature vectors.

Sentence representation Based on the sequence of word representations the sentence representation is calculated. Depending on the algorithm this calculation just combines the word representations or uses additionally the word order or even the syntactical dependencies. Taking advanced structures into consideration requires to conduct further steps, e.g. POS tagging or dependency trees.

Paragraph representation Finally the sentence representation is combined into a representation for larger utterances. This is the NLU feature used for further analysis, like classification tasks. The meaning of an utterance is predominantly determined by the meaning of each sentence. Sentence relations do not follow the same strict structure of sentence structure. Therefore, it is more complex to find those relations. Consequently they are seldom used.

All succeeding described features embody this basic structure with some specifics. Especially the preprocessing step is shared by all presented methods with minor changes,

like skipping some data reduction steps. Therefore, the preprocessing is placed in front of the feature descriptions and presented in the following section.

3.2. Data Preprocessing

The preprocessing for extracting NLU feature has two main objectives: Separating the text into featurizable units and cleaning the input data. The separation is arguable required for extracting any feature based on the introduced notion of hierarchical semantics. This tokenization is independent of the actual features. It provides the elements to calculate those features. On the contrary data cleansing relies on the actual extracted features, because it changes key characteristics of the dataset to increase the generalizability and adopts to the requirements of the feature extraction models.

3.2.1. Tokenization

Splitting the input text yields the parts used to calculate features. The introduced general pipeline to extract NLU features uses words as atomic units and groups those into sentences and finally into a larger utterance. Thereby tokenization must divide the text into words, sentences and in some cases into larger utterances like paragraphs. Since the separation into paragraphs is not required for the short texts analyzed during this study it is not described in detail. Additionally, paragraphs depend heavily on the used convention, e.g. two-line feeds as separator. Those conventions lack of ambiguity and are consequently easy to separate.

3.2.1.1. Sentence boundary detection

The task of separating single sentences is called sentence boundary detection. From a linguistic point of view, it is necessary to distinguish between different sentences because words within a sentence depend on each other to define the meaning of the overall sentence.

At the first glance the separation of written sentences seems rather simple. English, as most languages, uses a period *(.)* to separate two sentences. Consequently, sentences could be split by just searching for periods. Of course, this heuristic would have already failed to split the last sentence since it contains a period as part of the content and not as the delimiter of a sentence. Periods are not exclusively used to mark the end of a sentence. Among others they mark abbreviations, initials and are used within floating point numbers[29]. Sentence boundary detection has to classify the occurrence of all periods to detect those used as sentence delimiter. Apparently such a classification task can be solved with typical ML methods. Abbreviations is the most difficult class as they are based on the occurrence frequency. Complementary question marks *(?)* or exclamation marks *(!)* can be used to end a sentence as well. In general, those symbols contain less ambiguity than a period. Therefore, the classification of periods remains the most important part to disambiguate sentence boundaries.

In many cases heuristic rule-based systems already provide enough accuracy. However, the detection of abbreviations in rule-based systems relies on pre-defined word sets which

introduce the need of additional resources, similar to supervised approaches. Unsupervised methods do not rely on pre-defined abbreviations. They generate an abbreviation vocabulary based on a few properties of abbreviations. Naturally the accuracy of those models tends to be lower than methods using additional resources. In the case of extracting semantic knowledge for information retrieval tasks based on lexical and unannotated compositional approaches the accuracy of sentence boundary detection does not need to exceed the precision of those methods, due to the fact that syntactical context is rather limited for most by this considered methods.

One example for unsupervised models, which produces competitive results, is the *Punkt* algorithm proposed by Kiss and Strunk[29]. It is implemented in NLTK and will be used within this work[2]. The key characteristic of identifying abbreviations uses the assumption that they always have a final period, they tend to be short and also frequently have internal word periods[29]. Aside from these basic properties to understand possible shortcomings the details of this algorithm are not important to understand and are consequently not introduced here.

3.2.1.2. Word segmentation

Word separation is necessary, because they are used as the atomic meaning units and subsequently are the fundamental parts of any NLU feature.

After inflating the complexity of sentence boundary detection in the last section based on ambiguity this thesis continues doing so with the problem of detecting single words. Fortunately, the whitespaces used as word separators in most languages do not have multiple functions. However, exceptions introduce ambiguity for word segmentation. The basic rule that words are embedded into whitespaces does not apply for the following cases:

- English contractions, like *I'll* or *isn't* are in each case two words without a whitespace in between. Treating them as one single word creates especially for syntactical rules like constituent grammars an issue, because this word could be part of two-word classes[48].

- Another simple case is punctuation. Most full stops, commas and other punctuation marks are attached to a word and would consequently be considers as a part of that word. Preferably this should not be the case, because it would assume two different meanings for a word with punctuation marks and the same word without it. Strictly speaking punctuation marks are no words. This is the reason why those parts of this whitespace separation are commonly referred to as tokens.

- Sometimes hyphenation is used to separate two words as well. In general hyphenation is an ambiguous word delimiter, because some words containing a hyphen are just one word[48]. An example for this ambiguity is *e-mail*, which is just one word and *text-based* which are basically two words.

- Lastly compounding words create ambiguity with open compounds as well. Open compounds are treated syntactically as one word and have one distinct meaning,

nevertheless they are divided by whitespaces. Consequently, an open compound must be one distinct token.

Contraction and punctuation can be found with a regular expression. Hyphenation and open compounds are more difficult to filter out. For most use-cases hyphenations are treated as one word. Generally, hyphenations do not face the same uncertain word classes as contractions do. Open compounds are merged into one token by n-gram approaches. They analyze the occurrence of word sequences. During the course of this thesis the NLTK implementation for tokenizing words is used. It applies filter to distinguish between contraction and punctuation[2].

3.2.2. Data cleansing

One crucial part for all practical machine learning tasks is the data quality. Typical real-world data has a low quality due to missing measurements, faulty measurements or simple inaccurate measurements. These errors introduce unwanted noise. Therefore, data cleansing is required to improve the quality and boost the final outcome. Textual data is no exception to this general rule.

In addition to data specific cleaning steps textual data faces commonly a problem with dimensionality. Two properties of text contribute to this problem. On one side some words occur frequently in almost any text without adding information on the other side some words are so unique that they do not occur in other texts. Both aspects lead to a larger vocabulary and consequently to more dimensions in the initial BoW model. Therefore the *curse of dimensionality* is supplementary handled by removing and generalizing specific tokens.

3.2.2.1. Token removal

Removing irrelevant or redundant features is frequently used for most ML methods. At this point tokens are the most basic features of text. Consequently, irrelevant tokens are removed before more complex features are calculated. The removal of tokens depends on the actual task and is thereby a domain specific task. The three token types listed below are commonly removed.

Numbers The meaning of a number depends on the context and format of it. This makes a number particular difficult to interpret. For instance, numbers might be nominal or numerical and different units can measure the same property as meter and centimeter. This required knowledge is difficult to acquire and numbers cannot be represented in the same way words are encoded, therefore numbers are predominantly ignored or replaced with one special token for all numbers. Some specific methods introduce additional hand-crafted rules for numbers.

Special Characters Apart from punctuation marks many texts use other special characters as well. Commas, parenthesis, bullet points, etc. are used to structure a text. Logically those structure elements do not carry any meaning if the structure is disregarded. Nevertheless, certain texts assign some meaning to certain sequences of special characters. The most prevalent case is text emoticons.

Stop words Not all words contribute the same amount of meaning, therefore words not contributing to the requested meaning can be ignored. In contradiction to the TF-IDF algorithm, which solely uses information entropy to take this difference into account, stop words are identified through a manually assembled corpus and are entirely removed. Typical stop word corpora contain words like *the*, *to* and *a*.

Regular expressions are used to remove number tokens and tokens containing special characters. Stop words are removed based on a NLTK corpus containing 179 stop words. The corresponding statements are displayed in Listing 3.1.

```
import re
import nltk

stopWords = set(nltk.corpus.stopwords.words('english'))

def isValidToken(token):
  return not( re.match(r'.*\d+.*',token) #\d digit
              or re.match(r'\W+',token) #\W non-alphanumeric
              or token in stopWords )
```

Listing 3.1: Python implementation for removing numbers and special characters based on a regular expression. Additionally stop words are removed using a NLTK set

Due to the difficult interpretation and the small expected information gain for representing the meaning of a whole paragraph numbers are removed for the whole evaluation. This restriction abolishes the need to use an additional representation for numbers. Additionally, the removal of special characters does not distinguish between those denoting lexical meaning and those just structuring text. The evaluated datasets are reports and other formal documents in which emoticons or similar tokens are not used.

3.2.2.2. Token generalization

Comparable to the aggregation of data objects it is possible to reduce the distinguished words by grouping them. As mentioned in Section 2.1 many words are appended with affixes to alter grammatical properties. Tempus, numbers and other grammatical details are in most cases not important to determine the overall meaning of a paragraph accordingly those lemmas can be transformed into the corresponding lexeme.

The recovering of lexemes is called lemmatization. The word usage and the context are analyzed to determine the lexeme. For example POS tags have to distinguish between *meeting* as a verb or a noun because the lexeme of the verb is *meet* but the noun should not be changed at all. This kind of morphological analysis is a complex and time-consuming task, therefore stemming is often considered as a heuristic alternative. Stemming only analyses the word itself and tries to reduce the word to its stem. This stem differs from the lexeme. Lemmatization focuses on converting lemmas to lexemes by removing inflections while stemming focuses on grouping the words with a similar meaning and therefore removes also derivational suffixes. Stemming would remove the *-ing* in both cases for the *meeting* example and would not change *is* or *women* at all. Nevertheless stemming

provides for English text similar results than lemmatization, however the results of either methods depend heavily on the actual NLP tasks[48]. According to the most likely similar results for English, the lower complexity and the larger groups during this thesis just stemming is considered for generalizing tokens.

Stemming analyses single words with rule-based approaches to determine the stem. The group of methods is called stemmer. According to Schütze et al. Porter's algorithm is the most common stemmer for English[48]. Subsequently the NLTK implementation of Porter's algorithm is used during the conducted study. The following description is an overview of this stemmer based on the detailed definition from Porter[44]. It contains 5 steps. For each step rules are defined to transform the suffixes and determine the longest possible suffixes that fit the applied condition. In order to ensure a minimal length of the stem a condition based on a length measurement of transitions between vowel and consonant is defined. The first step removes plurals and past participle suffixes, while the steps two to four remove specific suffixes and step five removes some remaining characters, like *-e*. One example rule part of the second step is formalized as: $(m > 0)$ATIONAL $\rightarrow$ ATE. This means if at least one transition from vowel to consonant exits before ATIONAL, than it can be replaced with ATE. One example for this rule is RELATIONAL $\rightarrow$ RELATE.

3.3. Lexical approaches

After describing the commonly used preprocessing steps we can delve into the actual NLU features starting with lexical approaches. All three considered approaches are with the exception of calculating the word representation similar, therefore they are described jointly.

The TF-IDF model is used as a baseline due to its frequently usage in topic clustering/classification for many IR tasks. Furthermore, it does not measure any semantic information aside from the information entropy of each occurring word. This conceptual difference to all other models makes TF-IDF suitable to provide conclusions about the advantages or disadvantages of semantic models in general. This baseline model is accompanied by word2Vec and GloVe models on lexical level. Undoubtedly both models involve semantic knowledge within the word representation and use dense vectors. Similar to the TF-IDF approach both models are popular for many NLP task, especially ANN methods apply frequently Word2Vec due to the possibility to use backpropagation. Both approaches are considered because the influence of the global optimization used by GloVe and the local context window of word2Vec is not foreseeable. This word embeddings also build the fundament for the other two presented models, because those advanced models use basically the same lexical representation and expand it by taking compositional semantics into account.

As mentioned in Section 3.1 lexical methods do not use the word order of a text. According to this sentence and paragraph representation consider the word representations respectively sentence representations as unsorted list. Since no additional dependencies must be considered the main task for both steps is normalization. This maps the sets with different cardinalities to a fix-sized vector. Therefore, the models become invariant to text lengths. TF-IDF uses averaging, a particularly often used way to normalize data. This is

frequently used for word embeddings as well. Cer et al. provide an example for this on the STS task[8]. Consequently, averaging the word representation to calculate the sentence and paragraph representation is also used in this work. The resulting process pipeline for all lexical approaches is depicted in Figure 3.3.

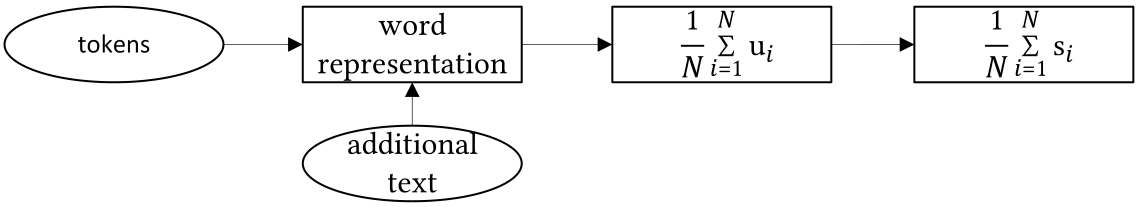


Figure 3.3.: Process pipeline for lexical approaches. The sentence and paragraph representations are calculated by averaging the word representation.

The following sections take a closer look into the single word representations, the reasons for averaging them and the implementation. Finally, the hyperparameters of each model are discussed.

