Predicting Price Residuals in Online Car Marketplaces with Natural Language Processing

Master's Thesis of

Maximilian Blanck
Matriculation Number: 1654271

in Information Engineering and Management
at the Department of Economics and Management
Institute of Applied Informatics and Formal Description Methods (AIFB)

Reviewer:	Prof. Dr. Harald Sack
Second reviewer:	Prof. Dr. York Sure-Vetter
Advisor:	Dr. Maria Koutraki
External Advisors:	M.Sc. Marcel Kurovski (inovex GmbH)
	Dr. Florian Wilhelm (inovex GmbH)

Submitted: January 23rd 2019

Karlsruher Institut für Technologie
Fakultät für Wirtschaftswissenschaften
Kaiserstraße 89
76133 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Neuenbürg, January 18th 2019

. . . . . .

(Maximilian Blanck)

Abstract

Texts written in natural language are an unstructured data source that is hard for machines to understand. The amount of text in the world wide web is growing every minute. To deal with this huge number of unstructured data automated text analysis is crucial. Natural Language Processing (NLP) is part of artificial intelligence that makes natural language texts comprehensible for machines.
In this thesis, I use state-of-the-art methods in NLP to analyze user-generated product description texts of cars with respect to their price information. Online car marketplaces offer a platform for their customers to sell and present vehicles with images, technical attributes and descriptive texts. To improve the user-experience of prospective buyers online car markets provide a neutral pricing model that predicts a car's price on its technical attributes that are available in a structured format. However, automobiles are very individual and their price also depends on their condition and additional information, which I assume is included in the descriptive texts rather than in the technical attributes. I test whether it is possible to improve price prediction by incorporating these texts. This thesis covers a human-based analysis of the user-generated descriptions. Furthermore, Artificial Neural Networks, such as Feed-Forward Neural Networks and Recurrent Neural Networks are used to predict price residuals. I use price residuals as the target variable that indicate the difference between the predicted price calculated by the price model based on the technical attributes and the observed selling price.
In this context, the thesis covers the theory about recent state-of-the-art techniques in NLP and machine learning (ML). I focus on text classification here. Different types of word embeddings and Neural Networks with attention mechanisms are presented. I demonstrate that the vehicle description texts are on average very short and approximately 50% of them explain a price residual. Nevertheless, my approach shows that price prediction can slightly be improved by analyzing user-generated description texts.

Zusammenfassung

In natürlicher Sprache geschriebene Texte sind eine unstrukturierte Datenquelle, die für Computer schwer zu verstehen sind. Die Textmenge im World Wide Web wächst von Minute zu Minute. Um mit dieser Vielzahl an unstrukturierten Daten fertig zu werden, ist eine automatisierte Textanalyse unerlässlich. Natural Language Processing (NLP) ist ein Forschungsgebiet im Bereich der Künstlichen Intelligenz, das sich mit der maschinellen Verarbeitung von Natürlicher Sprache beschäftigt.
In dieser Arbeit werden aktuelle NLP -Methoden verwendet, um benutzergenerierte Produktbeschreibungstexte von Fahrzeugen im Bezug auf ihre Preisinformationen zu analysieren. Online-Fahrzeug-Marktplätze bieten ihren Kunden eine Plattform, um Fahrzeuge mit Bildern, technischen Merkmalen und beschreibenden Texten zu präsentieren und zu verkaufen. Um die Nutzererfahrung von potenziellen Käufern zu verbessern, bieten die Online-KFZ-Märkte ein Preismodell an, das den Preis eines Fahrzeugs anhand seiner technischen Eigenschaften vorhersagt. Jedoch sind Automobile sehr individuell und ihr Preis hängt von ihrem Zustand und zusätzlichen Informationen ab. Ich nehme an, dass diese Eigenschaften in den Fahrzeug-Beschreibungstexten genauer beschrieben werden. Es wird überprüft, ob es möglich ist, die Preisprognose durch diese Texte zu verbessern. Diese Masterarbeit enthält eine manuelle Analyse der benutzergenerierten Beschreibungen. Darüber hinaus werden künstliche neuronale Netze, wie Feed-Forward Neural Networks und Recurrent Neural Networks, zur Vorhersage von Preisresiduen eingesetzt. Als Zielvariable verwende ich Preisresiduen, die die Differenz zwischen einem vorhergesagten Preis, der durch das neutrale Preismodel basierend auf technischen Attributen berechnet wurde, und dem Verkaufspreis angeben.
In diesem Zusammenhang behandelt die Arbeit die theoretischen Grundlagen zu neuesten Techniken in den Bereichen NLP und Machine Learning (ML). Ich konzentriere mich dabei vor allem auf die Klassifizierung von Texten. Es werden verschiedene Arten von Word Embeddings und neuronale Netze mit Aufmerksamkeitsmechanismen vorgestellt. Diese Arbeit zeigt, dass die Fahrzeugbeschreibungstexte im Durchschnitt sehr kurz sind und nur ca. 50% der Texte einen Preisunterschied erklären. Dennoch ist es mit meinem Ansatz möglich die Preisprognose durch die Analyse von benutzergenerierten Beschreibungstexten leicht zu verbessern.

Acknowledgments

First, I would like to thank inovex GmbH for the supportive environment and the opportunity to write my thesis in a practical research area. I want particularly express my gratitude to Marcel Kurovski and Dr. Florian Wilhelm for their dedicated supervision and inspiring discussions. Their motivational mentoring has been a great help during my work.

In addition, I want to thank Prof. Dr. Harald Sack and Dr. Maria Koutraki from the Institute of Applied Informatics and Formal Description Methods at the Karlsruhe Institute of Technology (KIT) for their feedback and generous support.

Contents

List of Figures

List of Tables

List of Abbreviations

Adam Adaptive Moment Estimation

AI Artificial Intelligence

ANN Artificial Neural Network

BERT Bidirectional Encoder Representations from Transformers

biLM bidirectional Language Model

BPTT Back-Propagation Through Time

CBOW Continuous Bag-of-Words Model

CE Cross Entropy Loss

CNN Convolutional Neural Network

CRISP-DM Cross-Industry Standard Process for Data Mining

DL Deep Learning

DM Data Mining

DNN Deep Neural Network

ELMo Embeddings from Language Models

ELU Exponential Linear Unit

FN False Negative

FNN Feed-Forward Neural Network

FP False Positive

GloVe Global Vectors for Word Representation

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

HAN Hierarchical Attention Network

IDC International Data Corporation

IDF Inverse Document Frequency

IMDb Internet Movie Database

KD Knowledge Discovery

LM Language Model

LSTM Long Short-term Memory

MAD Mean Absolute Deviation

ML Machine Learning

MSE Mean Squared Error

NLFF Natural Language based Financial Forecasting

NLP Natural Language Processing

NLTK Natural Language Toolkit

ntf_{max} Maximal Normalized Term Frequency

ntf_{rel} Relative Normalized Term Frequency

OOV Out of Vocabulary

PReLU Parametric Rectified Linear Unit

ReLU Rectified Linear Unit

RMSE Root Mean Square Error

RNN Recurrent neural networks

SGD Stochastic Gradient Descent

SiL Sigmoid-weighted Linear Unit

swr Stop Word Removal

t-SNE t-Distributed Stochastic Neighbor Embedding

tf Term frequency

tf-idf Term Frequency divided by Inverse Document Frequency

TN True Negative

TP True Positive

1. Introduction

„The reading of all good books is like conversation with the finest men of past centuries.“

René Descartes

1.1. Motivation

Technology is changing the communication of contemporary society. The International Data Corporation (IDC) estimates that by 2025 a connected person will interact with data-driven technologies every 18 seconds on average [RGR17]. Natural language, which is the language normally used for speaking and writing, defines a popular way in which people communicate with machines and humans. Texts preserve what has been said and have long been an important format for preserving human knowledge. Not only books or newspapers are based on it, texts are a common information resource on the world wide web. Especially in social media, users generate massive amounts of textual data every single second. In 2018, 473,400 short text messages are posted on *Twitter* every minute and *Reddit* receives approximately 2,000 new comments every 60 seconds [Dom18]. A vast amount of data is stored in this unstructured format, which is easy for people to understand but very difficult for machines.

In order to make human languages comprehensible for machines, research has been carried out in the field of Natural Language Processing (NLP) since the end of the 1940s. At the beginning of NLP machine translation was in the focus of investigation [Jon94]. Meanwhile, more and more research is being done in the areas of NLP and Machine Learning (ML), caused by the boom in Deep Learning (DL) that is driven by improvements in the performance of computer resources, increased availability of data and advances in research. NLP enables technologies like Virtual Assistants, such as *Apple's Siri* or *Google Assistant*, that are driven by innovative trends as shown in Figure 1.1. Many currently emerging technologies are impossible to use without the application of NLP.

Almost every interaction between man and machine has something to do with the field of NLP. Due to the large amount of information contained in texts and the frequent contact with them, their automatic analysis is crucial for overcoming information overload. Some applications that rely on the use of NLP, such as information retrieval, text summarization and categorization, aggregate information content to make it more accessible to humans and machines. For example, NLP is used to analyze news and social media posts in a short

time to predict real-time stock market prices [XWC18]. Thus, the analysis of texts can provide a strategic advantage in decision making.

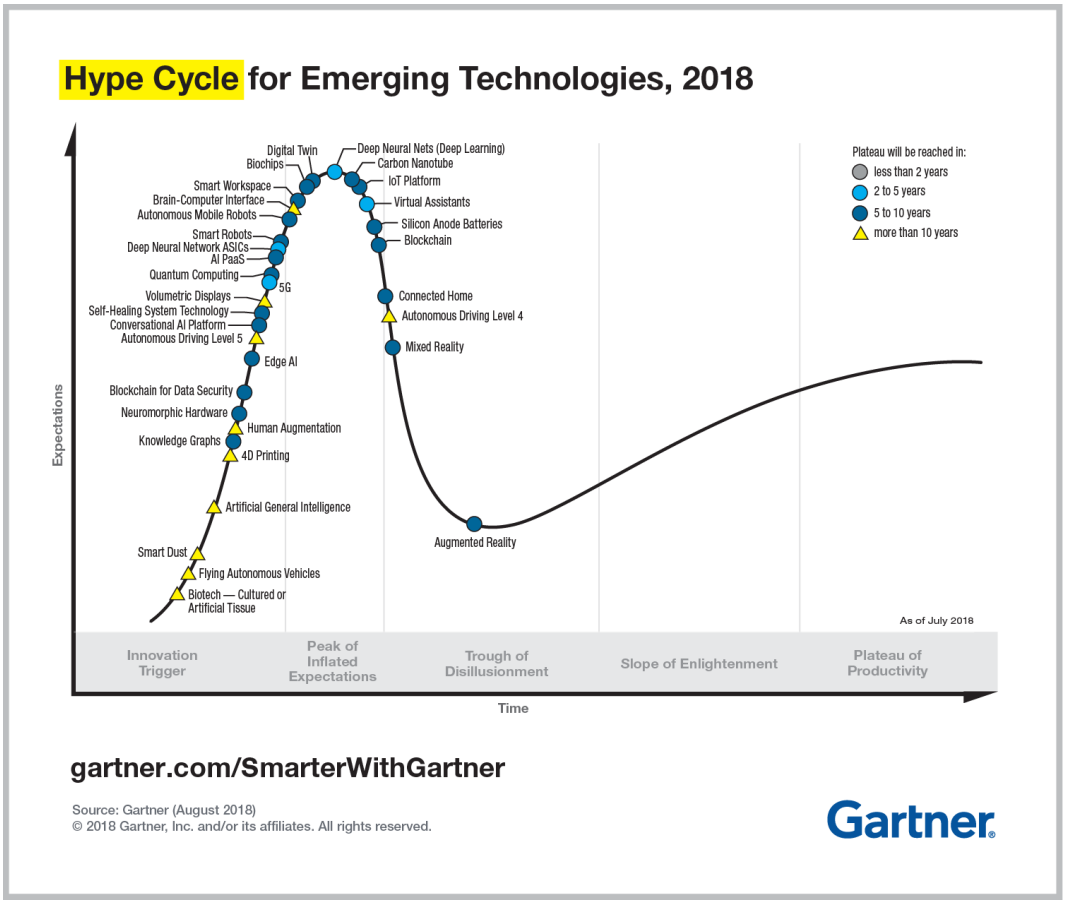


Figure 1.1.: *Gartner*'s Hype Cycle 2018 [Gar18] shows technological trends and a forecast for the future. NLP enables technologies like Virtual Assistants and Conversational AI Platforms. High expectations are set on Deep Learning.

Online marketplaces such as *ebay.com*, *amazon.com* or *alibaba.com*, also prefers to obtain more insights into pricing strategies and customers' willingness to pay. Products are usually presented with images, structured attributes, such as technical product-specific data, and descriptive texts. The structured data and user-interaction data can be evaluated through complex Data Mining (DM) models to gain information about market prices and customer behavior. For example, marketplaces analyze a product's sales volume in comparison to it's price and characteristics. Meanwhile, many products do have similar specifications and thus the structured attributes do not necessarily provide enough information for price differentiation. Product description texts that display the products in a more detailed way may help to provide this distinctiveness.

Online car markets offer their customers the opportunity to buy and sell new and used vehicles on their market platform. A couple of different online car markets exist worldwide, e.g. *mobile.de*, *pkw.de*, *cars.com* and *autotrader.com*. The vehicles are usually represented by

the same data formats as in other online marketplaces that were mentioned above. Sellers can add some images of a car that is to be sold and they are able to set structured attributes. This could be the selling price, the color or the horse power. Finally, a user-generated text is added in which the seller can specify any additional information. These texts have structural differences. They can be written in the form of a list, single words or paragraphs. The vocabulary in the texts is very domain specific and contains many terms that are used in the automotive industry.