3.3.1. Word representation

Naturally the word representation is the defining element of pure lexical approaches. The three considered models are default approaches. In consequence those representations have been introduced in Section 2.3. Those basic concepts are not altered, but in order to improve the representations aspects regarding the vocabulary and training are discussed in the following paragraphs.

Vocabulary TF-IDF uses a weighted one hot encoding for words. This requires a fix vocabulary size otherwise each word would be represented by a vector with a different length. This vocabulary can be built by analyzing the corpus. Including all words creates on the one hand the most accurate representation but on the other hand the representations get sparse. As mentioned data sparsity is an undesirable property for further analysis. Based on this trade-off the definition of the vocabulary is a crucial part of the word representation for TF-IDF. Apparently considering the most frequent words will decrease the dimensionality and overall the least amount of words are ignored. On the other side frequent words do not carry the same information value, therefore stop words are removed from the vocabulary, too.

The dimensionality of word embeddings is independent of the vocabulary size. Accordingly, no fixed size vocabulary is required. The trade-off between vocabulary size and vector dimensionality does not exist. Consequently, the removal of stop words or less frequent words is unnecessary. This allows adding new words to the vocabulary. The previous weights are simply appended and trained again. Assuming well-trained embeddings the additional words are not going to change the old embeddings significantly.

In order to avoid any adaption at all the old embeddings can even be kept fixed and only the new representations are trained. This highlights the most important aspect for word embeddings: training.

Training In opposition to the one-hot representation word embeddings are defined by their occurrence. Therefore, the corpus must contain a sufficient occurrence of each word to define the lexical semantic of it. A large corpus is more likely to fulfill this requirement, but it also contains word ambiguity. This trade-off between domain-specific training and general pretrained embeddings was described in Section 2.3. Improving the training without introducing more ambiguity has to append the corpus with domain-specific data. This work gathers additional data by identifying domain related Wikipedia articles based on the nouns of the corpus. The extraction utilizes the POS tagger of NLTK to identify those nouns and the Wikipedia Python library to gather articles of those identified.

3.3.2. Paragraph representation

The next processing step generates a representation for larger utterances. Pure lexical approaches normalize the word representations to generate length invariant features. As mentioned this normalization step calculates for all three approaches the average value defined as:

$$\vec{w_d} = \frac{1}{N} \sum_{i=1}^{N} \vec{u}_i \tag{3.1}$$

where N denotes the total number of words in the paragraph, and $\vec{u}_x$ the x-th word representation. This calculation creates especially for lexical semantic encoded in word embeddings two problems.

Contradiction Since a word embedding $\vec{u}$ with n dimensions is defined as a vector $\vec{u} \in [-1, 1]^n$ averaging word embeddings cancel out certain aspects of the overall meaning. One theoretical example are oxymorons, like *awful good*. By averaging contradicting words, the meaning of each word gets annihilated. Since oxymorons are used to emphasis certain aspects this would not be a desirable result. Due to the unspecific notion of word relations used by word embeddings an oxymoron is just an artificial example for two appropriate reverse vectors. Avoiding this behavior requires a more complex model which considers compositional semantics.

Weight Another aspect which is not accurately modeled by simple averaging word representations is the individual influence of words. Depending on the POS, word position and the occurrence the meaning of each word has to be weighted differently. For example, nouns and verbs contribute more to the overall meaning than adjectives. Partially the IDF factor applied in TF-IDF addresses this problem. Consequently, the averaging of word embeddings could be improved by including similar weights as well[35]. This extension is defined in Equation 3.2.

$$\vec{w}_d = \frac{1}{N} \sum_{i=1}^{N} IDF(i, D)\vec{u}_i \tag{3.2}$$

3.3.3. Implementation

Two lexical approaches (TF-IDF, GloVe) are implemented from scratch with Python and in the case of GloVe with TensorFlow. The Word2Vec representation is calculated by the Gensim[1] Python library. All paragraph representations are calculated with a trivial NumPy averaging function. Therefore, just the word level implementation is discussed briefly.

One-hot encoding The most frequent words are determined with a simple counter. This standard collection class counts the word occurrence in the whole text corpus, stored as a list in `tokens` and ranks them accordingly. The source code is defined in Listing 3.2 where `vectorSize` denotes the hyperparameter to determine the maximum vocabulary size.

```
1 import collections
2 wordCount = collections.Counter(tokens) # sort
3 wordCount = wordCount.most_common(vectorSize)# select
```

Listing 3.2: Usage of Counter to sort words based on the occurrence in `tokens`

On word level the one-hot encoding is not implemented as a vector, because of the extreme sparsity of such a vector. Instead a simple index for each word is saved which indicates the position of the one in the vector notation. This index is stored in a simple dictionary with `voc[key] = index`. The corresponding word string is used as a key and vice versa the word string is added with the index as key to allow resolving the one-hot encoding.

The IDF factor for each word is moved outside of the sum to reduce the total amount of calculations and consequently multiplied with TF (or paragraph representation).

GloVe The GloVe model is implemented with TensorFlow. Similar to the one-hot encoding a word is mapped to one index which refers to the actual embedding. The TensorFlow graph receives a matrix of indices for context words, center words and the corresponding co-occurrence counts as input. The placeholder defined in line 1 and 2 in Listing A.2 shows this for the context words and the co-occurrence count. The center words behave like context words on that account they are omitted. In line 3 the corresponding embeddings are generated and randomly initialized with a value between −1 and 1. Based on the index matrix a pointer to the related embedding is generated in line 4 for each matrix element.

```
1 cooccurrenceCount = tf.placeholder(tf.float32)
2 contextInput = tf.placeholder(tf.int32)
3 contextEmbeddings = tf.get_variable(shape=[vocabSize,
    vectorSize], initializer=tf.random_uniform[...])
```

[1] https://radimrehurek.com/gensim/

```
4 contextEmbedding = tf.nn.embedding_lookup([contextEmbeddings
    ], contextInput)
```

Listing 3.3: Simplified lookup of embeddings based on indices.

The cost function of GloVe as defined in Equation 2.10 is straight forward implemented as TensorFlow graph accessing the `contextEmbedding` pointer list. The minimum is found by a standard TensorFlow Adagrad optimizer. A shortened version of the source code is attached in Appendix A together with the Tensorboard visualization of it.

Word2Vec The calculation of Word2Vec embeddings using Gensim is conceptual the same process as extracting the Par2Vec embedding which is presented in Section 3.4. Therefore, it is not described in detail at this point.

3.3.4. Parameter

Each model uses several hyperparameters. Those parameters influence each model substantially. Therefore, they are summarized in this section and their influence is discussed.

TF-IDF The described TF-IDF implementation uses one hyperparameter: the vocabulary size. This parameter defines the size of each word vector. Consequently, the data sparsity and the number of considered words is determined by this parameter as previously discussed. Typical the vocabulary size for models ignoring compound words is set to a few thousand words. This number corresponds to the number of words a human needs to know to understand the text.

Word embeddings The presented word embeddings are more complex models than TF-IDF. They contain more hyperparameters. All embeddings define the following parameters.

- Vector size: The dimensionality of the embedding vector determines the number of weights. More weights in a network allow a better adaption of the network to the original probability distribution but might also cause overfitting the model to the training dataset. Commonly vectors with 100 to 500 dimensions are used[39].
- Context window length: The context window defines which words are included for the prediction task. Small windows encode more grammatical structure and larger windows influence embeddings to focus on the overall meaning[43].
- Training parameters: Additionally, typical training parameters like the learning rate, batch size and number of epochs are incorporated to each model. Those parameters influence the optimum search by setting the adaption rate of all optimized parameters, determining the subsamples to approximate the probability distribution and the repetition to find better minima for non-convex functions.

Additionally, each model has its own parameters to influence the cost function to reduce the influence of frequent words and improve the training.

GloVe The hyperparameter focus on the weight function of the co-occurrence counts for GloVe.

- Co-occurrence maximum x_{max}: Is the upper limit of the weight function. All occurrences above that limit are mapped to the maximum value of 1. This cutoff prevents disproportionate influence of high co-occurrence counts. Pennington et al. set $x_{max} = 100$[43].
- Scaling factor α: Below the maximum value the weight of the co-occurrences is determined by the scaling factor. Pennington et al. argue that empirical $\alpha = 0.75$ improves the embedding slightly compared to a linear scaling factor[43]. This scaling factor emphasis rare words.

Word2Vec The Word2Vec model provides two hyperparameters to improve the learning performance by adjusting the subsampling and negative sampling.

- Subsampling t: This hyperparameter used in Equation 2.9 determines how fast the drop probability for a sample converges to 1. This scales down the update frequency for common words. Mikolov et al. recommend to set t around 10^{-5}.
- Negative samples: The number of negative samples used to train the embeddings is the last hyperparameter. For most applications adapting 5 to 20 negative examples for each prediction step is sufficient[39]. The negative sample parameter does not exist for models using the hierarchical softmax to improve the performance.

3.4. Unsupervised, compositional approach: Par2Vec

Considering solely lexical semantics restricts the possible understanding captured by NLU features to the sum of the parts of a text. It fails to capture the additional semantic component denoted by the relation between the different words. Many methods address this disadvantage and try to capture compositional semantic as well. Consequently, these methods have to use the position of words to generate sentence and paragraph representations.

One model using the word order is the unsupervised Par2Vec model introduced by Le and Mikolov[30]. This method extends the language model of Word2Vec to additionally optimize a paragraph representation. Instead of just using the word embeddings to predict words for each paragraph a vector is added to the word embeddings and is subsequently also altered during the optimization. The Par2Vec models assumes with this additional vector for each paragraph that the accurate prediction of a word does not only depend on the corresponding context/center words, but also on the overall topic of this paragraph. Therefore, the vector distinguishes between paragraphs with different topics. Due to the simultaneous calculation both representations influence each other. Therefore, the word embeddings of this model differ from the simple lexical approaches.

Par2Vec considers the word order based on the context windows, but it does not use any syntactical structure other than unordered neighboring words. According to this property

it is controversial whether Par2Vec can be considered a sequential model. For the course of this thesis it is classified as a sequential model due to its usage of minimal word order. However the hierarchical model introduced in Section 3.1 distinguishes between sentence and paragraph representation is not fully applicable due to the minimal incorporated word order. The intermediate sentence representation can be skipped. Based on this lack of considered linguistic structure specific word relations are not considered. This structure leads to the process pipeline illustrated in Figure 3.4.

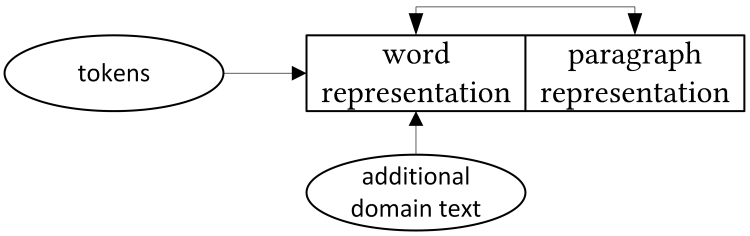


Figure 3.4.: Process pipeline for the Par2Vec model. Word and paragraph representation are simultaneously calculate by optimizing a language model.

In the following sections the Par2Vec algorithm is described in greater detail with a special focus on the used Gensim implementation and the parameters.

3.4.1. Paragraph representation

The paper by Le and Mikolov introduces two extensions to the Word2Vec approach. Each extension adds a paragraph representation to the two different prediction tasks of the Word2Vec model. The distributed memory approach adapts the Continuous Bag of Words model and the distributed BoW alters the skip-gram approach and is compatible to it.

Distributed memory The CBoW model uses the average / concatenation of all context words to predict the center word. This approach can be appended with a paragraph representation by simply adding a paragraph vector to the context vectors. It is illustrated in Figure 3.5. The additional vector is denoted by $\vec{w}_d$.

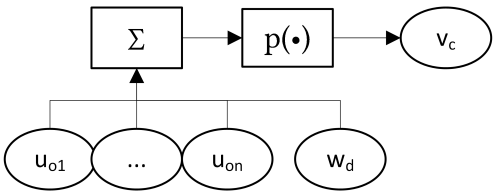


Figure 3.5.: Visualization of the paragraph vector (distributed memory) model. The context word vectors u_o and the paragraph vector w_d are combined to predict the center word v_c.

This vector alters the probability function of Continuous Bag of Words slightly. Accordingly, the probability function is defined as:

$$\begin{aligned} \vec{u}_{(o,d)} &= \frac{\vec{w}_d + \sum_{o\in} \vec{u}_o}{1 + |o|} \\ p(c|(o,d)) &= \frac{\exp(\vec{v_c}^T \vec{u}_{(o,d)})}{\sum_{\substack{w=1 \\ d=1}}^{V,D} \exp(\vec{v_c}^T \vec{u}_{(w,d)})} \end{aligned} \tag{3.3}$$

where d denotes the index of the paragraph vector $\vec{w}_d$ and D denotes the set of all initial training paragraph indices. Compared to Word2Vec CBoW the intermediate averaged representation of all context words $\vec{u}_{(o,d)}$ is appended by $\vec{w}_d$. As a result, the sum over all possible windows have to adapt the paragraph vectors as well. The prediction task is distributed between context words and paragraph vector.

New paragraph representations (not used during the initial training) are obtained with fix word representations as a result the training can be accelerated, because just the randomly initialized new paragraph vector must be adjusted to minimize the cost function.

The paragraph representation is held constant for all windows of one paragraph regarding the cost function. This allows a variable length of the actual paragraphs which helps to encode single sentences and whole text documents. Due to the optimization on multiple windows a paragraph vector encodes the topic of each paragraph. This topic tends to generalize more for larger text, because a single word prediction does not contribute as much as the same prediction for a smaller text. The second Par2Vec approach exhibits this as well.

Distributed BoW The second introduced approach appending word embeddings to paragraph embeddings is oriented on the skip-gram model. Skip-gram uses the center word to predict all context words. Derived from this model the distributed BoW representation incorporates just the paragraph vector to predict all words of one paragraph. The process is visualized in Figure 3.6.

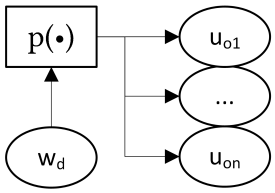


Figure 3.6.: Visualization of the paragraph vector (distributed BoW) model.

In difference to the distributed memory approach it is not necessary to calculated a intermediate representation out of multiple words. For each word the in Equation 3.4 defined probability has to be calculated.

$$p(o|d) = \frac{\exp(\vec{u_o}^T \vec{w}_d)}{\sum_{w=1}^{V} \exp(\vec{u_w}^T \vec{w}_d)} \tag{3.4}$$

The prediction solely based on paragraph vectors removes the dependency of the center word v_c for the probability. This cuts the word embedding parameters by half and consequently creates a simpler model. Additionally, the context word embeddings u_o are just used to compare the prediction to the actual result. Updating of those embeddings is unnecessary[30]. They can remain randomly initialized, if their only task is to identify the different words. The distributed BoW model only optimizes the paragraph vectors, therefore it is a far simpler model than the distributed memory approach. But the abandonment of center words also dispenses the word order. The distributed BoW approach does not use any compositional semantic. Strictly speaking just, the distributed memory approach is an unsupervised compositional method.

Le and Mikolov report for their evaluated tasks the combination of both models as consistently best model and the distributed memory approach alone as close to those results[30].

3.4.2. Implementation

Similar to the Word2Vec model the Par2Vec representation is implemented by the Gensim Python library. Following the usage of this library is described, which does not differ greatly between Word2Vec and Par2Vec. Gensim renames the Par2Vec model to Doc2Vec, which established it as a synonym for Par2Vec.

The Gensim approach is oriented on Scikit-learn classifier. One classifier object is created with all hyperparameters, trained and finally the classifier predicts the results. This basic process is shown with the actual function calls in Listing 3.4.2.

```
from gensim.models import Doc2Vec
model = Doc2Vec([...])
model.build_vocab(trainText)
model.train(trainText, total_examples=len(trainText), [...])
parRep = model.infer_vector(paragraph)
```

Both models (Word2Vec, Par2Vec) require a special input format of tagged words / paragraphs. This preprocessing step creates `trainText` as a list of `namedtuple`. For each element the words and a unique tag is required to train the embedding. The `inver_vector` function does not need this tag later on. Listing 3.4.2 shows this preprocessing with a simple index as tag, because the tag is not used further on.

```
from collections import namedtuple
gensimTextRows = namedtuple('tokens', 'tags')
trainText = []
for index,sent in enumerate(ticketSet):
    trainText.append(gensimTextRows([token for token in
       itertools.chain.from_iterable(sent)],[index]))
```

3.4.3. Parameter

Due to the close relation between the Par2Vec and the Word2Vec model they share all hyperparameter. The influence of each parameter to the paragraph representation is similar to the influence on word embeddings. Le and Mikolov emphasis that especially the window size parameter should be cross-validated since they report that the error rate varies up to 0.7% for the evaluated sentiment task.

3.5. Transfer-learning approach: BiLSTM

The last reviewed feature is the most elaborated model with respect to the syntactical structure of a text. Instead of using context windows similar to Par2Vec the presented model includes the full word order and represents thereby a typical sequential model on sentence level. As such it can enclose the linguistic structure of a sentence, like dependencies to gather compositional semantics. This approach extends the lexical model (word2Vec or GloVe) by replacing the word averaging on sentence level with a RNN which is trained to provide compositional semantics. Likewise a brief look at the process pipeline in Figure 3.7 shows this difference.

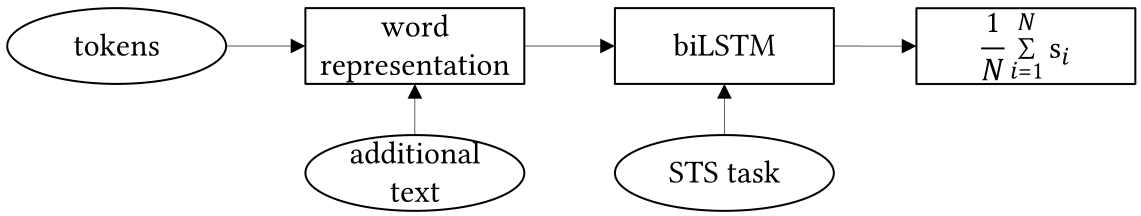


Figure 3.7.: Process pipeline for the BiLSTM model. The sentence representation is calculated with a BiLSTM network which is trained on the STS task based on word embeddings as word representations.

The sentence representation or to be more precise the ANN has to be trained to extract compositional semantics. This internal training task must include semantic knowledge. Through the training on such a task the parameters of the ANN learn to create an accurate sentence representation which captures the sentence's meaning. Afterwards the trained model can generate a representation for all kind of sentences. The learned knowledge from a specific task is transferred. In opposition to the Par2Vec and word embedding approaches this training data is not provided by large unlabeled corpora based on language models. Hence the model is supervised. This transfer-learning approximately follows the same approach commonly used in object recognition as Conneau et al. point out[14].

The two key factors of this transfer-learning approach are the model and the initial task. Especially the initial task must rely on semantic knowledge. One example for such a task is the STS task. It provides due to the SemEval workshop sufficient labeled data and benchmark models. Furthermore, the task of classifying sentences due to overall

similarity resembles the later contemplated problem of detecting duplicates. Based on the training task the BiLSTM model proposed by Tai et al. for STS is considered as classifier during this work[53]. This BiLSTM model shows on the STS benchmark a reasonably good performance[8]. Supplementary LSTM approaches are commonly used for many NLP tasks due to the variable input length. However, the focus of this work lays on the principal application of transfer-learning, therefore evaluating a great number of different models is out of scope for this work.

For completeness the two recently published approaches *InferSent*[14] and *Universal Sentence Encoder*[9], which both propose a general sentence representation obtained by transfer-learning tasks, should be mentioned as well. Neither of those papers were reviewed in detail during this work.

Consistently to the review of the aforementioned lexical approaches and Par2Vec the following section starts with a description of the basic algorithm, followed by implementation details and a summary of all hyperparameters of the BiLSTM approach.

3.5.1. Sentence representation

The sentence representation is generated by training the weights and biases of a BiLSTM model to predict semantic similarity between two sentences. The intermediate representation of this training task constitutes a NLU feature for each sentence, which includes compositional semantics and can be applied to tasks with less or none training data.

The process of the whole training task as proposed by Tai et al. is illustrated in Figure 3.8[53]. The intermediate representation of two sentences is compared with a simple feedforward network with one hidden layer. The resulting probability vector is compared to the actual label to calculate the loss and adapt the parameter of the feedforward model and the BiLSTM. Both components are in detail introduced in the following paragraph.

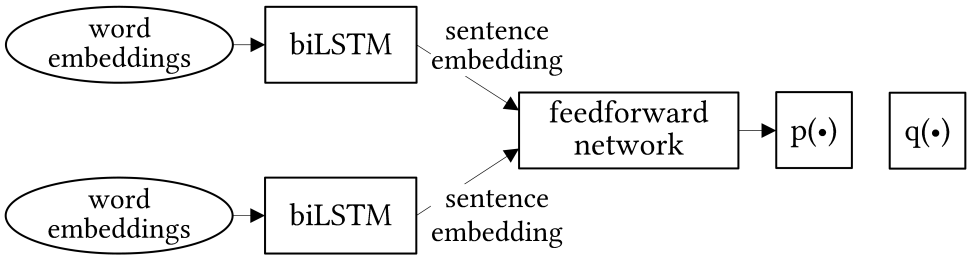


Figure 3.8.: Network architecture of all components for the STS training task.

BiLSTM The LSTM cell was introduced in Section 2.2. The used exemplary definition of all gates (Equation 2.6) matches the definition of a LSTM cell used in this model.

The input sequence of word embeddings is considered in forward direction and backward direction to avoid influencing the hidden state excessively by the end or beginning of the sequence. Each direction is fed into one single layer LSTM cell. This constitutes the BiLSTM. The last hidden state of both cells are concatenated and form the final output of the BiLSTM component.

Feedforward network The feedforward network compares two sentences regarding the similarity of the semantic annotated representations. To avoid training the feedforward network to append the semantic sentence representation, the inputs of this component are two simple difference measurements instead of the actual vectors. The semantic representation should be exclusively generated by the BiLSTM component because the feedforward network is exchanged later on. Consequently, all linguistic knowledge incorporated into it would be lost. In theory using a difference value as input separates the semantic representation and the similarity classification. Due to the difficult interpretation of ANN transformations this cannot be proven trivially. Therefore the sentence vectors calculate by the BiLSTM are compared regarding the absolute difference in each dimension and the angular difference. Both measurements are defined as:

$$\begin{aligned} \vec{h}_+ &= |\vec{h}_L - \vec{h}_R| \\ \vec{h}_\times &= \vec{h}_L \odot \vec{h}_R \end{aligned} \tag{3.5}$$

where $\vec{h}_L$ respectively $\vec{h}_R$ denote the sentence representations and $\odot$ denotes elementwise multiplication. According to Tai et al. a combination of both comparisons is empirical proven to be superior compared to either of them alone[53]. Thereby the hidden layer combines $\vec{h}_\times$ and $\vec{h}_+$ by adding both transformed vectors as formalized in Equation 3.6.

$$\vec{h}_s = \sigma(W_{\vec{h}_\times}\vec{h}_\times + W_{\vec{h}_+}\vec{h}_+ + b_{\vec{h}_s}) \tag{3.6}$$

Finally the output layer calculates the probabilities for each similarity class of the STS task by applying a softmax nonlinearity, therefore the result vector is defined as $\hat{p} in [0,1]^6$ and the output layer is:

$$\hat{p} = \text{softmax}(W_{\vec{h}_s}\vec{h}_s + b_{\hat{y}}) \tag{3.7}$$

Logically the predicted class is defined as $\hat{y} = \begin{bmatrix} 0 & 1 & \dots & 5 \end{bmatrix} \hat{p}$. By this definition the probabilities are used as weights to sum the classes and average them. This allows like the average label a continuous result instead of a strict classification into the six classes.

The loss compares the prediction $\hat{p}$ with a probability distribution q calculated based on the floating-point labels. The label spreads the probability to the two nearest classes based on the distance to either of those classes.

The actual loss is calculated between those two probabilities as cross entropy and optimized by simple gradient descent. Due to the many parameters of the model it tends to overfit, therefore, a L2 regularization term is added to the cost function. This term penalizes large weight values by adding the sum of quadratic weights. Additionally the BiLSTM applies dropout to introduce noise in order to avoid overfitting. Tai et al. report no significant improvement for the STS task, but due to the focus on transfer-learning and the reported improvements for the sentimental task dropout regularization is applied in this model.

Transfer-Learning The training task applies backpropagation to optimize the feedforward network and the BiLSTM. Afterwards only the BiLSTM is used to generate a sentence representation which is not used to classify two sentences but as general NLU feature.

Due to the training task the representation has been trained to include semantics of each sentence. The model learned in an abstract way to identify linguistic concepts between words like dependencies to infer the compositional semantic of a sentence. Ideally this knowledge can be transferred to identify the semantic of all sentences without any additional adaption. This best-case scenario is difficult to achieve because the training task introduces a bias based on the task and used training data. For example the STS tasks contains colloquial sentences which are not directly comparable to technical language. The slight change of the actual task alters the possible required features, too. Classifying a sentence as duplicates and non-duplicates might require completely different features than sorting sentences into five similarity classes. Due to the opaque, difficulty to interpret features generated by ANNs this cannot be theoretically excluded.

One method of solving those problems is additional training on the actual task. Similar to word embeddings the BiLSTM weights can be adjusted in the context of another task as well. The training on the STS task provides the initial weights which are adapted by additional task specific training. Due to the initialization this training requires less data to find a fitting representation. Due to the limited time and quality of training data this possibility is not explored during this thesis, but most likely would improve the obtained results for the transfer-learning model without additional adaption to the specific task.

3.5.2. Implementation

The BiLSTM approach is implemented like GloVe with TensorFlow. The BiLSTM component is assembled out of standard TensorFlow classes for LSTMs, calculating dropouts and passing a bidirectional sequence to RNN cells. Those abstractions allow it to implement it within a few lines, illustrated in Listing 3.4.

```
1 lstmFw = [...].DropoutWrapper([...].BasicLSTMCell(lstmSize),
      output_keep_prob=keepProb)
2 state = [...].bidirectional_dynamic_rnn(lstmFw,lstmBw,sentA,
      [...])
3 tensor = tf.concat([state[0].h, state[1].h],axis=1)
```

Listing 3.4: BiLSTM component

For the feedforward network component the defined equations 3.5 - 3.7 are straight forward implemented. The TensorFlow graph for the whole model is attached in Appendix A.