Because cars are more individual than usual consumer goods, such as milk or chocolate, it is hard for consumers and even for car experts to predict a market price solely on technical attributes. To provide users with more price information about a car, some online car markets predict vehicle market prices based on this structured attributes. Predictive models specify market prices for cars on the basis of previously sold cars. However, the structural attributes may not anticipate all information about a vehicle, caused by a high diversity in appearance and optional equipment. This is particularly true for older vehicles, as their individual condition is more important than that of newer vehicles. Therefore, I assume that additional information for price determination is contained in texts that the customers add to present their vehicles. Like humans, machines should also analyze the provided textual information in order to achieve more accurate results in predicting market prices.

1.2. Research Questions and Objective

This thesis will examine the possibility to improve price prediction for vehicles in online car markets by the analysis of their description texts. **The prediction of price residuals that indicate the difference between a predicted price based on structured attributes and the selling price will be in the main focus**. Due to the arbitrary nature of natural language this will be a challenging task. Therefore, current state-of-the-art methods in the field of NLP will be reviewed and evaluated against this objective. The research will specifically cover text preprocessing, text classification and text analysis in general. Two main objectives will be in the center of investigation that raise two research questions:

1. *Which state-of-the-art NLP-approaches exist to analyze user-generated product description texts with respect to their influence on the product price?*

2. *How do such approaches perform on real description text data for vehicles in an online car market?*

The thesis will test if the assumption holds that descriptive texts in an online car market contain information about the price. It will provide a detailed analysis of this texts to gain more insights into the data, which can not be obtained from structured car attributes.

1.3. Course of Investigation

Six main chapters build up the entire thesis. This motivational introduction (Chapter 1) will be followed by Chapter 2 that focuses on theoretical foundations in the field of NLP. It gives an overview of well-established methods in DM-projects and NLP in general. In addition, it discusses basic text preprocessing methods, such as tokenization, stemming or stop word removal. Moreover, the theory behind Artificial Neural Networks (ANNs) will be examined in the later parts of this chapter. Afterwards, the first research question is answered in Chapter 3. General state-of-the-art techniques in NLP, such as word embeddings, and application based research are the main objectives in this chapter. In the approach (Chapter 4) the focus is shifted to the practical analysis of vehicle description texts in an online car market. The theory is applied to real data and the second research question is answered. Chapter 5 will critically debate the outcome and results of the approach. Finally, Chapter 6 provides a conclusion and outlook according to the thesis' objectives and research field.

2. Theoretical Foundations

> *„It is the theory which decides what can be observed."*
>
> Albert Einstein

This chapter's scope are the theoretical foundations of Natural Language Processing (NLP). Due to the large versatility and enormous number of different applications in NLP, the chapter will only focus on subjects, which are important to this thesis. For a more comprehensive summary of the territory, I can recommend *„The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data" by Ronen Feldman and James Sanger, [FS07], „The Handbook of Natural Language Processing" edited by Nitin Indurkhya and Fred Damerau [ID10], „Machine Learning: An Algorithmic Perspective" by Stephen Marsland [Mar15] or „Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques" by Medi Allahyari et al. [All+17].* The chapter starts with the explanation of Knowledge Discovery and the Cross-industry Standard Process for Data Mining (CRISP-DM). It continues with the investigation of NLP in general and discusses technologies and methods that are crucial for this thesis' implementation. Due to the current trends in NLP, Neural Networks such as Feed Forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs) are presented.

2.1. Knowledge Discovery and CRISP-DM

For understanding the thesis' bigger picture, I will shortly introduce the terms Knowledge Discovery (KD) in Databases and Data Mining (DM). KD concerns

> *„the nontrivial extraction of implicit, previously unknown and potentially useful information from data […]."* [SSG08]

In literature, Data Mining and KD in Databases are often treated as synonyms but actually Data Mining is a sub-task of the KD-process, which is similar to the CRISP-DM described in the following section. DM can be considered as the modeling phase that extracts information and patterns from prepared data with different approaches [FPS97].

The Cross-industry Standard Process for Data Mining (CRISP-DM) was developed in 2000 and is applicable to many different kinds of data mining problems. In this master thesis, this process will be used to structure the whole project. Figure 2.1 gives an overview of the whole CRISP-DM. In detail, the process consists of six main steps, which are Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and

Figure 2.1.: Cross-industry Standard Process for Data Mining [Cha+00]. It illustrates the six phases of a Data Mining project.

Deployment [Cha+00]. Furthermore, the whole process is iterative, meaning that every step will be treated several times. For brevity's sake, I will not cover the deployment phase here.

Business understanding is the main task in the beginning of a data mining process. From a business point of view the whole project should be understood and analyzed. Furthermore, this phase tries to transform the whole problem into a data mining perspective, which focuses on derived business objectives.

Data collection is the initial step in **data understanding**. This step serves to provide initial insights into the data, form early hypotheses and identify problems within the data. In this way it is also possible to redefine the whole problem and return to the business understanding [Cha+00]. Afterwards, the focus is on **data preparation**, where different methods are applied to obtain a final set of data that can be used as input in the modeling phase. Relevant tasks for data preparation in NLP are described in the next section. Besides preprocessing, the data is usually split into smaller subsets before modelling. One is called the training set on which the model is trained, the second is the test set and it is used for evaluation. Sometimes a third set, called validation set, provides data to train hyperparameters of a DM-model.

Modeling is the stage of selecting and applying various models to prepared data. There are a number of techniques for each distinct type of data mining problem. It is often necessary to turn back to the data preparation phase as some models require different forms of input. Before **deployment**, where the model is transferred into production, it is necessary to evaluate the applied models with metrics that focus on the business objectives. Therefore, the **evaluation** step measures the performance of all previous phases with respect to the business targets.

2.2. Natural Language Processing

Natural language processing (NLP) is the main research area of this thesis. It is related to other fields such as Artificial Intelligence (AI) or Machine Learning (ML). The section covers different feature preprocessing methods, e.g. stop word removal and vector space model. This section also deals with evaluation metrics and discusses a classical machine learning model.

2.2.1. Definition and Distinction

The following subsection serves to put the topics Artificial Intelligence (AI), Natural Language Processing (NLP), Machine Learning (ML) and Deep Learning (DL) in one context. **AI** is a very broad term and is a way to describe systems that are able to „think“ [HKW08, p. 13]. In literature, there are many different explanations, which makes it difficult to define this topic exactly. For this thesis, the interpretation of [HKW08] will be taken, because it includes NLP and puts it in relation to AI. AI consists of four main parts, which are Machine Learning, Reasoning, Planning and NLP, as shown in Figure 2.2. **Reasoning** enables a machine to provide suggestions based on data, whereas **Planing** empowers systems to act autonomously on the interpretation of data.

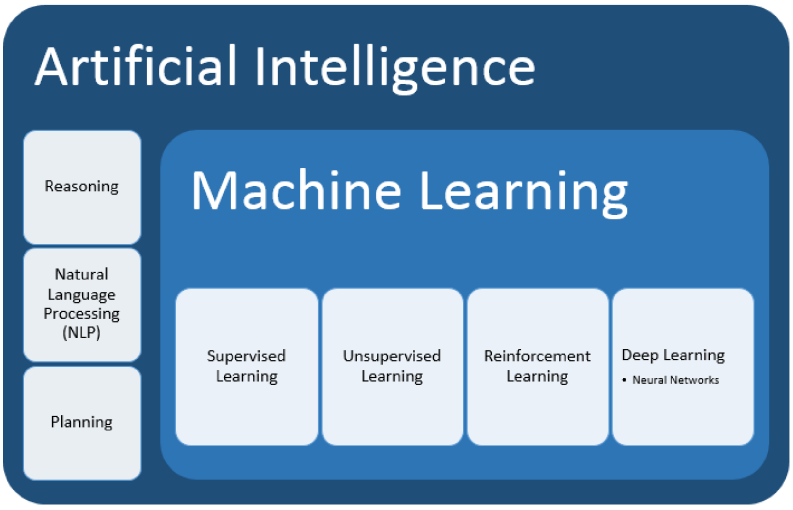


Figure 2.2.: Artificial Intelligence contains Machine Learning and NLP [HKW08, p. 13].

NLP deals with

> „*the use of human languages [...] by a computer*“. [GBC16, p. 461]

It does have many different applications, which all refer to humans' unstructured natural language. For example, its application areas are machine translation, speech recognition, dialog systems, named entity recognition, information retrieval and text classification. Thus, the domain of NLP encompasses all interactions between a computer and a human, by the use of written or spoken natural language. It is a research and application field,

which is concerned with the manipulation and understanding of natural languages [Cho03]. The processing of human language is based on understanding the intended meaning of a message, which is difficult even for humans, e.g. when irony is used. All components of natural language, such as phonetics, phonology, morphology, syntax, semantics and pragmatics, must be taken into account in order to gain complete understanding of a message.
Phonetics is about the acoustic properties of a sound produced by the human vocal tract. It examines how sounds are physically constructed, e.g. with the tongue or the lips.
The sound of a particular human language is studied by **phonology**. For example, the English language has 45 distinguishable sounds called phonemes. Phonetics and Phonology are particularly important aspects in speech recognition when converting sounds into real words that can be processed by a computer.
Morphology concerns about the meaning and the architecture of words. Stemming and lemmatization, which are described below, are based on this component by transforming words like „going“ back to their word stem „go“.
The ordering of words and the building of grammatical correct sentences is investigated by the **syntax**. In contrast, **semantic** examines the meaning of sentences that are constructed by the use of syntax and morphological word forms.
To obtain the intended overall meaning of a message, **pragmatics** uses the context of the situation. For example, let us assume, somebody asks: „Could you pass the salt?“. Given the context of the situation, the question is actually a request to pass the salt and not a question if someone is able to do that [Bri13]. Therefore, a computer needs to take all parts of natural language into account to use it.

One famous definition of **Machine Learning** (ML) is based on the idea of experience and illustrates the learning part in ML:

> *„A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.“* [Mit97]

The following example should help to clarify this definition. Suppose, we have a rain forecasting system that predicts whether it will rain today or not (T). The system is a binary classifier and its performance will be measured by its accuracy (P), which will be defined in Section 2.2.4.2. It learns from time and historic weather data (E) to predict the right outcome.
In addition, ML is one central subject in AI and splits into four subcategories: Supervised Learning, Unsupervised Learning, Reinforcement Learning and Deep Learning.
Supervised and Unsupervised Learning are the two main types of data mining problems. The difference between those tasks is the structure of training data. In a supervised set of data the target variable is known and can be used for model learning. Unsupervised problems do not have a known outcome and the solution is often based on instance similarity, patterns and groups.
Reinforcement Learning is similar to Supervised Learning, but does not learn on a fixed dataset. It is often used in game playing or self-driving cars. Reinforcement Learning algorithms use trial and error to learn from a given objective. They interact with an

environment and use a feedback to improve their experience [GBC16, p. 106].
In contrast to the three topics mentioned above, **Deep Learning** (DL) is a method of solving them by enabling a computer to infer complex patterns from simpler ones [GBC16, p. 5]. Deep neural networks (DNN) are the basic parts of DL and will be described in Section 2.3. According to the definition above, it is difficult to say whether DL is used or not, because a complex pattern could also be a neural network with a single hidden layer and many neurons. The network does not have to be deep in every dimension.

The terms AI, NLP, ML and DL should not be understood by itself only. The boundaries between the topics are not strict. Particularly, ML is extensively used in NLP to solve various types of problems.

2.2.2. Feature Selection and Preprocessing

Feature selection and preprocessing are significant tasks in Artificial Intelligence and mainly represent the data preparation step in the CRISP-DM. Especially in NLP, this task does have tremendous impact on the success of text analysis [All+17]. This is mostly caused by the unstructured and arbitrary nature of text data. Furthermore, machines need structure and numerical data. A couple of approaches for this transformation task, e.g. word embeddings or the vector space model, exist. This section's scope lies on the theoretical foundation of different preprocessing and feature selection techniques. This section will be accompanied by the English phrase „the best fox is running“ as an example to illustrate the application of preprocessing.
Nevertheless, every routine should be used with care. It is not always the case that a reasonably good preprocessing method leads to better results in every application [Gre+16]. The so-called no free lunch theorem makes it necessary [Wol95] to evaluate every suggested method separately in the practical part of this thesis.

2.2.2.1. Tokenization

For processing written natural language it is inevitable to split texts into smaller units, which are called tokens. Computers need to distinguish single entities of a text and tokenization is used to create them. Usually tokens represent simple words, which are the smallest independent units of natural language [RMD96, p. 15]. Furthermore, tokens can consist of idioms or a hyphens, e.g. „user-generated“. Tokenization breaks running texts into short text entities and is the very first task in any text preprocessing cycle [ID10, p. 4]. Besides the partition of small units, whole sentences can also be the output of a tokenizer. A simple word tokenizer can be realized in many languages by splitting the text at the occurrences of space symbols. This simple baseline approach does have a couple downsides, due to the lack of identifying words that semantically belong together [WK92]. However, a simple tokenizer divides the phrase, which was introduced above, is into the following five tokens:

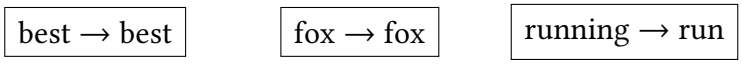
By using tokens, so-called *n*-grams can be created, which indicate a token set with the length of *n*. „Gramma “ is the Greek word for letter or token. When talking about a set of *n* letters in words, it is about character *n* grams.

2.2.2.2. Stop Word Removal

A very important approach to reduce the huge raw input space in NLP is stop word removal (swr). Most languages have specific words, which do appear more often than others or do not include much information about the content of the text, e.g. auxiliary verbs or articles [FS07]. Due to this, it often makes sense to exclude this so-called stop words in further analysis. In English such words could be "the", "a" or "an" and for German typical stop words are the articles "der", "die" and "das". The elimination could be done by checking the words against a standardized stop word list. These lists are available in literature and are often implemented in different software packages [DS17]. In our example, „the“ and „is“ are eliminated. Stop word removal should be used with care, especially in sentiment analysis, which attempts to predict a positive or negative intention of a text. The removal would exclude words that are able to change a whole statement, such as „not“ or „none“.