The cross-entropy losses are optimized with a standard TensorFlow gradient descent algorithms. The required probability for the actual label is calculated by Listing 3.5. The two equal comparisons mask the two nearest classes to the actual label and split the floating-point part between those classes.

```
1 rClass = tf.range(0,5)
2 floorLabel = tf.floor(label)
3 labelProb = tf.equal(floorLabel - tf.transpose(rClass),-1.0)
     * (label - floorLabel)) + tf.equal(floorLabel - tf.
    transpose(rClass),0.0) * (floorLabel - label + 1))
```

Listing 3.5: Calculation of the label class probabilities based on the floating point label.

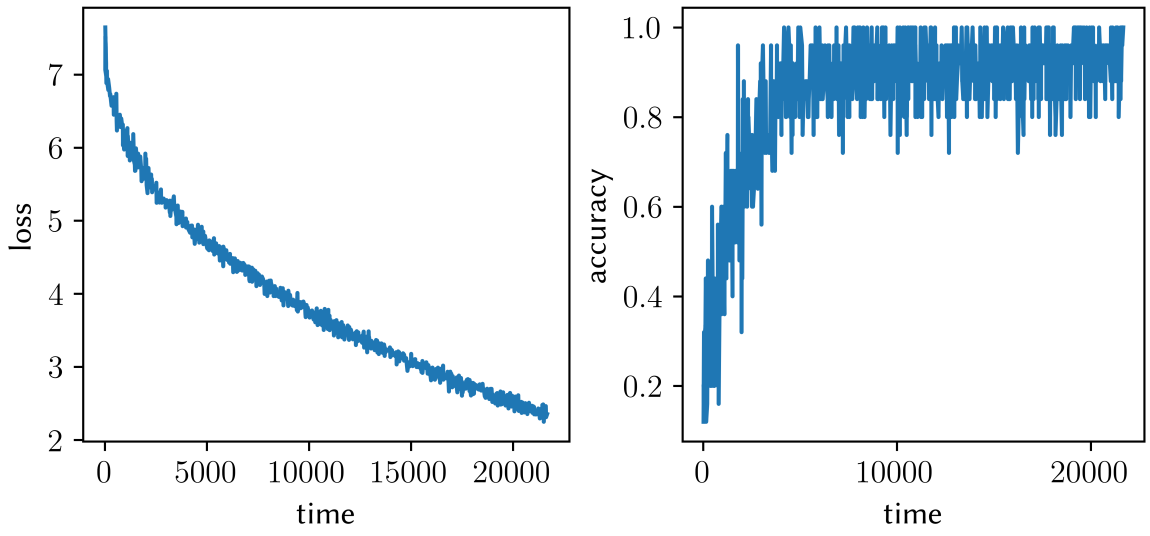
3.5.3. Training

One special aspect of the BiLSTM model is the prior training on STS data. The model is trained on 8,628 sentence pairs of the *STS Benchmark* dataset[8]. Initially the dataset is split into a train part with 7,249 pairs and a test set with 1,379 pairs. This split is used to analyze the model on the STS task. Accordingly, the following reported results apply this split. Since the final evaluation task does not use the STS test dataset the whole dataset is used to train the sentence representation later on.

During training the cost function is gradually optimized and must decrease over time. Correlated to the decreasing cost the prediction accuracy should increase. The accuracy is calculated as:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1 & \text{if } \lfloor \hat{y}_i + 0.5 \rfloor = \lfloor y_i + 0.5 \rfloor \\ 0 & \text{otherwise}\end{cases} \tag{3.8}$$

where $\hat{y}_i$ denotes the the i-th prediction, y_i the corresponding label and N the overall number of examples. The graphs in Figure 3.9 show a training run. The cost and the accuracy show that the model is able to learn comparing the two sentences. Especially the accuracy reaches almost 100% which means that the model predicts most training examples correct. In fact this model achieves on the training dataset a Pearson's r of 0.98.



(a) cost curve

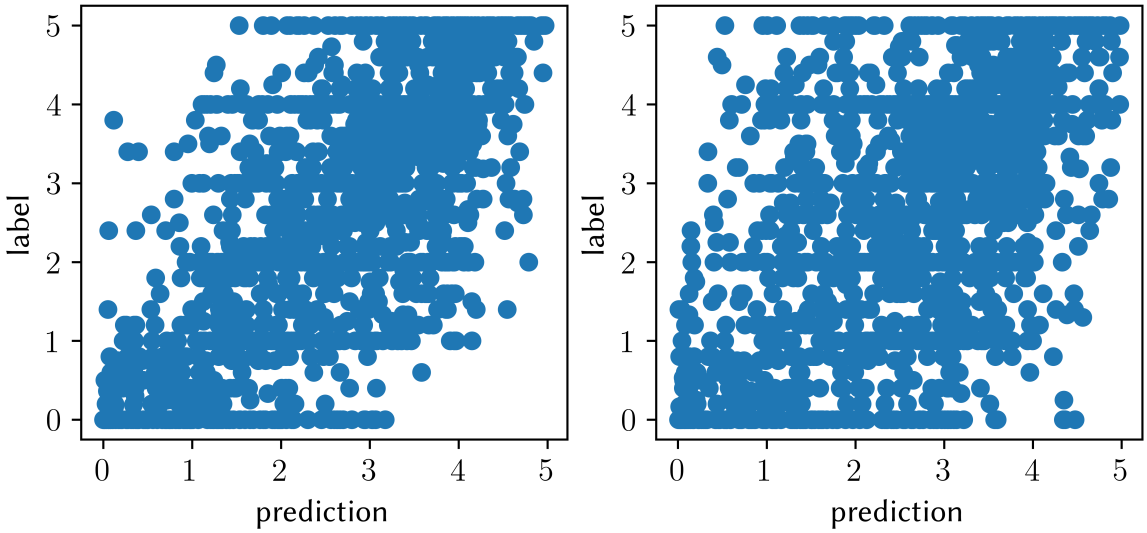
(b) accuracy curve

Figure 3.9.: The cost (a) and accuracy (b) curve for one training run of the BiLSTM model.

The relevant performance on the test dataset is at $r = 0.677$, which is in the same range of the reported $r = 0.711$ for this model[8]. The slight difference is most likely caused by the different preprocessing steps and epochs. For example, the presented model removes all number tokens without replacing them with a special number token. The predictions and labels are illustrated in Figure 3.10.

Another important training aspect is the used vocabulary. The actual task most likely uses a different vocabulary than the STS dataset. Consequently, during training this different vocabulary must be used. This deteriorates the performance of the STS task

as illustrated in Figure 3.10. The example uses domain specific vocabulary of the Bosch dataset which will be introduced in Section 5.1. The discrepancy of both vocabularies is quite large, therefore, the correlation drops to $r = 0.468$. This illustrates the challenge for transfer-learning. The BiLSTM model has to learn linguistic rules to extract compositional semantics and apply those rules to a possible completely different domain. Nevertheless the model shows that it is still able to predict STS classes and thereby learn some linguistic rules with domain specific vocabulary.



(a) general vocabulary $r = 0.677$ (b) domain vocabulary $r = 0.510$

Figure 3.10.: Predictions of the BiLSTM model compared with actual labels. The results using a general vocabulary is shown in (a) and results with filtered domain specific vocabulary is illustrated in (b).

3.5.4. Parameter

The number of hyperparameter for such a complex model is higher than the number of hyperparameter of aforementioned models, but Tai et al. provide considerable information and empirical results for the STS task about most hyperparameter. However, some intuition about the parameters is useful, too. The following list summarizes those parameters.

- Word embeddings: The input of the sentence representation step is of course not directly a hyperparameter of the BiLSTM model, but influences the overall result. The potential hyperparameter are listed at the lexical approaches. Tai et al. used a 300 dimensional pretrained GloVe vector[53]. They do not mention Word2Vec embeddings and two authors contributed to both papers.

- Maximum sentence length: The maximum sentence length is required to optimize the LSTM, because infinite backpropagation would not be possible. Logically a larger value increases the memory requirements, because shorter sentences must be padded. On the other hand, cuts a small value many sentences off and information

gets lost. This parameter has to be selected based on the expected sentence length. A limit of 50 token covers almost all sentences and does not waste too much memory. Tai et al. do not report this parameter[53]. For the STS task it is most likely that the maximum sentence length of the used STS dataset was chosen.

- BiLSTM output size: This hyperparameter determines the size of the sentence representation and thereby the number of weights within the LSTM unit. More parameters allow a better approximation of the actual probability function but requires more training data and increases the possibility to overfit.
- Hidden layer of feedforward network: The hidden layer size defines the complexity of the comparison between sentence representations. A larger model most likely shifts the analysis of linguistic rules into the feedforward network. Subsequently the BiLSTM component would produce worse sentence representations. But a small network might not be able to classify the similarity accurately. Tai et al. used a hidden layer of size 50[53].
- Regularization strength λ: The regularization add an incentive to train less complex models with small weights. The hyperparameter λ controls the weight of this additional term. Tai et al. apply a per-minibatch L2 regularization strength of $\lambda = 10^{-4}$ to avoid overfitting[53].
- Keep probability for dropout: Additionally, dropout reduces overfitting as well. The keep probability determines how many weights are removed during training. For the sentiment analysis Tai et al. reported a dropout probability of 0.5[53]. Typical this probability is in the range of 0.5 to 0.8[52]
- Training parameter: Of course, at last this approach uses the typical learning parameter learning rate, batch size, epochs, too. The influence does not change compared to other models. The BiLSTM model uses a learning rate of 0.05 and a minibatch size of 25[53]. The epoch iterations are not reported.

4. Information Retrieval

The previously introduced NLU features quantify the meaning of texts. Those text representations can be utilized for many analysis tasks requiring semantic knowledge. This chapter introduces Information Retrieval as such an analysis task. The NLU features allow to find information based on semantic knowledge of each document instead of simple occurrences like the typical vector space model (TF-IDF feature).

In general IR finds documents in a larger collection of documents which satisfy information needs[48]. Accordingly the main objective of all IR systems is grouping the collection into subsets of documents which satisfy a specific information need and optionally ranking these documents. As briefly mentioned in Section 1.2 the groups are either created in a supervised or an unsupervised fashion respectively through classification or clustering. Both approaches depend on measuring differences between documents or classes. Classification uses NLU features as input of a classifier. The cost function of this classifier measures the differences between the currently predicted class and the class label by comparing the probability distributions. The optimization step reduces this difference to a minimum. On the other side clustering measures the differences between entities of the document collection to group and rank them. For both steps a distance or similarity measurement is an important part. Due to the nature of the duplicate detection the similarity between documents is more important for this work. It is discussed in Section 4.1.

After establishing distance measurements for all NLU features the duplicate detection as a specific retrieval task is introduced in Section 4.2. This task finds based on one document other documents of the collection which describe the same real entity. Duplicates create inconsistency and redundancy in structured and unstructured datasets. Redundancy wastes memory capabilities and influences the efficiency of other tasks performed on the datasets, because of the additional unnecessary documents. The inconsistency which is the result of scattering information about the same entity into multiple documents influences aside from the efficiency also the effectivity. If information is spread over several documents some information might not be accessible to users. Therefore, the duplicate detection is an important task especially for large collections and is frequently applied in many contexts with different characteristics. Duplicate record detection filters and merges the records of structured databases, for unstructured data the duplicate detection filters the results of web crawling and as typical IR task it retrieves closely related documents to collect information of one entity from multiple documents. From an IR perspective the recognition of duplicates groups very similar documents together without labeling those groups. Accordingly, it resembles a typical clustering task without ranking the results. Therefore the detection of duplicates is established as showcase evaluation task for IR and consequently used to evaluate the NLU features in Chapter 5.

4.1. Document similarity

One particularly important aspect for all IR tasks is the notion of similarity between entities. The similarity or distance between entities allows the comparison between them and is the foundation of clustering or ranking algorithms. Many ML tasks aside from the IR field use those measurements as well. Hence many default similarity/ distance measurements exist. Especially the cosine similarity is frequently used for many text mining tasks[36]. Therefore, it is going to be used during this work as well.

Cosine similarity The cosine similarity is defined as:

$$d_{\cos} = \cos(\Theta) = \frac{\vec{h}_L \vec{h}_R}{\left\|\vec{h}_L\right\| \left\|\vec{h}_R\right\|} \tag{4.1}$$

where $\vec{h}_L$ respectively $\vec{h}_R$ denote the feature vectors and $\|\cdot\|$ denotes the Euclidean norm of each vector[]. Cosine similarity measures the cosine of the angle Θ between both vectors. Due to the normalization it is independent from the vector length.

Relation to NLU features The cosine similarity is frequently utilized in combination with TF-IDF[48]. Based on this empirical evidence the cosine similarity is consequently suitable for comparing TF-IDF vectors. In contrast word embeddings are designed to fit the cosine similarity. The cost function of word embeddings as defined in Equation 2.7 maximizes the similarity between context and center words by comparing both representations with the dot-product. The dot-product measures angular distance as well. Accordingly, the dot-product used for the optimization of language models and cosine similarity are closely related. Lexical approaches, Par2Vec and the BiLSTM features use those language models as basis. Therefore, the cosine similarity is most likely suitable as similarity measurement for those models as well.