2.2.2.3. Stemming

Besides stop word elimination, stemming is a useful technique to map words to their word stems and further reduce the input dimension. This helps to extract the real meaning of a text and makes the unstructured data better accessible for a machine. The first stemming algorithm based on deleting longest suffixes and spelling exceptions was developed in 1968 [Lov68]. By now, the porter stemming algorithm is a state-of-the-art approach and strips suffixes from words to retain the word stem [Por80]. While this method performs well in English, there are some drawbacks for the German language, due to the fact that German words are not usually build by adding suffixes. However, there is a German equivalent based on Porter's idea and the string processing language Snowball [PR01]. By using the English Porter Stemmer the words „best“, „fox“ and „running“ are assigned to the following words:



2.2.2.4. Lemmatization

Lemmatization is the process of mapping every word in a text to their dictionary type or intended originating structure. Verbs are transformed to their infinite form, a noun is reconstructed to it's singular representation and adverbs or adjectives anticipate their positive format [Liu+12]. The method is based on morphological analysis and often uses a dictionary, for example WordNet [Fel98], where the lemma of every modified word form could be retrieved. This preprocessing step is similar to stemming and reduces the input space, by mapping different word forms to their common representation. Since

lemmatization is supported by dictionary entries, it is able to map „best“ to its lemma „good“:

best → good	fox → fox	running → run

2.2.2.5. Vector Space Model

Besides, preprocessing the words themselves, their representations have to be changed into a machine readable format. Meanwhile, a couple of different approaches have been developed to transform texts into different kinds of numerical representations. Some of them only represent statistics of a word, such as the one-hot-encoding, and other formats also include the word's context, e.g word2vec (Section: 3.1).

The **Vector Space Model** is an approach that transforms a text into one vector. It is based on one-hot-encoding of words. Given a set of textual documents (corpus), it is possible to create a vocabulary with the length of N. The one-hot-encoded word vector represents a word by 1 at the corresponding vocabulary entry. For example, if the term „fox“ is the i-the unique word in the corpus, the vector has the length of N and a 1 in the i-th position, all other entries are 0:

$$\underset{i\text{-}th\ position}{one\text{-}hot(fox) = (0, \ldots, 1, \ldots, 0)} \in \mathbb{N}^N \tag{2.1}$$

The vector space model extends this model to documents. The function Ψ maps any document d to it's vector space representation. The distinct words, which are also called terms, in the vocabulary are represented by $t_1, \ldots, t_n$.

$$\Psi : d \longmapsto \Psi(d) = (tf(t_1, d), tf(t_2, d), \ldots, tf(t_{\mathrm{n}}, d)) \in \mathbb{R}^N \tag{2.2}$$

Ψ counts the occurrence of each term (tf) in the vocabulary per document [HQW06, p.203-204]. Therefore, the document vector can have more than one entry that is not 0. The function $tf(t_i, d)$ specifies how often the i-th vocabulary word appears in the document d. Besides simply using the term frequency (tf), it is also possible to give every particular word a weighting, according to its relative appearance in the corpus. This could be done by using term frequency divided by Inverse Document Frequency (tf-idf), which gives less meaning to common words in a corpus. To calculate the tf-idf for a term in a document, a normalized term frequency must first be determined. The relative normalized term frequency ($\mathrm{ntf}_{\mathrm{rel}}$) is defined as:

$$ntf_{rel}(t_i, d) = \frac{tf(t_i, d)}{\sum_{t_m \in d} tf(t_m, d)}. \tag{2.3}$$

It calculates the term frequency of the i-th term t_n in d and divides this by the sum of all other term frequencies in d. The maximal normalized term frequency ($\mathrm{ntf}_{\mathrm{max}}$) puts the highest term frequency in d into the denominator:

$$ntf_{max}(t_i, d) = \frac{tf(t_i, d)}{\max_m tf(t_m, d)}. \tag{2.4}$$

Both equations can be used for retrieving an adjusted term importance throughout a particular document d. For incorporating the word relevance with regard to the whole corpus, it is necessary to calculate the Inverse Document Frequency (IDF):

$$idf(t_i) = log\left(\frac{|D|}{|d : t_i \in D|}\right) \tag{2.5}$$

$|D|$ is the total number of documents in the corpus and the denominator represents the occurrence of the term t_i in all documents. For frequent terms in the corpus, $|d : t_i \in D|$ gets big, which results in fraction closer to 1. By reversing the Document Frequency with the logarithm, the common words are given a lower weighting, because they have less distinctiveness than rarer words.
The actual weight w for every word in the document is calculated by the product of normalized term frequency and inverse document frequency:

$$w(t_i, d) = idf(t_i) \cdot ntf(t_i, d) \tag{2.6}$$

By using the Vector Space Model, the sparsity of the document vectors could be one major problem, due to the fact that $N - length(d)$ positions are 0 [SS09, p. 12]. Another issue is that the distance between two different document vectors is very small (curse of dimensionality). Therefore, it is not easy to distinguish between documents, especially when it comes to grouping similar documents, e.g. in clustering.

2.2.3. Predictive Data Mining Models

As illustrated by the CRISP-DM, Data Preprocessing is followed by the modeling phase. Hereby, a Data Mining model is trained to predict the target variable that was defined in the step of Business Understanding. This subsection will illustrate details about Random Forest that is widely used in practice. I will use this model to provide a baseline for my approach. In addition, the section offers basic concepts that should be taken into account when dealing with modeling, such as the bias variance trade-off. The section ends with the explanation of this trade-off and starts with the presentation of Random Forest. Due to the recent trends in NLP, I will focus on Artificial Neural Networks (ANNs) and explain the concept of Feed-Forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs) in the Sections 2.3 and 2.4.

2.2.3.1. Random Forest

Random Forest is a DM-model that is based on an ensemble of randomly grown decision trees. The concept of it was introduced by Breiman in 2001 [Bre01]. It uses the decision trees to predict a class or perform a regression on a given input. The strength hereby is that the outcome is produced by the majority vote of all trees. Let the variable n_{tree} indicate the number of trees constructed by Random Forest. The training of Random Forest starts with the extraction of n_{tree} random bootstrap sets from the training dataset. Each tree is then learned by one set [LW02]. To understand the concept of Random Forest, it is necessary to give an overview of decision trees at first.

A decision tree can be seen as a nested concatenation of conditional statements. Moreover, it is a data structure that is made of decision nodes and leafs [Rug02]. The input are values of features and the tree predicts a target variable by checking whether the features of an instance have a specific value. The order of the conditional statements is like a tree in which the root node is the starting point.
To construct a tree, I will focus on the C4.5 algorithm that is widely used in practice and creates a decision tree by applying the divide and conquer paradigm. For the sake of brevity I will only explain the important parts of C4.5 here. Further details on C4.5 can be found in [Rug02] or [LW02], which form the basis of literature for this section. The training begins with the creation of a root node, where all instances from the training set are associated with. If there are two or more classes in this set, the algorithm selects the most distinctive feature at the root node that divides the training set into a set of subsets. Each subset is again associated with a node. At every node, a feature splits the tree again until the node's subset contains instances that are labeled with one or a few classes only. The most important part in the C4.5 algorithm is the selection of the best discriminating feature.
I will introduce Entropy $H(Y)$ and Information Gain $I(Y, X)$ for a discrete feature X. They are used to choose the feature at which the tree should be split. At first, $H(Y)$ is computed for the set of instances Y that is associated with a node. The Entropy

$$H(Y) = -\sum_{i=1}^{|C|} \frac{freq(C_i, Y)}{|Y|} \log_2 \left(\frac{freq(C_i, Y)}{|Y|} \right) \tag{2.7}$$

contains the number ($freq(C_i, Y)$) of instances in Y that are labeled with the class C_i. The entropy of Y is than used to calculate the Information Gain

$$I(Y, X) = H(Y) - H(Y|X) = H(Y) - \sum_{j=1}^{s} \frac{Y_x}{|Y|} H(Y_x) \tag{2.8}$$

for Y given the feature X. The feature X has got s different values and the set of instances in Y, where X has the value $j = 1, \ldots s$, is denoted with X_j. $I(Y, X)$ is computed for every feature and the one that produces the highest Information Gain is selected to split the tree. This procedure is repeatedly applied on the training data until the next split does not improve the Information Gain. Afterwards, C4.5 calculates the classification error in each node by adding the errors of its child nodes. If this error is greater than classifying the entire node with one class, the algorithm labels all instances in this node with the most common class.
Aside from training a number of decision trees, Random Forest does not necessarily split its trees by the feature that provides the best Information Gain. It randomly picks a feature from a subset of best splitting features. This helps to improve generalization [LW02].

2.2.3.2. Generalization and the Bias-Variance Trade-Off

Two important issues in the modeling and evaluation phase are the concept of generalization and the trade-off between bias and variance. As mentioned above, a model learns

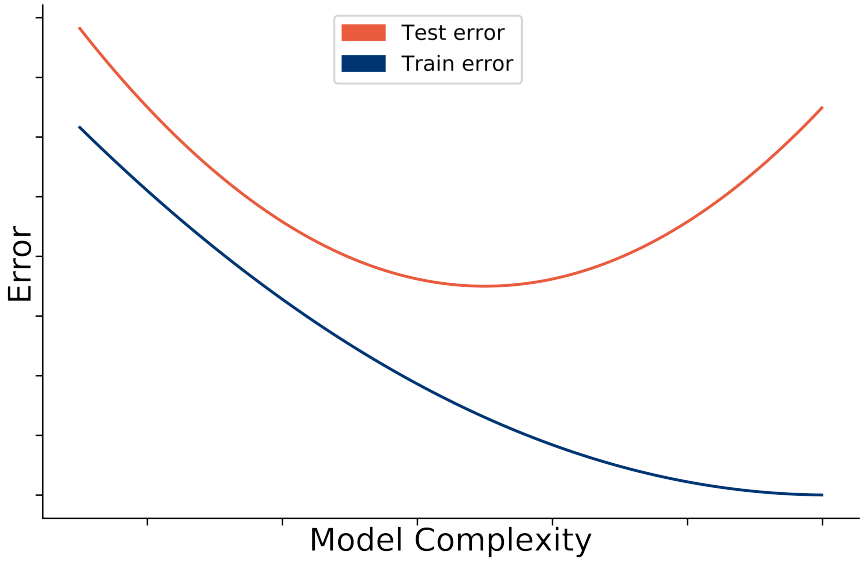


Figure 2.3.: Bias-Variance Trade-off cf. [HTF09]. The red graph shows the error on the test set, which is higher than the error on the train set (blue). The model complexity is on the x-axis to show how the error develops with changing model capacity.

from a given training dataset and adapts its parameters to it. The overall aim of every DM-model is a good prediction performance on previously unseen instances, which is known as generalization. The problem is that a model could easily overfit on the training dataset. Thus, it predicts training instances very accurate, but often fails to deal with unknown data, which is simulated by the test data. Plot 2.3 shows the trade-off between the error on the test and training dataset. The x-axis shows the model capacity, which also illustrates the adaption towards the training data. The train error is usually below the test error and decreases constantly when the model's complexity is increased. The best generalization is achieved when the test error has a global minimum. The bias indicates whether the model correctly analyzes the connection between input and output. In contrast, variance measures the generalization of the model and simultaneously indicates whether the model learns the underlying structure of the training data by heart [HTF09].

2.2.4. Evaluation Methods

The evaluation phase is one of the basic parts in any data mining project and succeeds the modeling step. This section presents various methods for evaluation and measuring the performance of different ML approaches. It particularly presents metrics that are used to score supervised learning tasks. It starts with the definition of the confusion matrix and ends with the description of evaluation measures, such as accuracy and recall.

2.2.4.1. Confusion Matrix

The confusion matrix is a method to represent results of a supervised learning task and is used in classification tasks, such as the rain forecasting example that was given in Section 2.2.1. The system predicts whether it rains or not, which could be realized by two classes „rain“ and „dry“. The confusion matrix compares the true labels of instances with their predicted classes. The sum of the matrix's diagonal is the amount of the overall true predictions.

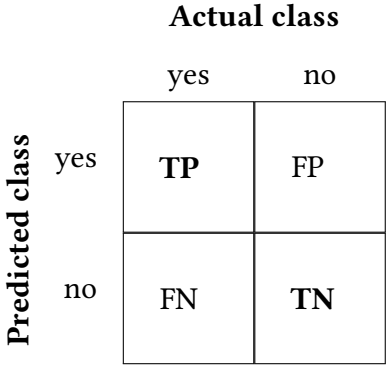


Figure 2.4.: Confusion Matrix contains the values for True Positives TP, True Negatives TN, False Positives FP and False Negatives FN. The matrix is illustrated for a binary class with two outcomes (yes, no).

In a multi-labeling situation, the confusion matrix can easily be expanded by adding a column and a row per class. An exemplary matrix with a binary class is illustrated in Figure 2.4, where TP stands for the number of true positives, TN for true negatives, FP for false positives and FN for false negatives. Applied to the rain forecast example, a TP would be if the system predicted the class „rain“ and the weather was rainy that day. A TN would indicate that „dry“ was predicted, which was true for that day, while a FP expresses that the system predicted „dry“. A FN is the opposite of a FP.

2.2.4.2. Evaluation Measures

To get an overview of the performance of a specific algorithm, **accuracy** is a good choice and can be determined for a classification problem. The accuracy is calculated by the sum of the diagonal, divided by the sum of all entries in the confusion matrix [Mar15, p. 23].

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \tag{2.9}$$

Recall is a measure, which calculates the percentage of relevant instances an algorithm has selected. Thus, it evaluates how many TPs are found from an actual class.

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{2.10}$$

Precision is another measure for evaluation. It computes the percentage of TPs in a set in which a model has classified all instances as positive [Mar15, p. 23].