Elementwise similarity Aside from comparing two vectors with a single scalar as result the representations can be compared elementwise and afterwards the output vector of this comparison can be used to classify the tuple. This method is just applicable for classification task which classify the relation between two feature vectors like duplicate detection. This comparison is frequently used for ANN classifier. The STS classification of the BiLSTM feature provides an example of such a case. The model compares both sentences as defined in Equation 3.5 with elementwise multiplication and the absolute differences between both vectors. The classifier learns to combine the dimensions to optimize the class prediction. Hence the elementwise multiplication measures angular distances and the absolute difference the length similar to the city-block distance.

4.2. Duplicate detection

The detection of duplicates removes redundancies and resolves inconsistencies by retrieving documents about the same entity. Therefore it is a frequent task in the IR field.

Commonly the duplicate recognition is restricted to identifying exact copies. Consequently checksums to detect copies of whole documents and fingerprints (selection of n-grams) or simple TF-IDF features are combined with a threshold classifier to detect partial copies are commonly used[15]. The detection of duplicates based on NLU features goes a step beyond those methods by exploiting semantic features. Logically the focus of the duplicate detection based on NLU features moves from recognizing exact copies to the recognition of same meanings.

Definition Duplicates are documents which describe the same entity. Therefore, a relation *isDuplicate* between two documents can be defined. This *isDuplicate* relation indicates whether the compared documents are duplicates or not. It is symmetric and transitive. This perspective illustrates the detection of duplicates as a binary classification problem.

From an Information Retrieval view the detection of duplicates resembles a search where one document describes the information need and all retrieved documents are duplicates of this document. This description corresponds to the general definition of an IR task.

NLU feature evaluation The definition points out that the duplicate detection represents a typical IR task. But in opposition to other retrieval task like web search it has a few properties which simplify the recognition. Those points are:

- The information need is formulated by the representation of one document. Consequently the representations of all possible information needs are equal to the representations of the document collection. This simplifies the IR task by eliminating possible systematic differences between the representations of the information need and the document collection. Typical query-based systems might be influenced by these different formulations.
- The key element for the duplicate detection is the comparison between two documents. Therefore the emphasis of this task is clearly on the NLU features and the similarity measurement. Subtasks like sorting or elaborated clustering is unnecessary.
- The differentiation between duplicates and non-duplicates is more objective than documents grouped based on satisfying an information need. Subsequently the labeled data is more reliable.

Due to those properties the recognition of duplicates is considered for evaluating the NLU features in Chapter 5. For this purpose, two classifier are described in the following sections.

4.2.1. Thresholding

The thresholding classifies the document tuple with a simple threshold and a similarity measurement. This approach is commonly applied for the duplicate detection with TF-IDF[15].

As first step the recognition of duplicates must compare the two considered documents to determine the overlap in meaning of both documents. This is handled by similarity measurements like the cosine similarity. The result of this comparison is a single scalar $d_{\cos}$ which quantifies the similarity between both documents.

The *isDuplicate* relation is true if both documents are like a certain level. Intuitively the tuples should be classified as duplicates if and only if the similarity values is greater than a certain threshold T. Mathematically formalized:

$$C_T = \begin{cases} 1 & d_{\cos} > T \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where T denotes the threshold. The output of 1 classifies the document tuple as duplicates.

This simple model is comprehensible. Therefore the influence of the NLU features is more easy to understand. Additionally, is the only hyperparameter the threshold T which can be set in a heuristic way by analyzing just a few examples. Due to those reasons the thresholding was selected as primary classifier for the evaluation.

4.2.2. Feedforward network

Aside from applying a simple threshold in combination with cosine similarity a more advanced classifier should be introduces as well. The presented feedforward network is oriented on the STS training task of the BiLSTM feature. Essentially this six-class classifier is simply reduced to a binary classifier.

Instead of a classical similarity measurement the comparison is done elementwise by the aforementioned elementwise similarities defined in Equation 3.5 to be able to emphasis specific dimensions within the hidden layer. The hidden layer is not changed either. But the output layer is reduced to just two dimensions. One element representing the probability for a duplicate prediction and the other element the probability for non-duplicates. Therefore, the prediction is defined by which probability is more likely. This is formalized in Equation 4.3.

$$\begin{aligned} \hat{p} &= \text{softmax}(W_{\vec{h}_s} \vec{h}_s + \vec{b}_{\hat{y}}) \\ C_F &= \arg\max(\hat{p}) \end{aligned} \tag{4.3}$$

The $\arg\max(\hat{p})$ function selects the index of the maximal dimension value.

The feedforward network requires training data and introduces multiple additional hyperparameter like the hidden layer size. The hyperparameter are listed at the BiLSTM feature (Section 3.5). The advantage of this ANN approach is the additional training possibility to adapt the NLU features as mentioned for the BiLSTM feature.

5. Evaluation

In order to compare the performance of all discussed features based on empirical data this chapter evaluates them in the context of the in Chapter 4 introduced IR duplicate detection. For this purpose two datasets were gathered and are presented in Section 5.1. In addition to the data characteristics the used evaluation metric is introduced in Section 5.2. The last part (Section 5.3) of this chapter discusses the evaluation results of all features.

5.1. Datasets

The different methods are evaluated on two datasets with diverging properties. Both datasets are gathered as part of this thesis. The Bosch bug report dataset is a real-world collection of bug reports. On the other side several Wikipedia summaries from different languages are gathered as artificial evaluation dataset. An overview of both datasets can be found in Table 5.1. The following sections describe those properties in detail combined with data preparation steps.

5.1.1. Bosch bug report dataset

Typical software projects contain many bugs. Therefore, applications are extensively tested by multiple stakeholders. The result of those tests are reports, which describe the error in natural language. Large projects gather thousands of those bug reports. The unstructured description and the large amount of bug reports creates a typical IR task. Particularly detecting duplicates provides a tangible improvement, because it reduces the amount of bug reports which have to be reviewed by developers.

The used collection of bug reports for the evaluation is provided by Robert Bosch Engineering and Business Solutions Private Limited. These bug reports are collected from

Property	**Bosch bug reports**	**Wikipedia summaries**
size	13,095 reports	200 summaries
• duplicate tuple	780	110
• non-duplicate tuple	3,900	550
data origin	real	artificial
structure	several paragraphs with headings, bullet points, etc.	one paragraph with no structural elements
language	keywords, English sentences	plain English sentences
vocabulary	specific domain	no specific domain

Table 5.1.: Overview of the most important dataset properties.

a car multimedia project. But due to the nature of those bug reports the dataset cannot made publicly available. It contains precise descriptions of sensitive data about Bosch software. Additionally, are some reports created by customers, whom have not agreed on publishing those reports. Due to this high confidentiality the shown examples within this thesis are not original reports either. The actual data is rephrased in a way to preserve the characteristics of those examples. These key characteristics of the used dataset are presented in the following section.

5.1.1.1. Data characteristics

The dataset contains 13,095 bug reports of which 1,515 reports are marked as duplicate of at least one other ticket of the dataset. Each report contains a short one-sentence summary, a longer description and a list of duplicate tickets. For the evaluation the NLU features are extracted solely from the description. Additionally, are the duplicate lists used to generate labeled list with two tickets and a Boolean value to identify those tickets either as duplicate tickets or not. Overall the 1,515 duplicate reports are transformed into 780 duplicate tuples. The small difference between tuple and duplicate number shows that a few reports have more than one duplicate. The remaining tuples are considered as non-duplicates. The tuples are subsampled to improve the ration between duplicates and non-duplicates. This is described as part of the data preparation, after taking a closer look into the properties of the actual descriptions.

Listing 5.1 and Listing 5.2 are two example descriptions of the same error. Primarily the different structure and different language stand out for those examples. Regarding the structure and used language the overall dataset is heterogenic as well.

```
* Sep.22 report Issue # 62
h4. Test Environment:
* Device Partnumber: XXXXXXXXXXXX
* Test-type (Bench/ In-car/ [...] / Automated): Bench
h4. Media Device CD (If used) :
* Mode (R/RW) :
h4. Action:
1. Destination -> Enter POI or Address
2. Enter any character -> Tap "Show All"
3. Tap next page arrow
h4. Observation: A redundant screen flashes a while.
```

Listing 5.1: Bug report with template and bullet points.

```
Enter a destination address and change into Show All view. Go to the next page.
Observation:
A different view is shortly visible, before the actual view is presented.
Expectation: Direct switch between the pages of Show All view.
```

Listing 5.2: Bug report in plain English with minimal headings.

Text structure The descriptions of bug reports are structure by several paragraphs with a heading for each paragraph. The paragraph either describes the sentence in plain English, uses bullet points or numbered lists. In some cases those structural elements are marked with annotations, like `h4.` for headings and `*` for bullet points. Listing 5.1 shows an example which uses annotations, in opposition to Listing 5.2, which does not use any additional annotations. In general, no annotation guidelines are used consistently.

The description of a bug covers naturally preconditions, a sequence of actions, an observation and the expected behavior. Based on these recurring elements of bug reports many of the reports uses templates. Unfortunately, no unified template can be identified within the dataset. Additionally, are parts of the template either removed or not filled in. Therefore parts like `h4. Action:` can be found in over 8,800 reports, but the term `h4. Notes (if any):`, which appears to be part of the same original template just occurs in 5,800 reports.

Overall the structure does not allow the extraction of more specific parts of the description like observations neither is the removal of the template without using a complex parser possible.

Language Aside from the difficult text structure the used language is not consistent either. Due to the commonly used template some points are answered using just keywords other parts are filled out with several sentences, but continuous text above the length of a few sentences is not used at all. As consequence of this full stops are seldom used to terminate a sentence. For most cases new lines are used to separate different parts.

Bug reports utilize domain specific vocabulary. It contains many technical terms and specific named-entities for elements of the software, like the `Show All view` in Listing 5.2. Based on the informal character of bug reports spelling errors cannot be precluded, too.

5.1.1.2. Data preparation

The original dataset is exported from an issue tracking software in CSV format. It contains 15,769 bug reports of which 4,448 reports are marked as duplicate. Those reports must be transformed into labeled pairs for the duplicate classification. Additionally, the original dataset contains cloned reports (one-to-one copies of other reports) and not all duplicates satisfy the transitivity property, therefore the corresponding reports are filtered out. The mentioned annotations are removed, too.

Removal of cloned tickets The raw dataset contained several exact copies of reports. Those reports are arbitrary easy to classify as duplicates and would skew the actual result, therefore those reports are removed by a binary comparison of the description text. 1,056 reports are affected of this removal. Additionally 991 reports are marked with `clone –` at the beginning of the summary. A few of those reports are slightly altered and consequently not binary equal to other tickets and not removed by binary comparison. In favor of avoiding any cloned data, all reports marked like this are removed as well. Overall 1,268 copies are removed.

Properties of *isDuplicate* relations The in Section 4.2 described properties of the *isDuplicate* relation can be used to validate labeled data. If two reports are duplicates, they have to be symmetric and transitive. The transitivity property creates distinct groups. All reports of those group have to be marked as duplicates of the same reports. If those criteria are not full filled at least one report of the group is not correct labeled. Similar to the removal of cloned tickets all tickets of a group not satisfying both criteria are removed. This leads to the removal of 1,406 reports.

Subsampling Adding one duplicate report into a dataset of n reports logically adds not only a duplicate tuple, but $n-1$ non-duplicate tuple, too. Logically all duplicate datasets are skew. For the evaluation and potential training unbalanced classes introduce a bias towards the more frequent class[]. The problem is handled by subsampling the non-duplicate tuple. Instead of all possible non-duplicate tuple just $5 * 780$ tuple are selected.

Annotations The mentioned annotations used by some reports add noise, therefore the most commonly used annotations are removed from the corpus. In Listing 5.3 the applied regular expression to replace the annotation is presented.

```
1 description = re.sub('h4..*?(:|\n)','',description)
2 description = re.sub('\*.*?(:|\n)','',description)
3 description = re.sub('{color.*}','',description)
```

Listing 5.3: Python expressions to remove marked headings, bullet points and color markup.

5.1.2. Wikipedia dataset

The Wikipedia dataset contains the summaries of different articles. The summary or lead section of a Wikipedia article is the text before the table of contents. This section is typical a few sentences long and summarizes the most important content of the following article. The summaries are gathered from the English, German, France and Spanish Wikipedia. All summaries are translated into English using Google Translate (Seq2Seq LSTM) with exception of the English texts. This creates several summaries about the same topic which are consequently labeled as duplicates. In order to avoid large differences the text size differs at most by 15% between the duplicates and is also manually filtered. The characteristics of the resulting summaries are described in the following section.

5.1.2.1. Data characteristics

The Wikipedia dataset contains just 200 summaries of which each summary is a duplicate. Therefore, it contains 110 duplicate tuples and 550 randomly selected non-duplicate tuples. The dataset is way smaller than the bug reports but provides more reliable *isDuplicate* labels and is more homogenous in language and text structure. Overall the data is less noisy. In Listing 5.4 and Listing 5.5 two shorted examples of the summaries are illustrated.

```
1 Gamma–ray bursts (often abbreviated GRB) are energy bursts of very high power in the
      Universe, which emit large amounts of electromagnetic radiation.[...]
2 The gamma–ray burst was first observed on 2 July 1967[...]
3 Gamma radiation in the narrower nuclear–physical sense does not involve gamma–ray
      bursts.
```

Listing 5.4: Shorted German summary of the gamma-ray burst article.

```
1 A gamma–ray burst or gamma–ray burst (sometimes translated as gamma–ray
      explosion) is in astronomy a burst of gamma–photons which appears randomly in
      the sky.[...]
2 The dominant theory is that the gamma–ray surge is generated[...]
3 The gamma–ray bursts are accidentally discovered in 1967.
```

Listing 5.5: Shorted France summary of the gamma-ray burst article.

The examples show that most summaries contain similar information. Nevertheless, all texts are most likely written by different authors and consequently differ in formulations and focus of the information. For example, the German gamma-ray burst summary focuses on the history aspects compared to the France summary which focuses on the physical explanation. The text properties

- Text structure: All summaries contain just one single paragraph. The paragraph does not use any heading or other structural elements like new lines.
- Language: The summaries use plain English sentences. All sentences use a full stop to terminate a sentence. The summaries are selected randomly form all Wikipedia articles. Therefore, no domain specific vocabulary is utilized and spelling errors are seldom.

5.2. Evaluation metric

The detection of duplicates as a binary classification problem can be evaluated by the Receiver Operating Characteristic (ROC) curve. This metric is commonly applied to evaluate binary classifier. The basics of the ROC are briefly described in the following section. This default model is complemented with a view into the distribution of similarity between duplicates and non-duplicates.

5.2.1. Receiver Operating Characteristic curve

The binary classification can be divided into four possible situations for each tuple:

- True positive (TP): The text tuple is labeled as duplicate and classified as such.
- False positive (FP): It is not labeled as duplicate but classified as one.
- True negative (TN): The tuple is not labeled nor classified as duplicate.

- False negative (FN): It is labeled as duplicate but classified as non-duplicate.

Based on these cases the recall or true positive rate (Equation 5.1) and the false positive rate (Equation 5.2) are defined. The rates depend on the number of tuples assigned to each situation. Consequently TP, FN and so on represent in each case one integer number. The true positive rate compares the correctly predicted duplicates with the amount of labeled duplicates. The best case are values close to one. In opposition to this the false positive rate compares all wrongly as duplicates predicted tuples with all non-duplicate labeled tuples. The FPR should be close to zero.

$$TPR = \frac{TP}{TP + FN} \tag{5.1}$$

$$FPR = \frac{FP}{FP + TN} \tag{5.2}$$

Both rates point out a different aspect of the classifier but influence each other. A good recall value most likely leads also to more false positive cases and vice versa. Therefore, there is a trade-off. Depending on the task the TPR or the FPR value are more important. Consequently, most classifier vary the sensitivity to tip it into one direction. The resulting set of rates are plotted as the ROC curve. This chart maps the false positive rate to the x-axis and the true positive rate to the y-axis of a chart. Consequently, good classifiers tend to the upper left of the diagram.

5.2.2. Similarity histogram

In addition to the ROC curve and the related measurements which evaluate the whole classification it is interesting to step back and take a closer look into the different NLU features by analyzing the cosine similarity distribution of duplicates and non-duplicates. Ideally should the distributions show a significant difference between duplicates and non-duplicates with minimal overlapping. This difference allows a good classification with a simple threshold classifier and indicates that more elaborated classifier with different similarity measurements would also perform better. Additionally, should the variance for the non-duplicate group be high and the variance for the duplicate group low, because a duplicate describes a very narrow relation. All duplicates must have the same meaning. Non-duplicates can range from completely different meaning to almost the same meaning.

Those similarity distributions are illustrated with two histograms. The histograms are normalized and plotted into the same diagram. The normalization step sets the area of each histogram to 1 which transforms both graphs into a probability density. Those densities allow an easy comparison between the two classes.

5.3. Results

The evaluation shows that no presented NLU feature is able to outperform the baseline TF-IDF feature. The following section discusses the evaluation details for each NLU feature.

All features utilize the in Section 4.2 established thresholding classifier in combination with cosine similarity. The ROC curves adjust the threshold value in 0.01 steps starting at

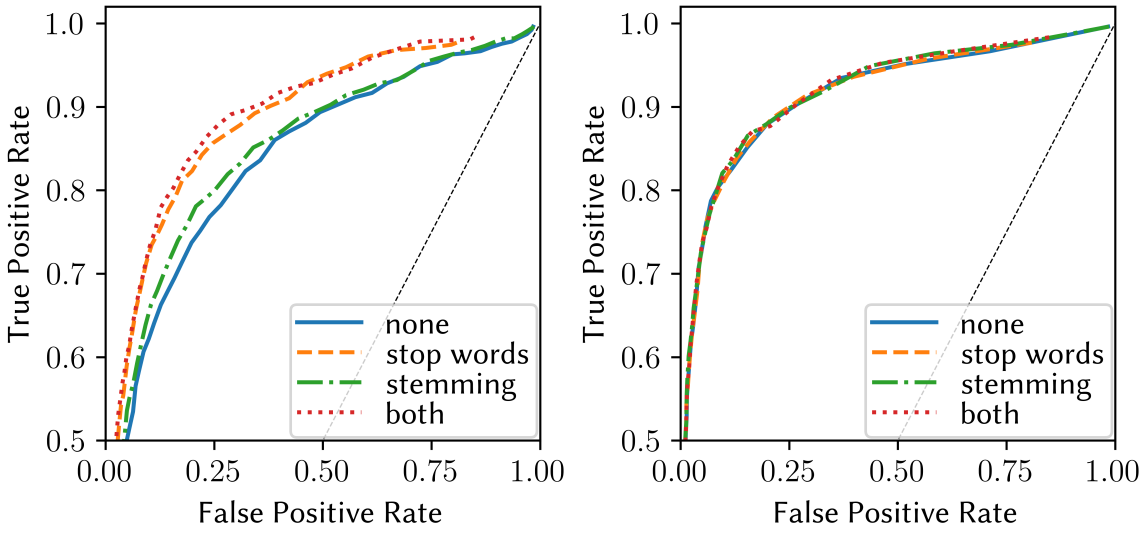
$T = 0$ till $T = 1$. The feedforward network mentioned in Section 4.2 is not used as classifier due to the small amount of labeled data. The hyperparameter settings of each feature is listed in the appendix (Appendix B).

5.3.1. Lexical approaches

Three lexical features were introduce in Section 3.3: TF-IDF, Word2Vec and GloVe. For these representations are two particular interesting points discussed in detail: the preprocessing to reduce the vocabulary size and the effect of training data on word embeddings with GloVe as example. Following those influence factor, the representations are compared with each other.

5.3.1.1. Preprocessing

The preprocessing steps stemming and stop word removal aim to reduce the sparsity of the word representations. Therefore just models which directly utilize BoW word representations can be improved by those methods. Consequently just the TF-IDF representation is evaluate regarding the preprocessing steps. The result on the Bosch dataset is presented in Figure 5.1.



(a) TF

(b) TF-IDF

Figure 5.1.: ROC curves for TF (a) and TF-IDF (b) representations with different preprocessing steps on the Bosch dataset. The y-axis is shifted by 0.5.

The TF-IDF representation does not show any improvement for stemming and the removal of stop words. The reason can be seen by comparing it to a simple TF model. The stemming does not significantly boost the performance of this model either. Most likely the vocabulary size is too small to observe any effect of generalization. Additionally are the reported improvements for other English NLP task quite small[48]. In opposition to the modest results regarding stemming the removal of stop words actually improves the TF

model. Stop words are by definition frequent words which means that the TF-IDF model already reduces the influence of those words with the IDF factor. Hence the expected positive influence of the preprocessing steps to reduce the data sparsity cannot be observed due to the small vocabulary and corpus size. The Wikipedia dataset which is even smaller shows a similar behavior. In order to align the TF-IDF representation with the other evaluated NLU features the both preprocessing steps are skipped in other comparisons.

5.3.1.2. Training data

One important aspect of word embeddings is the training corpus. Naturally embeddings benefit from a large training corpus, since more text provides more information for the word co-occurrences required to define the actual embeddings. On the other side due to word ambiguity and text-specific word usage the training dataset has to represent the word distribution of the actually analyzed text as well. Those two requirements often contradict each other. Therefore a closer look into the influences of training data for detecting duplicates is conducted on the GloVe embeddings. This evaluation requires domain specific language. Hence it uses the Bosch dataset and ignores the Wikipedia collection, because the Wikipedia dataset does not have a specific domain. Additionally, the pretrained GloVe embeddings are already trained on the Wikipedia corpus. Therefore, comparing domain specific training and pretrained models contains a bias on the Wikipedia dataset.

The pretrained embeddings are trained on a large corpus of *Wikipedia* articles and the fifth *Gigaword* corpus which contains text form several news agencies. The overall corpus contains up to 6 billion tokens[43]. The original Bosch dataset contains just 118,835 tokens. In addition, a small amount of domain-specific data is extracted from suitable Wikipedia articles. This collection contains additional 29,318 tokens. The differently trained embeddings are used as input of the mean model introduced in Section 3.3. The corresponding results of the threshold classification are illustrated in Figure 5.2.

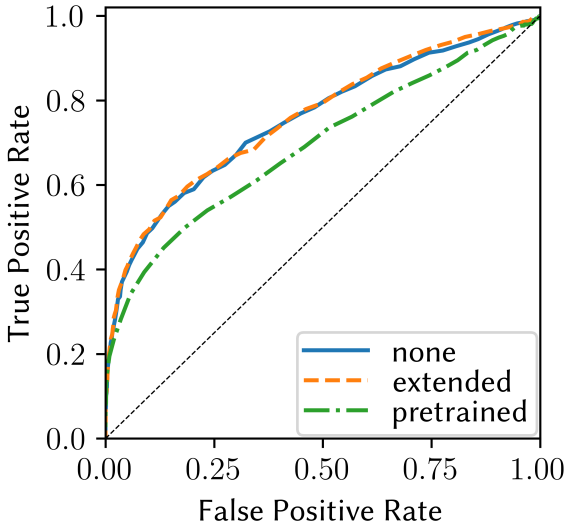
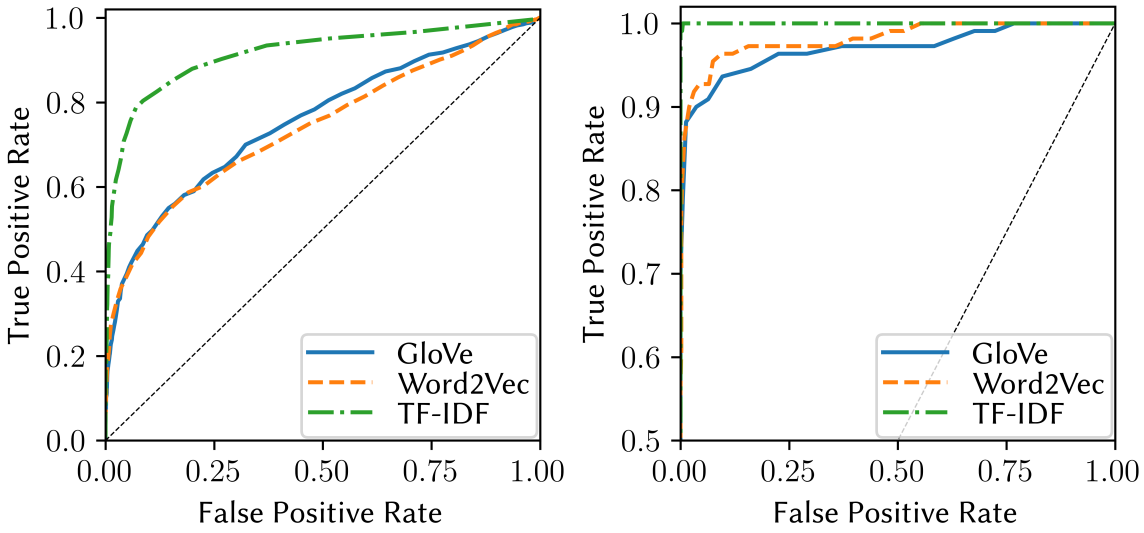


Figure 5.2.: ROC curves for GloVe embeddings trained on different datasets.

The domain specific training shows a consistently better result than the pretrained embeddings. Taking a closer look indicates two aspects which contribute to this result. Firstly, the domain-specific training captures the actually meaning even with less data apparently more accurate. Secondly, from the 11,309 different words in all Bosch tickets just 8,0085 are actually also words in the pretrained dataset. Over 3,000 words remain randomly initialized for the pretrained embeddings. However, the word embeddings trained on the extended dataset show a slight improvement over the plain dataset. The additional domain data just contains around 30,000 tokens. Consequently, the expected improvement is also small. The observed behavior is in line with the expectation and hints that a larger amount of domain data improves the model.

5.3.1.3. Comparison of lexical methods

Finally, the three introduced word representations are directly compared. The results for both dataset are illustrated in Figure 5.3.



(a) Bosch (b) Wikipedia

Figure 5.3.: ROC curves for the mean lexical models. The results of the Bosch dataset are illustrated in (a) and the results for the Wikipedia dataset in (b). The y-axis of (b) is shifted by 0.5.