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{2.11}$$

The **F-Measure** represents a harmonic mean between recall and precision. The weighting can be adjusted by the parameter β. This makes F_β adoptable to distinct data mining tasks, e.g. in a search engine, where recall could be more important than precision.

$$F_\beta = \frac{(1 + \beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{2.12}$$

With a $beta = 1$ both values are weighted equally, which is often described in literature as the F1-metric [Ber04, p. 211]. Besides the evaluation measures that are based on labeled target variables, metrics for continuous values do also exist.
Root Mean Square Error (RMSE) is used to score a given prediction for a set of instances. It calculates the squared difference between a true outcome y and a forecast $\hat{y}$ [XWC18].

$$RMSE(y_i, \hat{y_i}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (|\hat{y_i} - y_i)^2} \tag{2.13}$$

Mean Absolute Deviation (MAD) is similarly used as RMSE. It computes the absolute difference from a set of instances to their mean [OW17].

$$MAD(y) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \bar{y_i}| \tag{2.14}$$

2.3. Feed-Forward Neural Networks

This section's focus is on the concept of Artificial Neural Networks (ANNs) and Feed-Forward Neural Networks (FNN), as this is the first developed network type. ANNs take the human brain as an example and are based on the neurons of McCulloch and Pitts, which were established in 1943 [MP43]. Figure 2.5 shows the mathematical model of a single neuron. ANNs are able to solve various types of classification and continuous variable prediction (regression) tasks. ANNs are the generic term for neural networks, such as a FNN, which are used in AI. FNNs are networks without cycles and are based on the Single Layer Perceptron (2.3.1) and Mulitlayer Perceptron (2.3.3) [Sch15]. This chapter explains the basic theory they are based on. More complex ANNs, such as Recurrent Neural Networks (RNN), which are also important in NLP, are discussed in the next section. In this thesis ANNs will be used to actually predict the vehicle description texts' influence on the market price of a car. Thus, their use lies in the modelling phase of the CRISP-DM.

The McCulloch and Pitts' neuron takes vectors with length m as input and multiplies each value $x_1, x_2, \ldots, x_m$ by a corresponding weight $w_1, w_2, \ldots, w_m$. The neuron has the ability

to activate when the overall sum is higher than a given threshold θ. An activation in this particular case means that the neuron outputs a 1, otherwise a 0 [Mar15, p. 41]. This is useful for predicting binary class labels that are suitable for the rain forecast task, which was discussed above.

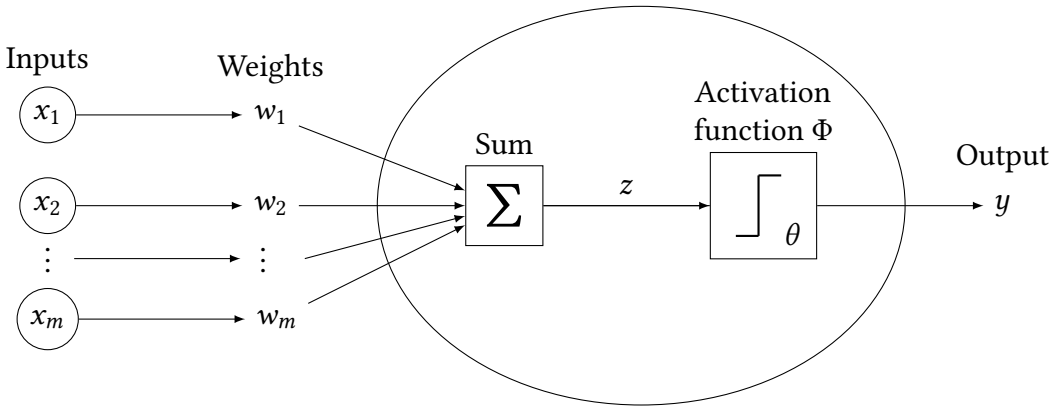


Figure 2.5.: McCulloch and Pitts' Neuron Maps m Input Values to One Output y [Mar15, p. 41].

At first the neuron calculates the logit z of the input vector $x_1, x_2, \ldots, x_m$:

$$z = \sum_{i=1}^{m} w_i x_i. \tag{2.15}$$

Then the result y is determined by the activation function Φ, which in this case is a Heaviside-Function. It takes the logit z as input and checks if it is below or above a certain threshold θ:

$$o = \Phi(z) = \begin{cases} 1 & if\ z > \theta \\ 0 & if\ z \leq \theta \end{cases}. \tag{2.16}$$

Therefore, the output y of a McCulloch and Pitts' neuron is 0 or 1.

2.3.1. Single Layer Perceptron

The Single Layer Perceptron is a collection of McCulloch and Pitts' neurons. The neurons are combined to create more complex ANNs. The model of a Single Layer Perceptron is shown in Figure 2.6, which is also known as a Single Layer Network. The network takes again m values as input. In addition, the network has more than one neuron in the output layer. Every input value is fully connected to each output neuron. Each connection is weighted to adjust the inputs received by the neurons. Therefore, the weights are stored in the form of a matrix and no longer in the form of a single vector.

For a Single Layer Perceptron only one weight matrix $W^{(1)}$ exists, which connects the m inputs to the n output nodes:

$$W^{(1)} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \\ w_{b1} & w_{b2} & \dots & w_{bn} \end{pmatrix}. \tag{2.17}$$

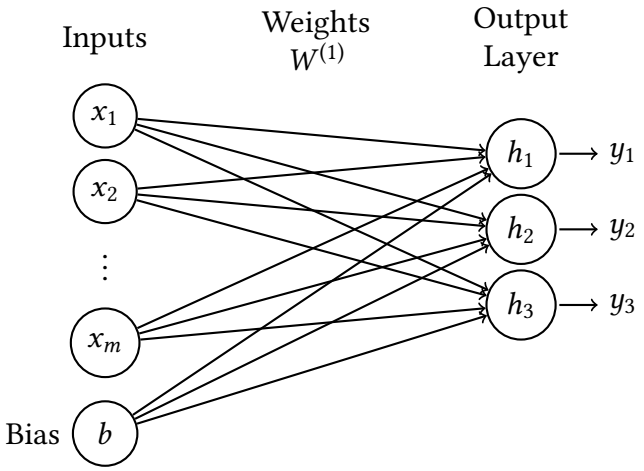


Figure 2.6.: Single Layer Neural Network with 3 outputs and m inputs [Mar15, p. 46]. It contains 3 Neurons in the output layer and a weight matrix W^1 that connects the inputs to the neurons.

The last row $w_{b1}, w_{b2}, \dots, w_{bn}$ in the matrix indicates the weights for a bias value b. The bias is handled as a separate input and is used to shift the activation of a neuron. If a neuron receives a zero for input ($x_1, x_2, \dots, x_m = 0$), it cannot be activated. To implement this capability, a bias b is added, which must not be 0. In practice this value is often set to -1 or 1. Moreover, the j-th neuron in the output layer is denoted by h_j. The weights and outputs are independent and a generalization of Equations (2.15) and (2.16) can be used to calculate the result y_j for each neuron's logit z_j in the output layer:

$$z_j = \sum_{i=1}^{m} {w_{ij}}^{(1)} x_i + {w_{bj}}^{(1)} \tag{2.18}$$

$$o_j = \Phi(z_j) \tag{2.19}$$

In addition, it is possible to use another activation function Φ to change the outputs of the neurons. Depending on the application, a certain output format is required. Moreover, the training algorithm (section 2.3.4) requires a differentiable activation function and the step function does not meet this criterion [Mar15, p. 46-47]. Different activation functions such as Sigmoid, Rectified Linear Unit (ReLU) or softmax will be introduced in the next section.

2.3.2. Activation Functions

Activation functions have an influence on the model capacity and complexity of an ANN. They can be linear or non-linear. Furthermore, their use also depends on the actual data mining problem. The usage criterion mainly applies to the last layer of an ANN that generates the overall output. If a continuous result in $\mathbb{R}$ is required, e.g. in a regression task, a function that has the same value space is more useful. In contrast, the step function and other functions that map to a value between 0 and 1 are suitable for binary classification. The softmax activation function is used for multi-class problems and uses the logits of all neurons in the same layer. It is often applied on top of the output layer. However, the hidden layers between input and output nodes may have other types of activation functions [Mar15, p. 107]. The concept of hidden layers will be described in Section 2.3.3. Figure 2.7 illustrates four distinct functions.

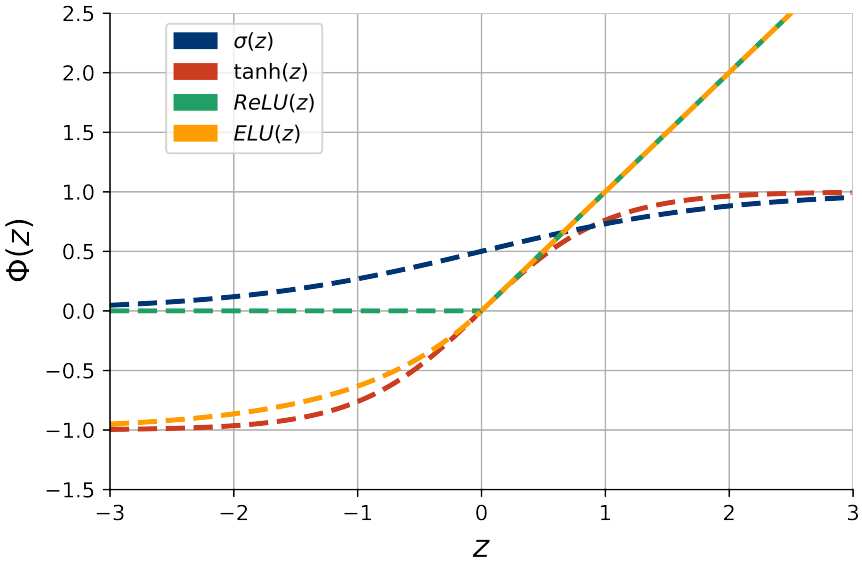


Figure 2.7.: The graph of four different activation functions illustrates their distinct output values.

The simplest activation function is the **linear** or **identity** function that returns the weighted sum z (Equation (2.18)). The output is a continuous value:

$$\Phi(z) = linear(z) = z. \tag{2.20}$$

The **sigmoid** (σ) function is similar to the step function that was defined in Equation (2.16). Sigmoid can be derived, which is an important feature for training ANNs.

$$\Phi(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.21}$$

Furthermore, sigmoid is a continuous function, tends to become 0 if $z \to -\infty$ and converges to 1 if $z \to \infty$. In practice, it is often used in classification tasks due to its output. One

drawback of sigmoid is its sensitivity to the vanishing gradient problem, which will be discussed further at the end of Section 2.4 [GBC16, p. 195]. Another widespread activation function is the **hyperbolic tangent** (tanh):

$$\Phi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.22}$$

It has a similar shape and characteristics as the sigmoid function, but converges to -1 if $z \to -\infty$ and to 1 if $z \to \infty$. In addition, it is also differentiable and vulnerable to a vanishing gradient [Sal+17].
The **Rectified Linear Unit** (ReLU) is a state-of-the-art activation function, which is now one of the most successful activation functions. Especially in Deep Neural Networks (DNNs), ReLUs lead to a faster learning time [LBH15]. Besides, it is not susceptible to a vanishing gradient and therefore often used in practice. ReLU calculates the maximum between 0 and the linear outcome of a neuron:

$$\Phi(z) = ReLU(z) = \max(0, z) \tag{2.23}$$

The Exponential Linear Unit (ELU) was introduced into Neural Networks in 2015 and outperforms ReLU [CUH15]. It can have a negative outcome, does better generalize and even learns faster than ReLU. It is defined as:

$$\Phi(z) = ELU(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases} \tag{2.24}$$

softmax determines a probability value for each class. The outcome for neuron z is dependent on the outcome of the other neurons h_i in the same layer, which makes it suitable for multi-class problems:

$$\Phi(z) = softmax(z) = \frac{e^z}{\sum_{i=1}^{N} e^{z_i}} \tag{2.25}$$

The total value adds up to 1 and all class probabilities are between 0 and 1. This list of activation functions is by far not complete and other functions, such as Sigmoid-weighted Linear Unit (SiL) or Parametric ReLU (PReLU), exist [RZL17]. Due to the fact that covering all activation functions would be too extensive for this thesis, only the most common ones were presented.

2.3.3. Multilayer Perceptron

The Single Layer Perceptron, which was discussed above, is capable of solving linear separable problems, but cannot decide nonlinear issues such as the XOR problem [MP69, p. 12-14]. To overcome this shortcoming more linear layers are added to the network to form a Multilayer Perceptron, such as shown in Figure 2.8. The first layer is called the **input layer** and has no neurons. It just receives the m input values. The **output layer** is the last layer of a Multilayer Perceptron and its neurons return the final result of the network. All intermediate layers are called **hidden layers**, because they perform the calculation within

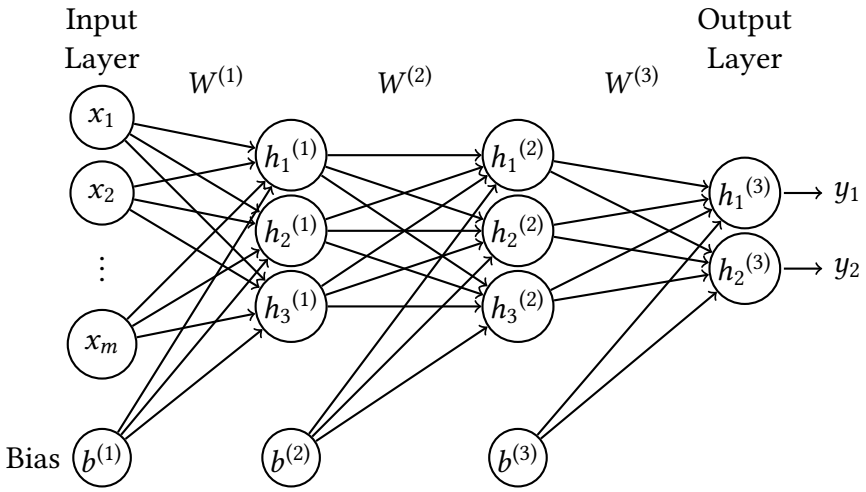


Figure 2.8.: Multilayer Perceptron with 2 hidden layers and 2 output neurons [Mar15, p. 72]. Three weight matrices connect the layers to pass the values through the network.

the network and are not visible to the network's external environment. The k-th layer of a network will be denoted with $h^{(k)}$. Every layer $h^{(k)}$ consists of a number of neurons and has its own bias $b^{(k)}$. Besides, it is connected to the previous layer with a weight matrix $W^{(k)}$ that is similar to the Matrix (2.17). By adding more layers or neurons to a layer, a deep neural network (DNN) can be created. The number of neuron layers in a network will be labeled with K.

The result of each neuron is calculated by using Equations (2.18) and (2.19). Each neuron's activation output from one layer is then forwarded as input to all neurons of the next layer. By applying this scheme in a row, the network maps the input area to the output space. Therefore, an ANN can be seen as a function $f(x, \theta)$, which predicts the outcome $\hat{y}$ of an instance $x = (x_1, \ldots, x_m)$. To shorten and simplify notation the variable $\theta = (W, b)$ is introduced, which represents the parameters of an ANN. W indicates all weight matrices of the network $(W^{(1)}, \ldots, W^{(K)})$ and b is the vector of biases $b = (b^{(1)}, \ldots, b^{(K)})$ [Mar15, p. 101].

For example, the network in Figure 2.8 could be a solution to the rain forecasting problem, which was briefly mentioned above (Section 2.2.1). Date and temperature data of the previous year are suitable features for input. One of the output neurons returns the probability for a rainy day and another for a dry day.

2.3.4. Training and Regularization

The training of an ANN is the basic part in this machine learning procedure. It puts the experience into the network. This section discusses details about the learning of a FNN. The training is similarly used for other ANN types. The section will start with the

explanation of a cost function, which leads us to an optimization criterion. Then, for a more detailed overview, the algorithm is divided into smaller parts.

2.3.4.1. Cost Function

Cost functions, denoted as J, determine the derivation of an instance's prediction $\hat{y}$ and its true value y, which is also known as error. The terms cost function, loss function and error function are often used interchangeably. Sometimes the terms are separated [GBC16, p. 82]. Instead of using the term error function, I will use cost function, loss function and the concept of an error itself, because the term error function could easily be mixed up with the Gauss Error Function. The most common cost functions are the Mean Squared Error (MSE) and the Cross Entropy Loss. The first one determines the average deviation between prediction $\hat{y} = f(x, \theta)$ and true label y:

$$J(\hat{y}, y) = MSE(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{2.26}$$

N indicates the number of neurons in the output layer. This function could be used for any problem in general. In an environment with multiple class labels another cost metric has to be considered. The Cross Entropy Loss (CE) function is calculated with:

$$J(\hat{y}, y) = CE(\hat{y}, y) = -\sum_{i=1}^{N} y_i ln(\hat{y}_i), \tag{2.27}$$

where each prediction $\hat{y}$ represents the probability for an independent class label. CE is often used in combination with softmax (Equation (2.25)), as it calculates the deviation in a multi-class situation [Mar15, p. 108]. It is important that the cost function and activation function in the output layer are compatible.

The goal of an ANN and the basic concept of the training is to reduce the cost function. Therefore, learning is an optimization problem with the following criterion:

$$\min_{\theta} J(f(x, \theta), y). \tag{2.28}$$

$\theta = (W, b)$ are the parameters of the network that can be changed to reduce the error.

2.3.4.2. Initialization Phase and Forward Pass

Before training an ANN the **initialization** phase takes place. Different approaches to initialize an ANN exist. A common method in practise is to set the weights W between the input nodes, hidden layers and the output layer to a small negative or positive random value. In addition to the weights, the bias $b^{(k)}$ for each layer k is set to a number, which is not 0. After this initial step the actual training starts.
A training step encompasses the **forward pass**, where the values of the first instance vector are passed to the input nodes. For each neuron the weighted sum is calculated with Formula (2.18) and inserted into its activation function Φ. Finally, the output is obtained

by performing a set of computations at each layer k using the results of the previous layer as input. In a network with K layers the forward pass is calculated by:

$$\begin{aligned} h^{(1)} &= \Phi^{(1)}(W^{(1)} + x + b^{(1)}) \\ h^{(2)} &= \Phi^{(2)}(W^{(2)} + h^{(1)} + b^{(2)}) \\ &\vdots \\ \hat{y} = h^{(K)} &= \Phi^{(K)}(W^{(K)} + h^{(K-1)} + b^{(K)}). \end{aligned} \tag{2.29}$$

The input x is put into $h^{(1)}$ and the result of $h^{(1)}$ is passed forward. Finally, the output layer $h^{(K)}$ emits the prediction $\hat{y}$ [GBC16, p. 197].

2.3.4.3. Backward Pass

In the **backward pass**, also known as back-propagation of error, the error is determined by a cost function. To actually update the network, function $J(f(x, \theta), y)$ needs to be differentiated with respect to θ [GBC16, p. 204]. The idea behind this process is to minimize the error by following the gradient of the cost function downwards (gradient descent), as illustrated by Plot 2.9.

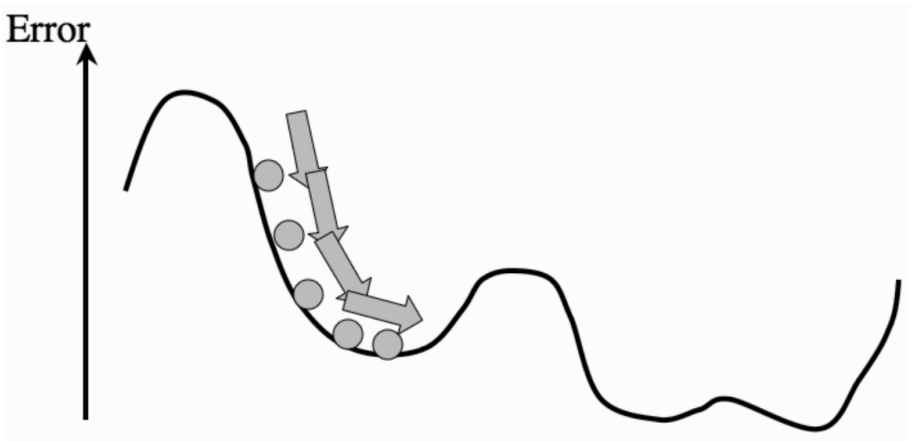


Figure 2.9.: Cost function going down the hill to find a local or global minimum [Mar15, p. 75].

The gradient of J with respect to θ is defined as:

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta}. \tag{2.30}$$

This error signal obtained by the output layer is now passed backwards through the whole network. By applying the chain rule of differentiation, the partial derivative of the cost function J at the output layer $h^{(K)}$ is calculated as:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial J(\theta)}{\partial h^{(K-1)}} \frac{\partial h^{(K-1)}}{\partial \theta} \tag{2.31}$$

$h^{(K-1)}$ is the result from the previous layer, which is passed to the output layer as input. Furthermore, $J(\hat{y}, y)$ includes the prediction of $\hat{y} = f(h^{(K-1)}, \theta)$. Equation (2.31) indicates the change of error at output level when the parameters θ are varied. Now, the chain rule can be applied backwards through the whole network. This makes it possible to compute the gradients recursively and obtain all $\frac{\partial J(\theta)}{\partial W^{(k)}}$ for the weight matrices and $\frac{\partial J(\theta)}{\partial b^{(k)}}$ for the bias vectors in θ [GBC16, p.205-209].

2.3.4.4. Gradient Descent

The gradient descent step happens after the backward pass. The parameters θ are updated with respect to the computed gradients. The weight matrices between the layers and the bias vectors are adjusted in the direction of the descending error. For a step wise adaption a learning rate η is multiplied with the gradient. This rate should be set appropriately, because a very low value would lead to slow learning and finding a local minimum only. A high learning rate could lead to an overshooting of the optimal minimum of the error. Three basic Gradient Descent manifestations exist: **Batch Gradient Descent**, **Stochastic Gradient Descent** and **Mini-Batch Gradient Descent**. The simplest one is Batch Gradient Descent, which is also known as Vanilla Gradient Descent. With this approach the weights and biases are updated by:

$$\theta = \theta - \eta \nabla_\theta J(\theta). \tag{2.32}$$

The term has to be subtracted from the actual parameters, since the error has to be reduced. Forward Pass, Backward Pass and Gradient Descent are repeated several times to improve the performance of the network [Mar15, p. 77-78]. Learning is stopped as soon as a criterion is met, e.g. if a certain number of training epochs has been completed or the training error has been reduced to a threshold value. An epoch is completed when all instances of the training dataset have been processed. Batch Gradient Descent updates the weights after processing all training instances. When learning on large data sets, it requires a lot of memory. In contrast, Stochastic Gradient Descent (SGD) updates the weights after each sample, which results in a high learning time. Therefore, the parameters are updated after each instance $x^{(i)}$ with its true label $y^{(i)}$ has been processed:

$$\theta = \theta - \eta \nabla_\theta J(\theta, x^{(i)}, y^{(i)}). \tag{2.33}$$

Due to the fact that both methods do have some drawbacks, mini-batching is often used in practice to get the best out of both [Rud16].

2.3.4.5. Mini-Batch Gradient Descent

Mini-batching aka Mini-Batch Gradient Descent improves the learning speed and saves memory. Without mini-batching, the weights of a neural network are not updated until all instances of the training data have been processed (Batch Gradient Descent) or the weights are updated after each instance (Stochastic Gradient Descent). To enable mini-batching the training data has to be split into groups of a batch size n, which is smaller than the size of the training set. In practice, batch sizes between 32 and 512 are commonly used

[Kes+16]. The weights are updated after a batch has been processed [Rud16]. The batch instances are passed into the network simultaneously:

$$\theta = \theta - \eta \nabla_\theta J(\theta, x^{(i:+n)}, y^{(i:+n)}). \tag{2.34}$$

Shuffling the data and picking these batches randomly also improves the whole learning process. Furthermore, it is necessary to use an averaged version of the cost function. This method updates the weights into the direction in which the most instances of a batch are about to be optimized. Local minima are found faster and the estimation of the gradient error is better [Mar15, p. 82].

2.3.4.6. Regularization and Early Stopping

Neural networks, in particular DNNs, have many parameters and can be applied to various tasks of machine learning. In training, ANNs easily overfit and learn the training data by heart, as discussed in Section 2.2.3.2. Regularization is an important method to prevent this problem. Mainly two methods are applied for their regularization: **Dropout** and **weight decay**

Dropout can be used at the input and output layer of a neural network. During training, it randomly drops neurons with a predefined dropout rate and their connections within the neural network. By doing this, dropout reduces the network's inner complexity and prevents overfitting on the training data. Furthermore, it improves the results in supervised learning tasks, such as document classification and speech recognition [Sri+14].

Weight decay limits the growth of the network's parameters and improves the generalization of the network. The growth of the parameters is limited by a weight decay value. The weights are chosen so that the network continues to classify the instance correctly, but does not adopt it by heart. Thus, in each step of updating the weights, all parameters are penalized with a term in the cost function:

$$J_{wd}(\theta) = J(\theta) + \frac{1}{2}\lambda \sum_{i}^{|\theta|} w_i^2. \tag{2.35}$$

Thereby, $J(\theta)$ represents a cost function like the the cross entropy loss or mean squared error. The actual weight decay factor is represented by λ and $|\theta|$ stands for the number of parameters in the neural network. In equation (2.35) one example for a simple penalty term is given. It is possible to take other forms of penalty terms [Kro92].

Early stopping is another important technique to prevent overfitting of ANNs. It stops the training phase of the network after a certain number of epochs and stops further adoption on the training data [Pre97]. It is not a regularization technique, but has the same purpose.

2.4. Recurrent Neural Networks

Recurrent neural networks (RNN) are a further development in the field of ANNs. In practice, RNNs are often used for sequential inputs. Especially in language, words have to

be processed step by step in order to retrieve their context. In the meantime, RNNs have proven to be very good at translation tasks since they are able to predict the next word by earlier words. To control the process of analyzing an instance after another, RNNs have hidden units to store state vectors. These vectors contain information about the sequence's history [LBH15]. Thus, the output $\hat{y}$ of an RNN is dependable on the actual input x_t and preceding inputs $(x_1, \ldots, x_{t-1})$. Besides, the hidden states are represented by a hidden vector sequence $h = (h_1, \ldots, h_t)$. The outputs of a RNN constitute a sequence $y = (y_1, \ldots, y_t)$, but do not necessarily represent the output for the overall task. RNNs can also be used to predict a single target for an input sequence. The hidden state h_t is calculated by:

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.36}$$

H stands for a hidden layer function, which is often the hyperbolic tangent (tanh). Furthermore, W is again the weight matrix between the layers and b represents the bias vector. The output is calculated similarly to the outcome of a Multilayer Perceptron by multiplying the weights with the result of the hidden state and adding a bias.

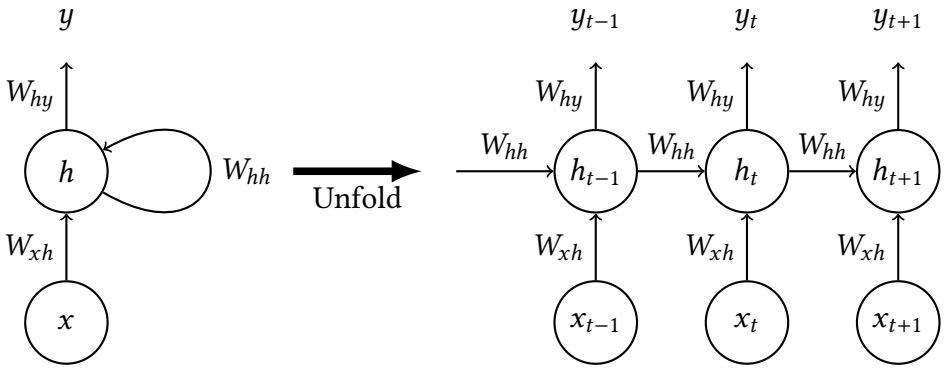


Figure 2.10.: Recurrent Neural Network predicts the outcome y of a sequence $x_1, \ldots, x_{t-1}$ [DH18]. It can be seen as a folded FNN.