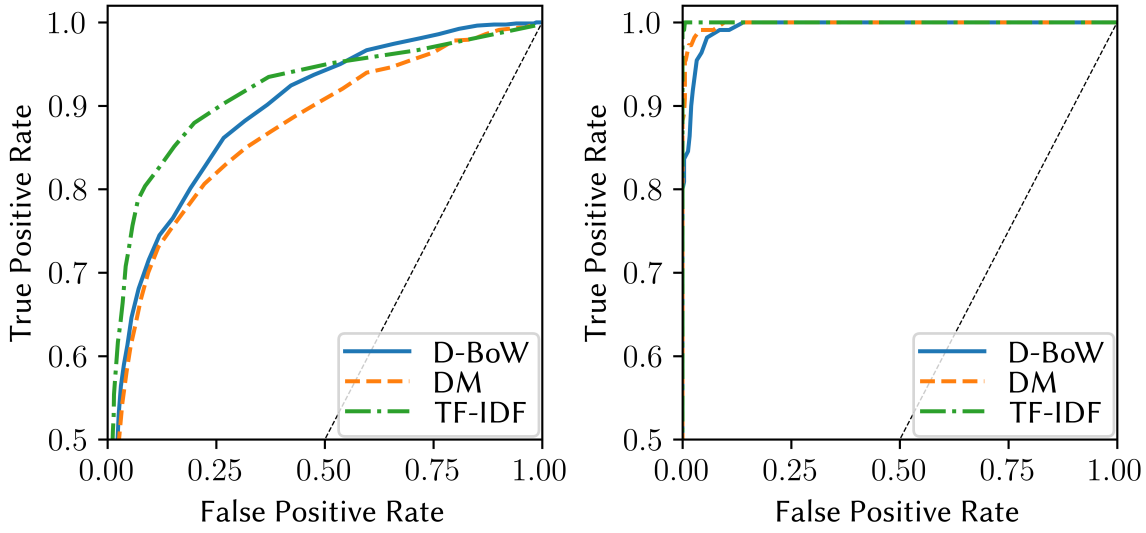
Neither the GloVe nor the Word2Vec approach outperforms the baseline TF-IDF model. But the word embeddings show a difference between the datasets. The divergence between embeddings and baseline are almost an order of magnitude greater for the Bosch dataset than for the Wikipedia collection. One key difference between the datasets on the level of lexical semantics is the heterogenic style of the Bosch dataset. Instead of full sentences bullet points and keywords occur frequently in the text. In opposition to this the Wikipedia dataset only consists out of complete English sentences. Arguably this lack of structure deteriorates the performance of the embeddings.

The TF-IDF features are even able to classify the Wikipedia dataset almost perfectly. For $T = 0.1$ the recall is 0.99 with a precision of 0.98. Naturally is the classification of the

Wikipedia dataset easier, because of the lack of an overall topic between all documents like the software application by bug reports. The low threshold indicates that most non-duplicates of the Wikipedia collection are not similar at all. This is illustrated in the appendix (Figure B.1). The Bosch histogram shows for non-duplicates a slightly bigger variation. Due to the high variance in both duplicate groups the prediction accuracy decreases significantly based on this difference.

5.3.2. Par2Vec

The first NLU feature taking compositional semantics into account is Par2Vec. The ROC curve of those unsupervised features is depict in Figure B.2.



(a) Bosch (b) Wikipedia

Figure 5.4.: ROC curves for the distributed BoW (D-BoW) and the distributed memory (DM) approach of the Par2Vec model. The results of the Bosch dataset are illustrated in (a) and the results for the Wikipedia dataset in (b). The y-axis is shifted by 0.5.

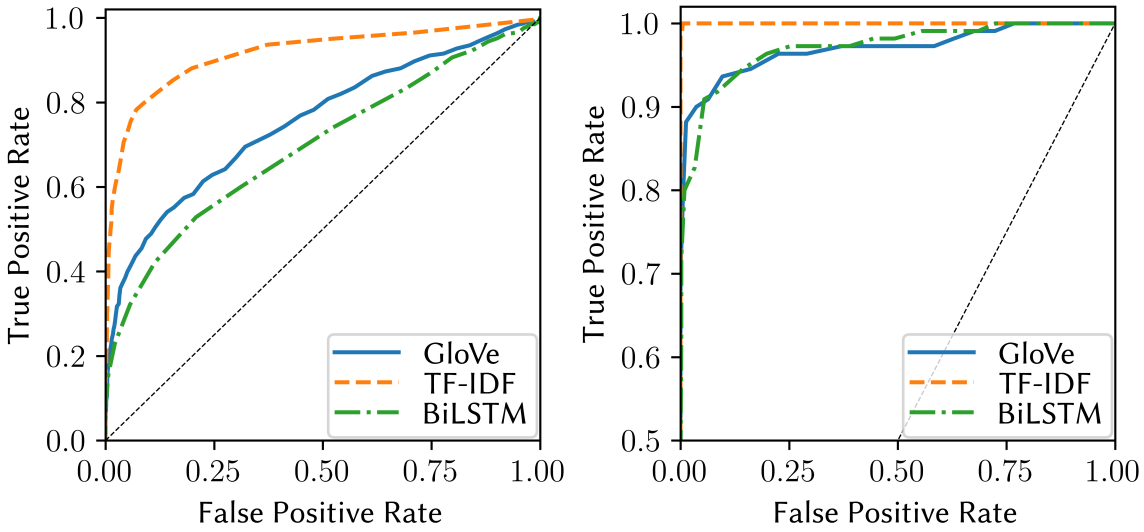
The graph shows for both models of Par2Vec an improvement compared to the simple averaging of the lexical approaches. They are on a similar level than the baseline TF-IDF although both feature are still slightly worse. Interestingly is the simpler distributed BoW model for the Bosch dataset even better than the more complex distributed memory feature. This emphasis the text structure again. The distributed BoW feature does not use a window to define context words. The representation is optimized to predict all words of one paragraph. Consequently, the feature is more suitable for the keyword oriented language of the Bosch dataset. This also indicates word embeddings with larger context windows are more robust against ungrammatical sentences.

The ROC curve of the Wikipedia dataset does not provide many information about the models due to the principal simple classification task. But the similarity histogram allows based on the variance a brief discussion. As presented in the appendix (Figure B.2)

is the variance of unique documents from the distributed memory model bigger and the duplicate variance smaller. This confirms the slightly better ROC curve as well.

5.3.3. BiLSTM

The last evaluated feature is the transfer-learning approach. The resulting ROC curves are illustrated in Figure 5.5.



(a) Bosch

(b) Wikipedia

Figure 5.5.: ROC curves for the BiLSTM model. The results of the Bosch dataset are illustrated in (a) and the results for the Wikipedia dataset in (b). The y-axis of (b) is shifted by 0.5.

The BiLSTM feature shows overall the worst performance on the Bosch dataset and improves the mean word embedding model insignificantly on the Wikipedia dataset. The sentence representations are learned with STS sentences. The text structure of the Bosch dataset does not resemble those sentences. Logically all possible learned linguistic rules cannot be applied. This example shows the obvious limitations of transfer-learning in a completely unsupervised setting.

However, the performance on the Wikipedia collection which has a similar sentence structure than the STS training task shows that the feature learned just to classify the STS sentences. The model was not able to generalize the learned representation to solve other tasks. However, the small difference between both datasets shows that at least a few linguistic rules have been learned. This hints that more elaborated models with more training data could possibly extract a larger amount of generalized linguistic rules.

6. Conclusion

This thesis compared NLU features for the usage in IR systems. Therefore, several features which encounter semantic knowledge were identified, categorized and described. Those features were evaluated by detecting duplicates in two datasets with different text structure. Due to direct comparison between documents is the detection of duplicates a perfect example for the grouping and ordering commonly used in IR.

Lexical features and Par2Vec were not able to outperform the TF-IDF baseline. Additionally both models were influenced by the text structure. For well-defined English sentences both models performed better. The additional compositional semantic incorporated in Par2Vec also improved the performance.

The transfer-learning feature was not able to improve the classification compared to word embedding approaches on documents consisting of English sentences and performed even worse on documents with a mixed text structure. This indicates that the BiLSTM was able to learn the structure of a sentence a little, but could not improve the classification with this knowledge.

In summary showed the simple TF-IDF feature an impressive result on both datasets with a small vocabulary size. On the contrary the NLU features fall short.

6.1. Further work

The evaluation results show that further work is necessary to improve NLU features for the field of IR to achieve better performances than the TF-IDF feature.

Labeled data The most obvious further work is the gathering of more data which is labeled as duplicates. More labeled data allows better classification models like the drafted feedforward network. Additionally, would it be possible to optimize the hyperparameter of all models. Due to the skew groups the labeling of randomly picked tuples out of a collection is difficult. Most likely the creation of a new document as a duplicate is easier. However, labeled data is the bottleneck for this concrete detection task and is not going to improve NLU features in general.

Delegate features The evaluation pointed out that the performance of most NLU features is significantly influenced by the text structure. Consequently are features and parameters optimized for one specific structure. But many real-world collections in the field of IR contain a mixed text structure which leads to a performance decrease for all features. Therefore, dividing a text regarding properties like structure and using specialized features for each part might improve the overall result.

Transfer-learning Despite the particular bad evaluation result of the transfer-learning feature is the approach promising for multiple NLP tasks. The minimal amount of learned sentence structure indicates that it is possible to learn linguistic rules on one task and apply those rules on another task. The results from Cer et al. and Conneau et al. show this as well. Most likely requires transfer-learning for IR due to the complex text properties of an IR corpus more sophisticated models with additional training data than the simple NLP tasks which already apply transfer-learning. The models can be improved by either training the feature additionally on the new task or by incentivizing the model to generate broader rules.

Bibliography

[1] Laura Banarescu et al. “Abstract meaning representation for sembanking”. In: *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*. 2013.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O’Reilly Media", 2009.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. “Latent dirichlet allocation”. In: *Journal of Machine Learning Research* (2003).

[4] Daniel G Bobrow. “Natural language input for a computer problem solving system”. PhD thesis. Department of Mathematics, MIT, 1964.

[5] Geert Booij. *The grammar of words: An introduction to linguistic morphology*. Oxford University Press, 2012.

[6] Samuel R Bowman et al. “A large annotated corpus for learning natural language inference”. In: *Preprint arXiv* (2015).

[7] Alexander Budanitsky and Graeme Hirst. “Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures”. In: *Workshop on WordNet and other lexical resources*. 2001.

[8] Daniel Cer et al. “SemEval-2017 Task 1: Semantic Textual Similarity-Multilingual and Cross-lingual Focused Evaluation”. In: *Preprint arXiv* (2017).

[9] Daniel Cer et al. “Universal Sentence Encoder”. In: *Preprint arXiv* (2018).

[10] Goutam Chakraborty and Murali Krishna Pagolu. “Analysis of unstructured data: Applications of text analytics and sentiment mining”. In: *SAS global forum*. 2014.

[11] Noam Chomsky. *Aspects of the Theory of Syntax*. Tech. rep. Massachusetts Inst. of Tech. Cambridge, 1964.

[12] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 2014.

[13] Noam Chomsky. “On Cognitive Structures and their Development: a Reply to Piaget”. In: *Language and Learning: The Debate Between Jean Piaget and Noam Chomsky*. 1980.

[14] Alexis Conneau et al. “Supervised learning of universal sentence representations from natural language inference data”. In: *Preprint arXiv* (2017).

[15] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*. Addison-Wesley, 2010.

[16] Scott Deerwester et al. “Indexing by latent semantic analysis”. In: *Journal of the American society for information science* (1990).

[17] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* (2011).

[18] Charles J Fillmore and Collin F Baker. "Frame semantics for text understanding". In: *Proceedings of WordNet and other lexical Resources NAACL workshop*. 2001.

[19] J. R. Firth. "A synopsis of linguistic theory". In: *Studies in linguistic analysis* (1957).

[20] Daniel Gildea and Daniel Jurafsky. "Automatic labeling of semantic roles". In: *Computational Linguistic* (2002).

[21] Yoav Goldberg and Omer Levy. "Word2Vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *Preprint arXiv* (2014).

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[23] Raj P Gopalan and Aneesh Krishna. "Duplicate bug report detection using clustering". In: *23rd Australian software engineering conference*. 2014.

[24] Gregory Grefenstette and Pasi Tapanainen. "What is a word, what is a sentence?: problems of tokenisation". In: *Proceedings of the 3rd Conference on Computational Lexicography and Text Research*. 1994.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* (1997).

[26] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences*. 1982.

[27] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *European conference on machine learning*. 1998.

[28] Adam Kilgarriff. *Wordnet: An electronic lexical database*. 2000.

[29] Tibor Kiss and Jan Strunk. "Unsupervised multilingual sentence boundary detection". In: *Computational Linguistic* (2006).

[30] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. 2014.

[31] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998.

[32] Omer Levy and Yoav Goldberg. "Linguistic regularities in sparse and explicit word representations". In: *Proceedings of the 18th conference on computational natural language learning*. 2014.

[33] Percy Liang. "Learning executable semantic parsers for natural language understanding". In: *Communications of the ACM* (2016).

[34] Percy Liang. "Talking to computers in natural language". In: *The ACM Magazine for Students* (2014).

[35] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. “Support vector machines and word2vec for text classification with semantic features”. In: *IEEE 14th international conference on cognitive informatics & cognitive computing*. 2015.

[36] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[37] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. “Building a large annotated corpus of English: The Penn Treebank”. In: *Computational Linguistic* (1993).

[38] Andrew McCallum, Kamal Nigam, et al. “A comparison of event models for naive bayes text classification”. In: *AAAI-98 workshop on learning for text categorization*. 1998.

[39] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems 26*. Curran Associates, 2013, pp. 3111–3119.

[40] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *Preprint arXiv* (2013).

[41] Andriy Mnih and Geoffrey E Hinton. “A scalable hierarchical distributed language model”. In: *Advances in neural information processing systems*. 2009.