Therefore, the hidden state is alike to a hidden layer [GMH13]. The outcome is calculated in a similar manner:

$$y_t = W_{hy}h_t + b_y \tag{2.37}$$

The output y_t and the hidden state h_t are connected with the weight matrix W_{hy}. This layer structure is comparable to a Single Layer Perceptron (Section 2.3.1). A RNN can be seen as a folded Multilayer Perceptron with cycles, due to the fact that the outcome of the previous input is also an input to the current instance. The model of a RNN is shown in Figure 2.10. Here, the weight matrices W are presented in general format to improve the readability of the presentation. Normally they should be adapted to the underlying scheme: $W_{h_{t-2}h_{t-1}}, W_{h_{t-1}h_t}, W_{h_th_{t+1}}, \ldots$

The training procedure is also similar to those of the Multilayer Perceptron. At first the network is initialized and the initial hidden state h_0 is usually set to 0. Then, the prediction $\hat{y}$ for the input x_t is calculated in the forward pass. Afterwards an evaluation of the true outcome y takes place. In the backward pass the error's derivative with respect to weights

is calculated using back-propagation through time (BPTT) [Sal+17].
One shortcoming of standard RNNs is the problem of **vanishing** and **exploding gradients**. The first problem makes it difficult for RNNs to keep track of the dependencies in longer input sequences. This is due to two reasons. Firstly, the hyperbolic tangent and the sigmoid function, which are often used in RNNs for activation, saturate very quickly and so their gradient gets closer to 0. Secondly, by applying BPTT the gradient is exponentially reduced by multiplying it with the recurring weight matrices. This also causes the gradient to converge to zero very fast [Sal+17]. The phenomena of the exploding gradient leads to high oscillation of the network's weights and increases learning time, which could lead to network failure [Hoc+01].

2.4.1. Long Short-Term Memory Network

Long short term memory networks (LSTMs) were proposed in 1997 and were developed to avoid the drawbacks of plain RNNs [HS97]. One basic improvement in a LSTM is the introduction of a forget mechanism. Besides saving information about the last input, they are able to forget earlier processed instances stored in their internal state. Moreover, LSTMs also have another input c_{t-1} and output c_t which is called cell state. The LSTM consists of several gates: the input gate, forget gate and output gate. The gate mechanism works similar to a water faucet since the gates regulate the values passed through the LSTM. All gates are based on sigmoidal activation functions.

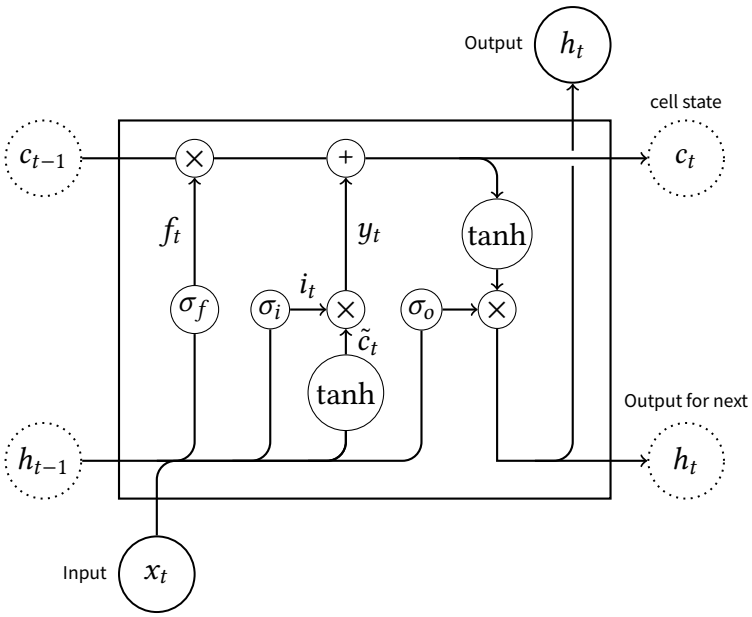


Figure 2.11.: Architecture of a recurrent cell in a Long Short-Term Memory Network (LSTM) with three different gating mechanisms σ_f, σ_i and σ_o [Bia+17, p. 12].

2.4.1.1. Forget Gate

The forget gate enables the LSTM to drop information from cell state c_t. By looking at the input x_t and the previous hidden state h_{t-1}, it calculates a value between 0 and 1 for every entry in the cell state c_{t-1}. This number indicates whether the corresponding entry should be kept (1) or deleted (0) [Ola15].

$$f_t = \sigma_f(W_f[h_{t-1}, x_t] + b_f) \tag{2.38}$$

Thereby f_t is the result of the forget gate, W_f and b_f represent the corresponding weights and biases. The output is then multiplied with the previous cell state c_{t-1}, as shown in Figure 2.11.

2.4.1.2. Input Gate

Like the forget gate, the input or update gate σ_i combines the previous internal state h_{t-1} and the current input x_t to i_t. A layer with sigmoid activation decides on the factor they should be included in the new internal state c_t:

$$i_t = \sigma_i(W_i[h_{t-1}, x_t] + b_i) \tag{2.39}$$

The actual new value $\tilde{c}_t$ is generated by a tanh layer with the same two input nodes:

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.40}$$

Both results are multiplied and added to the internal state c_{t-1} from time step $t-1$, which is already filtered by the forget gate:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{2.41}$$

Therefore, the input gate works as a regulator over the input sequence. For further calculations c_t is passed on to the next time step and serves as input for it.

2.4.1.3. Output Gate

The overall output h_t of a LSTM-cell is finally modified by the output layer gate σ_o. The cell state from equation (2.41) is taken and passed through another tanh-layer. Similar to the gates explained above, the output gate filters candidates according to h_{t-1} and x_t. Moreover, it weights the instances to be removed from the possible results:

$$o_t = \sigma_o(W_o[h_{t-1}, x_t] + b_o) \tag{2.42}$$

The calculation of the next state h_t is done by:

$$h_t = o_t * \tanh(c_t) \tag{2.43}$$

A LSTM can be applied to any sequence with the length of n. The final result is then represented by the last hidden state of the sequence h_n. In practice it could be necessary to append a linear layer of a FNN to transforms the hidden state into a desired outcome.

2.4.1.4. Bidirectional Long Short-Term Memory Network

Bidirectional RNNs were introduced in 1997 by Schuster and Paliwal [SP97]. Bidirectional LSTMs are based on them and extend the standard LSTM model. The basic idea is to process the sequence $x = x_1, \ldots, x_t$ forwards and backwards. Therefore, a bidirectional LSTM has a forward and backward part. Their hidden states $\overrightarrow{h}$ and $\overleftarrow{h}$ are connected to the output layer by a FNN. The forward part of the network processes $x_1, \ldots, x_t$ and the backward part $x_t, \ldots, x_1$ [GMH13]. Using the last hidden states the outcome y_t at the output gate is computed similarly to Equation (2.37):

$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y \tag{2.44}$$

2.4.2. Gated Recurrent Unit

The Gated Recurrent Unit GRU is another type of RNN and was introduced in 2014 by Cho et al. [Cho+14]. Similarly to the LSTM, it uses gate mechanisms to regulate the information flow within the recurrent cell [Chu+14]. Figure 2.12 illustrates the architecture of a GRU in detail.

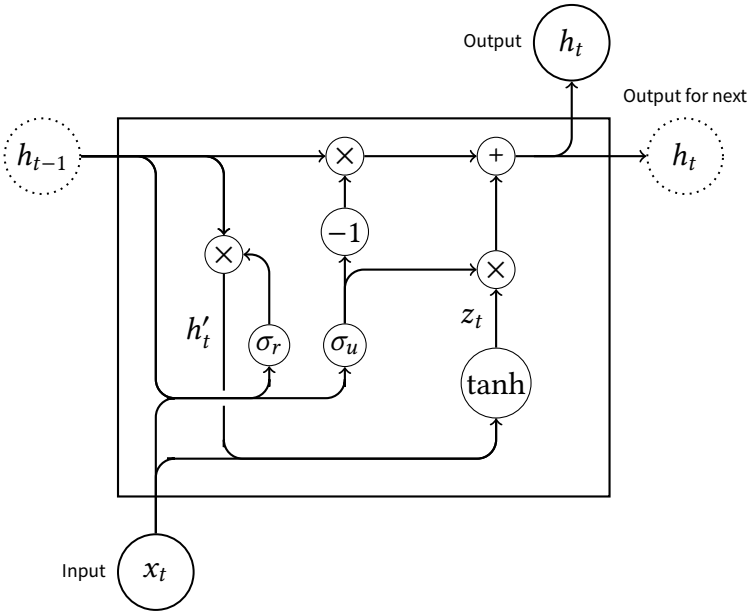


Figure 2.12.: Model of Gated Recurrent Unit (GRU) that uses two gates σ_r and σ_u to regulate the information flow within the network [Bia+17].

A GRU has an update σ_u and a reset gate σ_u. The update gate is comparable to a LSTM's input gate. It controls the influence of the current input of the new cell state by applying a sigmoid layer to it. The output u_t of the update gate is computed with Equation (2.39). For managing the internal memory of a GRU, the reset gate is used. It determines the extent to which the history of sequential input is maintained or deleted, such as the forget gate in a LSTM:

$$r_t = \sigma_r(W_r[h_{t-1}, x_t] + b_r). \tag{2.45}$$

The current state h'_t is a product of the previous state h_{t-1} and the reset gate r_t, which is passed to the tanh node to create a new candidate state:

$$z_t = \tanh(W_z[h'_t, x_t] + b_z). \tag{2.46}$$

This candidate z_t is then used to create the new state for the current GRU cell. In addition, it uses the computations of the update gate u_t and the previous hidden state h_{t-1} [Bia+17]:

$$h_t = (1 - u_t)h_{t-1} + u_t + z_t). \tag{2.47}$$

As in a LSTM, the hidden state works also as an input for the next recurrent cell. In addition, this state can also be used to predict the result of an input, e.g. by adding a fully connected layer.

2.5. Technologies

Besides describing the crucial theory, I will shortly describe the technologies that are applied in my thesis' implementation. Different Python packages supports the coding part, especially *PyTorch* and *spaCy*.

2.5.1. spaCy

spaCy is a Python open-source library for NLP tasks[1]. Its architecture is perfect for usage in production. In addition, it supplies implementations for text preprocessing, deep learning and other tasks in the domain of NLP. In this thesis it is mainly used for text preprocessing, such as tokenization, lemmatization and stop word removal. The methods are based on the german language model which was trained on Wikipedia and the TIGER Corpus[2]. Besides, *spaCy* supports more than 31 languages and does have pre-trained word vectors using the word2vec approach.
Prodigy[3] and *Thinc*[4] are two platforms that are extending *spaCy*. Prodigy is a web based tool to create fast and various amounts of training data. It starts to train a model on a small set of data and makes suggestions for new instances. The user is able to tag whether the model was correct or not. *Thinc* is *spaCy*'s deep learning library and supports fast state-of-the-art models in NLP based on the „embed, encode, attend, predict" architecture[4]. I do not use Prodigy or *Thinc* in my implementation, but I came across them while working with *spaCy* and found it necessary to mention them here.

2.5.2. PyTorch

PyTorch is a platform that supports deep learning and provides libraries to easily create ANNs[5]. It is used to build a linear Multilayer Perceptron and a Long-Short-Term-Memory

[1] https://spacy.io/usage/spacy-101
[2] https://spacy.io/models/de#de_core_news_sm
[3] https://prodi.gy/
[4] https://github.com/explosion/thinc
[5] https://pytorch.org/features

network. One of the main benefits of *PyTorch* are implemented models, functions and learning algorithms. *PyTorch* is based on so-called tensors that represent multidimensional matrices. All calculations are done by tensor operations. Besides, it has implemented support for Graphics Processing Units (GPUs), which makes it easier to train and analyze neural network models on GPUs. Due to many tensor operations that will be done during training it makes sense to use them, as they are designed to do matrix calculations.

2.5.3. Additional Python Libraries

Besides *spaCy* and *PyTorch*, other libraries are used to assist the programming part of this thesis. They are listed and briefly described in this sub-section.

2.5.3.1. SciPy

The *SciPy* ecosystem is needed when scientific computing is done in Python. It refers to many different packages such as *NumPy*, *pandas* and *matplotlib*[6]. The *SciPy* library itself offers many different routines for numerical calculation[7]. *matplotlib* will be applied in this thesis to create diagrams.

2.5.3.2. NumPy

NumPy's most used functionality is the creation of N-dimensional arrays. Plain Python does not support this fundamental data structure. It also provides implementations for linear algebra and random number generation[8]. In addition, it is used by many other Python packages such as *PyTorch* to realize the tensor data structure.

2.5.3.3. pandas

pandas is an open source library that supports data analysis for Python by providing custom data structures such as DataFrames and Series[9]. The structures support the handling when working with a large amount of data.

2.5.3.4. scikit-learn

scikit-learn is a collection of data mining and data analysis tools. For example, Random Forest, SVM and various regression models can be found in the catalogue[10]. *scikit-learn* provides in the programming part different implementations for evaluation metrics, Random Forest and tf-idf.

[6] https://matplotlib.org
[7] https://www.scipy.org/about.html
[8] http://www.numpy.org/
[9] https://pandas.pydata.org/about.html
[10] http://scikit-learn.org/stable/

2.5.3.5. NLTK

The Natural Language Toolkit (*NLTK*) is a platform that supports the development of programs that deal with human language data. It is similar to *spaCy* and also offers various preprocessing methods such as stemming, tokenization or classification[11]. For the implementation of this thesis, the Snowball-Stemmer is taken over from *NLTK*.

2.5.3.6. gensim

gensim is another language processing toolkit for Python. It is designed for working with large corpora and provides various scripts for Topic Modeling[12], such as Latent Dirichlet Allocation [ŘS10]. *gensim*'s implementation of word2vec is used in the practical part. The theory of word2vec will be described in the next chapter.

[11] https://www.nltk.org/

[12] https://radimrehurek.com/gensim/about.html

3. State-of-the-Art

„You shall know a word by the company it keeps.“

John R. Firth

This chapter gives an overview of state-of-the-art technologies and approaches in NLP. The scope is defined by the first research question: *Which state-of-the-art NLP-approaches exist to analyze user-generated product description texts with respect to their influence on the product price?* Research indicated that the analysis of product description texts alone to predict a business outcome, such as a price, is sparsely covered in literature. The work of Pryzant et al. can be compared to my approach. They are predicting sales volumes based on product description texts [PCJ17]. In addition, they also stated that „the effect of [product] descriptions alone is not studied“. Many publications that aim to predict a price influence from texts are part of Natural Language based Financial Forecasting (NLFF). This research area focuses on predicting stock market prices using NLP [XWC18]. Here, researchers analyze texts, such as social media texts [BMZ11], financial reports [Lee+14] or news texts [SC09]. Another similar research field is the analysis of user-generated product reviews that sets the focus focusing on what influences users' buying decisions [AGI11]. Both research areas differ from my approach in terms of text structure or target objectives. Thus, I will focus on more general methods that are used to predict a target for a text.
My analysis of texts to predict price residuals is similar to text classification that aims to assign a specific class to a text. In my approach, a text will be mapped to a target price residual. Therefore, advanced modeling approaches that are used in text classification will be presented later in chapter. The Attention Mechanism (Section 3.3), Hierarchical Attention Networks (Section 3.4) and Convolutional Neural Networks (Section 3.5) are state-of-the-art in predicting classes for texts.
In addition to modeling methods, features that represent the input data are also very important for any Data Mining task. Especially when dealing with unstructured data, such as natural language texts, the generation and transformation of features is crucial for achieving good results. Thus, the chapter begins with the explanation of methods like word embeddings and continues with deep contextualized word representations.

3.1. Word Embeddings

Word embeddings are a general approach to map high-dimensional word vectors, such as one-hot encoded vectors, into a low-dimensional representation. The basic idea was first

mentioned in 2001 by Bengio et al. and was motivated to overcome the curse of dimensionality [Ben+01]. Word embeddings are computed for a set of texts with a vocabulary of size N. For the purpose of illustration, let us assume, we want to create embeddings for a single document: „the best fox is running“. The sentence has five distinct words, which leads to $N = 5$. A word w_i in this vocabulary with $i = 1, \ldots, N$ is presented by a vector with a predefined dimension. If an embedding dimension (ED) of 4 is used,

$$E = \begin{pmatrix} 0.78 & 0.14 & 0.31 & 0.20 \\ 0.12 & 0.93 & 0.45 & 0.32 \\ 0.21 & 0.18 & 0.67 & 0.89 \\ 0.38 & 0.19 & 0.24 & 0.30 \\ 0.15 & 0.48 & 0.29 & 0.25 \end{pmatrix}. \tag{3.1}$$

saves the word embeddings for all $N = 5$ words in the document. E is a $N \times ED$ embedding matrix, which resembles a dictionary. Here, every word in the vocabulary is represented by one row entry. Moreover, E is the desired output of every task that aims to create word representations. The embedding

$$v_3 = (0, 0, 1, 0, 0) \times E = (0.21, 0.18, 0.67, 0.89) \tag{3.2}$$

for the third word, which is w_3=„fox“ in the illustration, is stored in the third row of E. It can be retrieved by multiplying the word's one-hot-encoding with E. For later use of the embeddings, one problem might be to handle words that are not contained in E. These are called out of vocabulary (OOV) words. Meanwhile, there are many different methods to create word embeddings. Word2Vec, GloVe and fastText are the most popular approaches and will be discussed in the below sections.

3.1.1. Word2Vec

The **word2vec** model was introduced by Mikolov et al. in 2013 [Mik+13]. The idea is to transform a word into a continuous vector, which also represents the word's local context. This approach is based on John R. Firth's famous quote: *„You shall know a word by the company it keeps“* [Chu07]. Embeddings of word2vec do not have the shortcoming of one-hot encoded word vectors, which can only recognize whether a word is exactly the same or not. Word2Vec makes these differentiations more distinctive by including a word's antecedents and successors.
The word representations include semantic and syntactic information, which is retrieved from the context words. Figure 3.1 illustrates the position of word vectors with two dimensions. The arrows indicate a mathematical distance between two words. For example, the cosine distance is a suitable function for calculating the difference between word vectors [MYZ13]. In an embedding space, it is possible that same relations between words can be represented by a similar distance and direction. For example, the English words „Man“ and „Woman“ do have a similar distance to „King“ and „Queen“. This means that the contextual

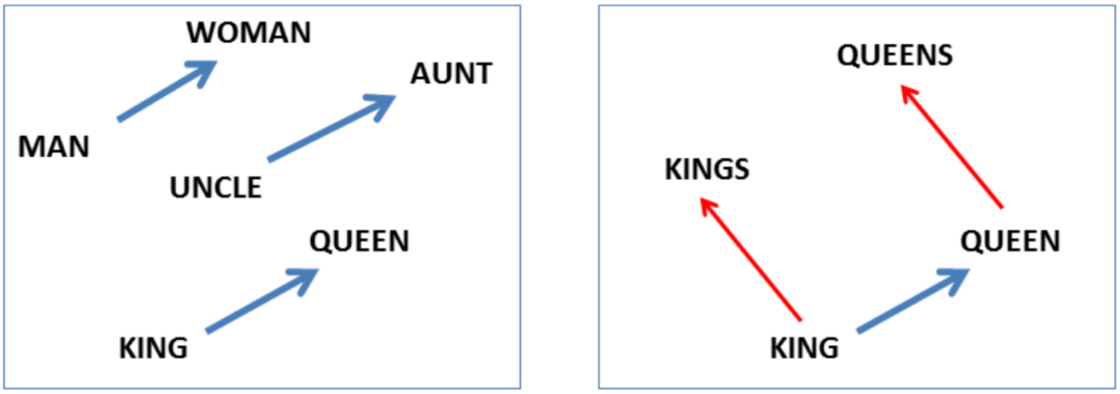


Figure 3.1.: The arrows show the distances between two word embeddings in a 2-dimensional space. Left: Distances between genders. Right: Distances between plural words [MYZ13]

difference between „Man“ and „Woman“ is comparable to „King“ and „Queen“. Words that correspond to the male gender are used in the context of „Man“ and „King“ , whereas „Woman“ and „Queen“ are surrounded by more female word forms. Thus, the syntax and semantic of a word is incorporated into a word representation. This is the real strength of word embeddings or word2vec. It makes it easier to find synonyms and extract a speaker's intended meaning more easily.

The initial paper of word2vec proposes two different approaches to learn word vectors out of text corpora with the vocabulary of N. The first approach is the Continuous Bag-of-Words (CBOW), which predicts a word based on its context. The second approach is called Skip-gram and forecasts a word's context. Both attempts minimize computational complexity and are based on FNNs, which were described in Section 2.3. Figure 3.2 depicts both architectures in more detail.
To calculate a word's vector representation, the neural network of CBOW takes the context terms to the left w_{t-2}, w_{t-1} and right w_{t+1}, w_{t+2} of the word w_t as input. This word window can be adjusted to increase or decrease the size of the local context associated with the word embedding v_t. In practice, a context window of 5 for CBOW and 10 for Skip-gram is favored[1]. CBOW takes the one-hot encoding of every input word and passes them to the network to predict the one-hot-encoding of the target word w_t. The FNN models the place of 1 in the output vector, which can be seen as a classification task.

The network that is used in Skip-gram, swaps the in- and output. The target word is now the input and the FNN predicts the word's context. Skip-gram leads to better results for small corpora, while CBOW is more efficient and recommendable for larger text sets [MLS13]. For training the word vectors; both approaches use a Multilayer Perceptron with

[1] https://code.google.com/archive/p/word2vec/

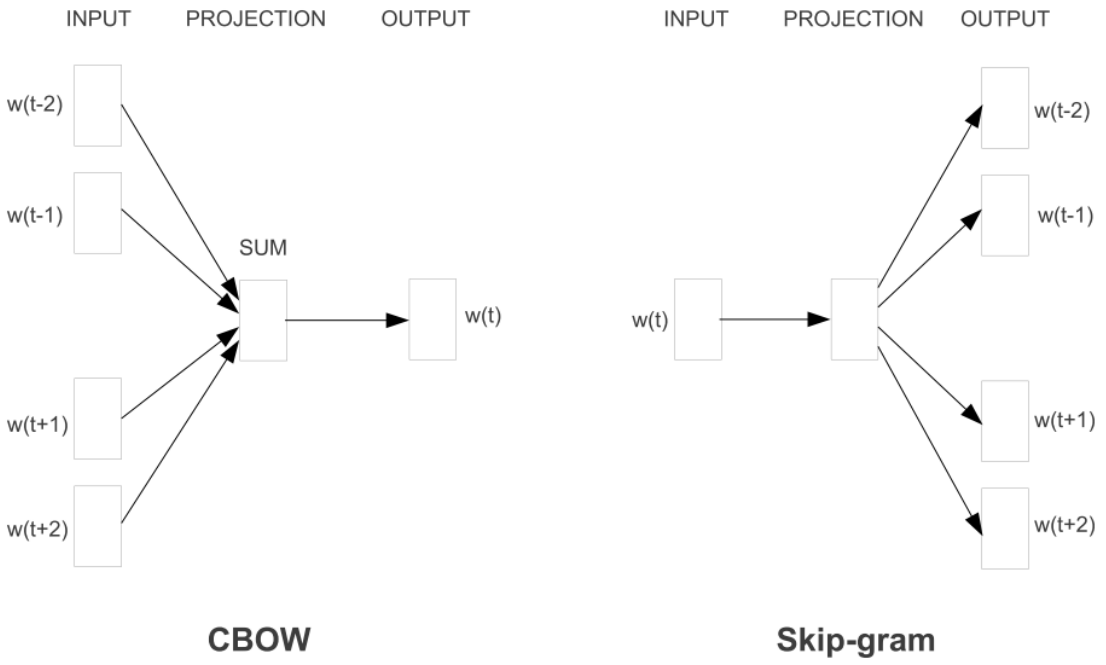


Figure 3.2.: FNN architecture of the Continuous Bag-of-Words (CBOW) and Skip-gram model [Mik+13]. CBOW uses context words to predict the target word. Skip-gram takes a word to model its context.

one hidden layer. Therefore, the network has two weight matrices. The first one $W^{(1)}$ is between the input layer and hidden layer. $W^{(2)}$ is the second matrix, which connects hidden and output layer. The number of neurons in the hidden layer is set to 300 and softmax activation in the output layer is used in both approaches [Mik+13].

To retrieve the word vector v_t for word w_t in the Skip-gram architecture, the t-th row vector from matrix $W^{(1)} = E$ is taken. CBOW takes the row vectors in $W^{(1)}$ for its input words and averages them.[2] The number of hidden neurons equals the embedding dimension, which is determined by the row length of $W^{(1)}$. CBOW models the probability $p(w_t|C)$ for the context $C = \{w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}\}$. In contrast, Skip-gram computes

$$p(w_c|w_t) = \frac{e^{v_t^T u_c}}{\sum_{i=1}^{N} e^{v_t^T u_c}} \tag{3.3}$$

for every context word $w_c \in C$ to create a word embedding v_t. The term $v_t^T u_c$ indicates how the network transforms a one-hot-encoded input word to a one-hot-encoded output word in the context. v_t is multiplied with an underlying context representation u_c for w_c that is obtained in the c-th column of matrix $W^{(2)}$. Due to the fact that N indicates the length of the total vocabulary, calculating softmax could be expensive for large N. Hierarchical softmax is used to overcome this. For further details on CBOW and Skip-gram, I recommend reading [MLS13] or [Mik+13].

3.1.2. Global Vectors for Word Representation

Global Vectors for Word Representation (GloVe) do have another approach for building word vectors. They were initially proposed in 2014 by Pennington et al. and put their focus not solely on the word's local context [PSM14]. The training of the GloVe vectors is done based on a global word-to-word co-occurrence matrix X_{ij}. This matrix stores the frequency with which two words w_i and w_j occur in the same context within a corpus.[3] Therefore, not only the local context, which is determined by the co-occurrence matrix, is taken into account, but also statistical aspects from the corpus.
First, the matrix is built up in a initialization phase in which each text is viewed once. Since the co-occurrence matrix requires lot of memory when analyzing large quantities of documents, only word combinations with at least one co-occurrence are stored.

Next, the co-occurence probabilities are calculated with $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$, as shown in Table 3.1. The table illustrates probabilities for the words „ice“ and „steam“ computed on a real corpus.
For example, the words ice and solid occur with a probability of 1.9×10^{-4}, while „steam“ and „solid“ occur less frequently. In contrast, „steam“ and „gas“ are more likely to occur than „ice“ and „gas“, which is indicated by the ratio between $P(k|ice)/P(k|steam)$. A division by zero is not possible, because matrix X_{ij} only saves co-occurence frequencies that are at least one. If both words or none of the words are related to the word k, the ratio is

[2] https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html
[3] https://nlp.stanford.edu/projects/glove/

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Table 3.1.: Example of word-word co-occurrence probabilities, which where calculated on a 6 Billion word corpus. The conditional probabilities of 4 context words k given the terms „ice“ and „steam“ [PSM14].

close to one, because both probabilities do have a similar value. This situation is shown by „water“ and „fashion“. „Ice“ and „steam“ are frequently used in the context of „water“, but are less frequent nearby „fashion“.