[42] Joakim Nivre et al. “Universal Dependencies v1: A Multilingual Treebank Collection.” In: *LREC*. 2016.

[43] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing*. 2014.

[44] Martin F Porter. “An algorithm for suffix stripping”. In: *Program* (1980).

[45] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *Herbert Robbins Selected Papers*. Springer, 1985, pp. 102–109.

[46] Stephen Robertson. “Understanding inverse document frequency: on theoretical arguments for IDF”. In: *Journal of documentation* (2004).

[47] Xin Rong. “Word2Vec parameter learning explained”. In: *Preprint arXiv* (2014).

[48] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge University Press, 2008.

[49] Dafna Shahaf and Eyal Amir. “Towards a theory of ai completeness.” In: *AAAI Spring symposium: logical formalizations of commonsense reasoning*. 2007.

[50] Yang Shao. “HCTI at SemEval-2017 Task 1: Use convolutional neural network to evaluate semantic textual similarity”. In: *Proceedings of the 11th international workshop on semantic evaluation*. 2017.

[51] Richard Socher et al. “Recursive deep models for semantic compositionality over a sentiment treebank”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013.

[52] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *The Journal of Machine Learning Research* (2014).

[53] Kai Sheng Tai, Richard Socher, and Christopher D Manning. “Improved semantic representations from tree-structured long short-term memory networks”. In: *Preprint arXiv* (2015).

[54] Shuohang Wang and Jing Jiang. “Machine comprehension using match-LSTM and answer pointer”. In: *Preprint arXiv* (2016).

[55] Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. Tech. rep. Massachusetts Inst. of Tech. Cambridge, 1971.

[56] Ludwig Wittgenstein. *Tractatus logico-philosophicus*. Kegan Paul, Trench, Trubner and Co., 1921.

[57] Jiaming Xu et al. “Self-taught convolutional neural networks for short text clustering”. In: *Neural Networks* (2017).

[58] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level Convolutional Networks for Text Classification”. In: *Advances in Neural Information Processing Systems 28*. Curran Associates, 2015, pp. 649–657.

[59] Du Zou, Wei-Jiang Long, and Zhang Ling. “A cluster-based plagiarism detection method”. In: *Notebook papers of CLEF LABs and workshops*. 2010.

A. Appendix: TensorFlow Graphs

A.1. GloVe model

```
weightingFactor = tf.minimum(1.0, tf.pow(cooccurrenceCount /
    countMax, scalingFactor))

embeddingProduct = tf.reduce_sum(centerEmbedding *
   contextEmbedding, axis=1)
logCooccurrences = tf.log(cooccurrenceCount)
distanceExpr = tf.square(embeddingProduct + centerBias +
   contextBias - logCooccurrences))

cost = tf.reduce_sum(weightingFactor * distanceExpr))
optimizer = tf.train.AdagradOptimizer(learningRate).minimize
   (cost)
```

Listing A.1: Simplified GloVe cost function implementation. The constant hyperparameter are scalingFactor and countMax.

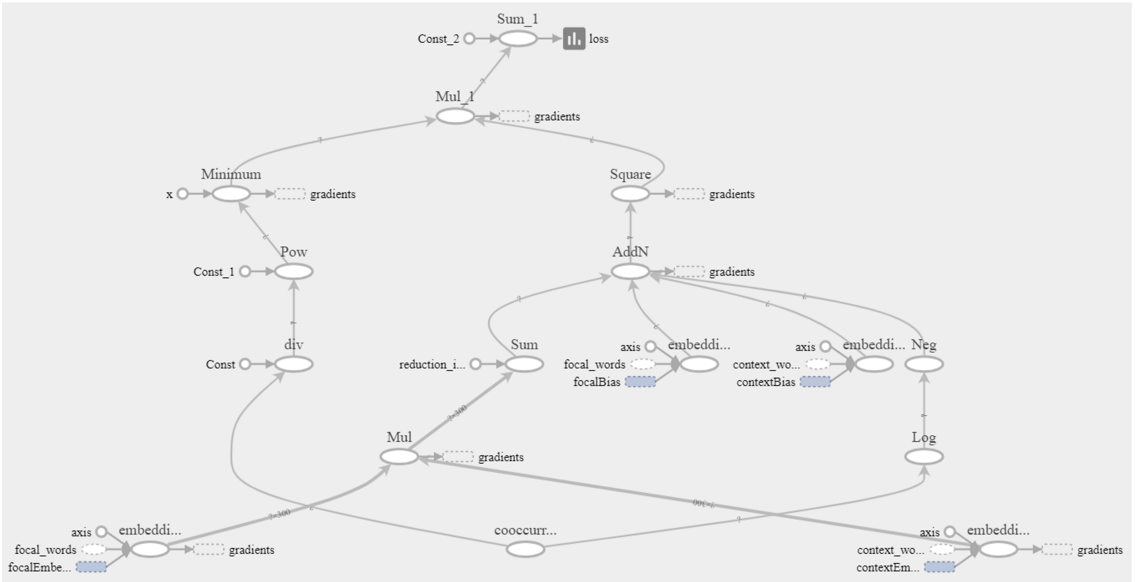


Figure A.1.: TensorFlow Graph visualization of the GloVe model

A.2. BiLSTM model

```
compAngle  = tensorA * tensorB
compDiff = tf.abs(tensorA - tensorB)
compSig = tf.sigmoid(tf.matmul(compAngle,wAngle) + tf.matmul
    (compDiff,wDiff) + bSig )
compProb = tf.nn.softmax(tf.matmul(compSig, wSig) + bProb)
prediction = tf.matmul(compProb,rClass)
regularizer = (lambdaReg/2) * tf.reduce_sum([tf.nn.l2_loss(x
    ) for x in tf.trainable_variables()])
compProb = compProb + 0.00001
crossEnt = -tf.reduce_sum(labelProb * tf.log(compProb))
cost = tf.reduce_mean(crossEnt,axis = 0) + regularizer
optimizer = tf.train.GradientDescentOptimizer(learning_rate=
    learning_rate).minimize(cost)
```

Listing A.2: Simplified Feedforward network to compare sentence representations.

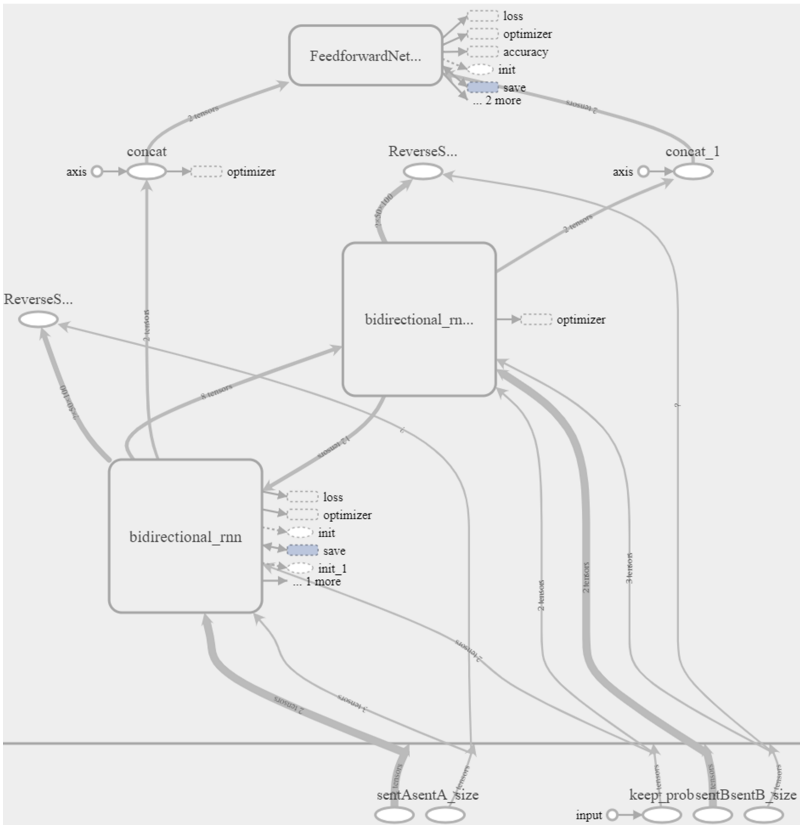


Figure A.2.: TensorFlow Graph visualization of the BiLSTM model.

B. Appendix: Evaluation data

B.1. Hyperparameters

model	parameter	value
TF-IDF	vocabulary size	10000
Word2Vec	vector size	300
	context window length	5 (symmetric)
	subsampling	0.001
	negative samples	5
GloVe	vector size	300
	context window length	5 (symmetric)
	co-occurrence maximum	100
	scaling factor	0.75

Table B.1.: List of all lexical hyperparameters with the exception of learning parameter.

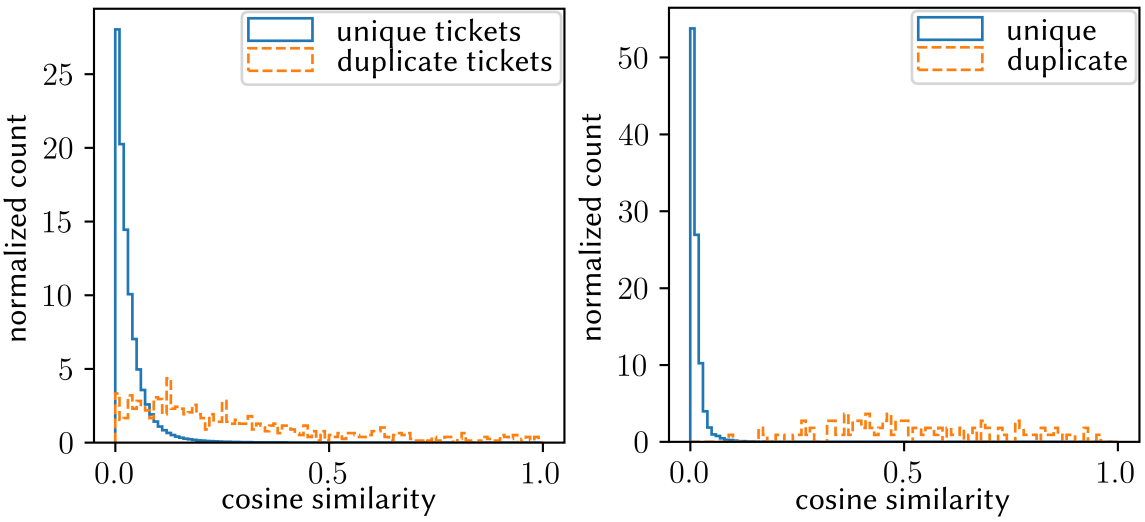
model	parameter	value
Distributed Memory	vector size	300
	context window length	5
	subsampling	0.001
	negative samples	5
Distributed BoW	vector size	300
	subsampling	0.001
	negative samples	5

Table B.2.: List of all Par2Vec hyperparameters with the exception of learning parameter.

submodel	parameter	value
Word embedding	GloVe	
BiLSTM	output size	100
	maximum sentence length	50
Feedforward	hidden layer size	50
Regularization	strength λ	10^{-4}
	keep probability dropout	0.5

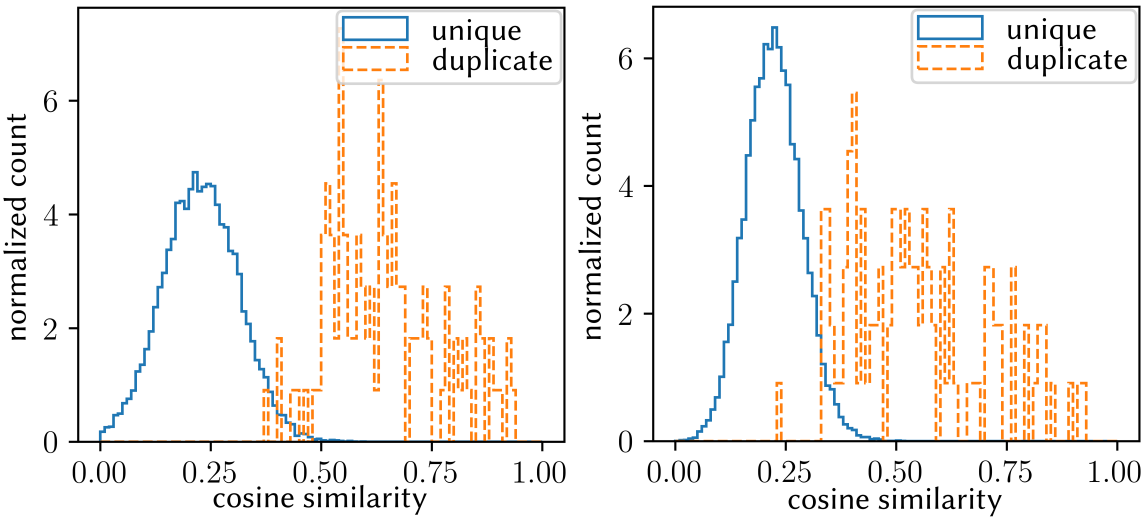
Table B.3.: List of all BiLSTM hyperparameters with the exception of learning parameter.

B.2. Similarity histograms



(a) Bosch (b) Wikipedia

Figure B.1.: Histogram of TF-IDF with the Bosch dataset(a) and Wikipedia dataset(b).



(a) Distributed memory (b) Distributed BoW

Figure B.2.: Histogram of the distributed memory(a) and the distributed BoW(b) model.