After computing the co-occurrence probabilities, the training step begins. The GloVe-approach learns word vectors according to the co-occurence ratio of a context word k. The word vectors for the words w_i and w_j are optimized with respect to the following equation:

$$F(v_i, v_j, \tilde{v}_k) = \frac{P_{ik}}{P_{jk}}. \tag{3.4}$$

The authors use a log-bilinear model to train the word vectors v_i and v_j. They use the dot product between the difference $v_i - v_j$ and the context vector $\tilde{v}_k$ to actually predict the co-occurence ratio. Further transformation of Equation (3.4), such as rewriting the fraction $\frac{P_{ik}}{P_{jk}}$ to a difference of logarithms, leads to a regression task that tries to minimize a cost function J. For the sake of brevity, this reshaping and the theory of a log-bilinear model will not be part of this thesis. They are described in more detail in [PSM14].

$$J = \sum_{i,k=1}^{N} f(X_{ik}) \times (v_i^T \tilde{v}_k + b_i + \tilde{b}_k - log(X_{ik}))^2 \tag{3.5}$$

depends on the vector v_i, the context vector $\tilde{v}_k$, their biases b_i, b_k, the number of the total vocabulary N and a weighting function $f(X_{ik})$. Function $f(X_{ik})$ is used to avoid the overweighting of rare and frequent co-occurences. After training it is possible to retrieve the word vector v_i for every word in the corpus.
The authors state that GloVe outperforms word2vec in different NLP exercises, such as word analogy, named entity recognition or word similarity tasks. This is due to the fact that GloVe also includes statistical information about the words in the corpus [PSM14].

3.1.3. fastText

fastText is a library for creating word embeddings and was developed by the Facebook AI research group.[4] fastText learns word representations based on subword information. It is

[4] https://fasttext.cc/

comparable to word2vec, but uses character *n*-grams as inputs and not only the word itself. This improves the ability to produce vectors even for made up or rare words. Furthermore, this method is suitable for languages that have a lot of inflections and are built upon the same small word parts, such as Finnish or Turkish [Boj+16]. The authors use CBOW or Skip-gram that were discussed in Section 3.1.1. For the sake of brevity, the subword modeling of fastText is explained using the Skip-gram approach only.

In word2vec, Skip-gram predicts the vector of context words (output) given the actual word w (input) by using one-hot-encoded word vectors. These representations do not share any common features between words, because they can only indicate if a word is the same or not. In contrast, fastText splits its input words into sets of character *n*-grams and passes them into the network. Character *n*-grams can be shared between different words. For example, the word „research“ is divided into <*re*, *res*, *ese*, *sea*, *ear*, *arc*, *rch*, *ch*> if *n* equals 3. Thereby, <*se* and *de*> contain the special characters „<“ and „>“ to indicate the start and end of a word. It is done to illustrate a difference between words that are contained as a character *n*-gram in another word, such as „*sea*“ in „*research*“.

For obtaining the word vector for a word w_t in a given set of documents, a scoring function $s(w_t, w_c)$ is introduced. It calculates the sum of the characters *n*-grams multiplied by a surrounding word $w_c \in \{\dots, w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}, \dots\}$ of w_t. The number of the context words of w_t is set by the length of Skip-gram's output window. fastText's Skip-gram version calculates the conditional probability

$$p(w_c|w_t) = \frac{e^{s(w_t,w_c)}}{\sum_{i=1}^{N} e^{s(w_t,w_c)}} \tag{3.6}$$

by replacing the scalar product between $v_t^T u_c$ with $s(w_t, w_c)$. Let G be a dictionary of *n*-grams and $G_t \subset \{1, \dots, G\}$ the set of character *n*-grams for the word w_t. Then the score

$$s(w_t, w_c) = \sum_{g \in G_t} z_g^T c_c \tag{3.7}$$

is determined for every context word w_c. Instead of a one-hot-encoded word vector, a set of character *n*-grams predicts a context word in fastText. All vectors z_g with $g \in G_t$ are character *n*-gram embeddings of the word w_t. Their set builds the word embedding v_t. Due to the fact that the overall set of *n*-grams for a large corpus is very big, a hashed version of the character *n*-grams is used as input [Boj+16]. An advantage of fastText is the performance. The training time of fastText's word representations is short and it is even possible to train the vectors on a standard multi-core CPU in a reasonable time. Moreover, the accuracy of fastText in NLP tasks, such as sentiment analysis or tag prediction for captions, is comparable to other state-of-the-art methods [Boj+15].

3.2. Deep Contextualized Word Representations

Word embeddings, as shown above, are a state-of-the-art representation of words in a numeric format. However, due to their shallow network structures, they are only able to

include the nearby local context, semantic and syntax of a word. For example, CBOW or skip-gram incorporate 5 to 10 context words in practice that influence a word embedding. Therefore, they lack to project all global connections of words [Pet+18]. The same words are often used across many documents in different linguistic contexts, e.g Polysemy. In 2018, Peter et al. developed a „deep contextualized word representation“, which overcomes this drawback [Pet+18]. These word representations are created with a deep bidirectional Language Model (biLM). Furthermore, the model is pretrained on a large text corpus.

First of all, I will shortly introduce the idea of Language Models (LMs). For a set of N words $(w_1, w_2, \ldots, w_N)$, a LM calculates the probability of the whole sequence. It uses the conditional probability of the next word w_i given the history of previous words $(w_1, \ldots, w_{i-1})$:

$$p(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} p(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{3.8}$$

Peters et al. use this underlying structure to build word representations. At first a word embedding v_i, such as shown in Section 3.1, is passed into a LSTM with K layers, that processes the sequence in forward direction. Every Layer $k = 1, \ldots, K$ outputs a context-dependent word representation $\overrightarrow{h}_{i,k}$ for the word w_i. The top layer in the LSTM is trained to predict the next word w_{i+1} and uses a FNN with a softmax layer on top to compute the output. Another LSTM processes the sequence in reverse direction, as in a bidirectional LSTM. It predicts the previous word w_{i-1}. To combine both directions into a biLM, they maximize their joint log-likelihood as optimization criterion:

$$\begin{aligned} &\sum_{i=1}^{N} (logp(w_i | w_1, w_2, \ldots, w_{i-1}; \theta_x, \overrightarrow{\theta}_{LSTM}, \theta_s) \\ &+ logp(w_i | w_{i+1}, w_{i+2}, \ldots, w_N; \theta_x, \overleftarrow{\theta}_{LSTM}, \theta_s)) \end{aligned} \tag{3.9}$$

θ_x represents the parameters for the input word embeddings E and θ_s mark the parameters for the softmax-FNN. The LSTMs have separate parameters for forward and backward direction. The softmax layer takes the LSTM states to predict the word w_i given the input sequence.
Peters et al. call their approach Embeddings from Language Models (ELMo). It combines the input word representation v_i and the hidden states $\overrightarrow{h}_{i,k}$, $\overleftarrow{h}_{i,k}$ from the bidirectional LSTM. $\overrightarrow{h}_{i,k}$, $\overleftarrow{h}_{i,k}$ are concatenated to $h_{i,k} = [\overrightarrow{h}_{i,k}; \overleftarrow{h}_{i,k}]$. Thus, their K-layered biLM calculates a set V_i of $2K + 1$ representations for every word w_i:

$$\begin{aligned} V_i &= \{v_i, \overrightarrow{h}_{i,k} \overleftarrow{h}_{i,k} | k = 1, \ldots, K\} \\ &= \{h_{i,k} | k = 0, \ldots, K\}. \end{aligned} \tag{3.10}$$

The bidirectional structure of ELMo is shown in Figure 3.3. Here, the underlying word representations in V_i are denoted with $h_{1,2}, h_{2,2}, \ldots, h_{N,2}$ for the second bidirectional LSTM-layer.

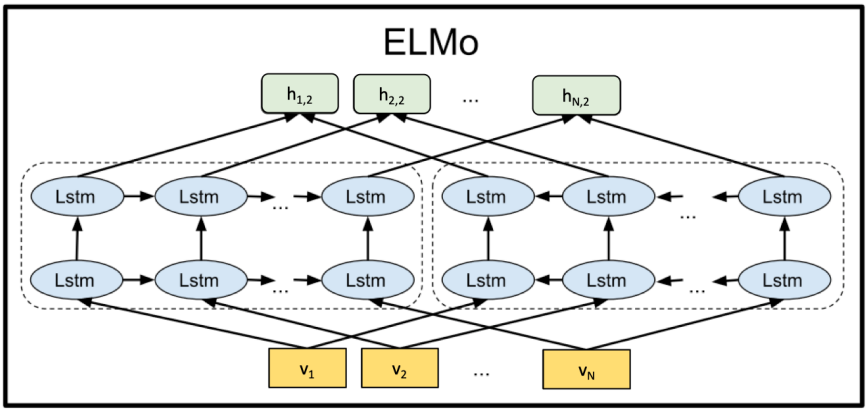


Figure 3.3.: ELMo uses a 2-layered Bidirectional Language Model as architecture [Dev+18]. A sequence of embeddings $v_1, v_2, \ldots, v_N$ is passed into the bidirectional LSTM to retrieve their concatenated word representations $h_{1,2}, h_{2,2}, \ldots, h_{N,2}$ that are computed by the second LSTM-layer.

ELMo uses these representations and optimizes them with respect to a given task, e.g. document classification. The word vector

$$v_i^{task} = \gamma^{task} \sum_{k=0}^{K} s_k^{task} h_{i,k}. \tag{3.11}$$

is a linear combination of all word representations for word w_i that is optimized with parameters s^{task} and γ^{task}. The different representations in V_i are weighted by softmax-normalized weights s^{task}. Besides, all ELMo vectors are scaled by γ^{task}. Peters et al. pretrain the representations in V_i on a large corpus. Afterwards, they learn s^{task} and γ^{task} for every particular task. When classifying documents with a RNN, for example, the previously trained representations v_i^{task} are passed on as input to the RNN. During the RNNs learning phase, the task-specific weights (γ^{task}, s^{task}) are adjusted. The simplest representation of a task-specific word vector, which ELMo can create, is the input embedding itself v_i or the the representation $h_{i,K}$ from the last bidirectional LSTM-layer.
Their approach outperforms current state-of-the-art methods in different NLP tasks, such as question answering, named entity recognition and sentiment analysis. Using ELMo the task-specific error reduces by $0.7\% - 4.7\%$ [Pet+18].

Recent research in developing word representations is mainly based on deep learning architectures. Radford et al. [Rad+18] and Devlin et al. [Dev+18] successfully trained a transformer-encoder system to create representations of words. Both approaches do also achieve state-of-the-art results in tasks, such as question answering or sentiment analysis. Devlin et al. call their approach BERT which is an acronym for Bidirectional Encoder Representations from Transformers. BERT pretrains meaningful word representations on large corpora and combines the approaches of Radford et al. and ELMo.

3.3. Attention

Another state-of-the-art approach in NLP is the use of attention and was first applied in machine translation in 2014 by Bahdanau et al. [BCB14]. They use a bidirectional sequence to sequence LSTM to translate texts from one language into another. The attention mechanism is close to human's attention. Attention helps to shift the focus to the important parts of an information sequence. The intensity and position at which attention should be directed depends on the experience and the task at hand. Besides Bahdanau et al., other attention mechanisms have been introduced by Luong et al. [LPM15]. The authors propose a global and a local attention system. The global approach is similar to Bahdanau's model and includes all instances from a sequence. The local attention focuses only on the nearby context. In addition, attention is also used in image caption generation and helps to name objects seen in an image [Xu+15]. In [PCJ17] the authors propose an approach for sales volume prediction based on product description texts. Their best solution is a LSTM with Bahdanau-Attention, which outperforms other methods like Lasso Regression or Mutual Information. Thus, their approach using Bahdanau-Attention will be discussed further.

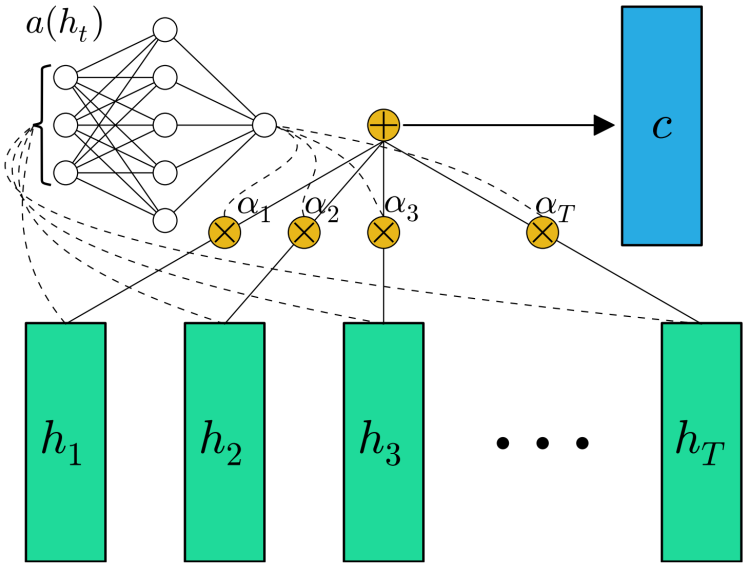


Figure 3.4.: Bahdanau Self-Attention applied to a RNN [RE15]. A FNN $a(h_t)$ computes the attention weights $a_1, \ldots, a_T$ to retrieve an averaged context state c.

The general attention mechanism is shown in Figure 3.4. A RNN, such as a LSTM or a GRU, is emitting its hidden states h_t for every time step $t \in T$ and passes them to a fully connected Feed-Forward Neural Network (FNN). The network itself calculates an attention weight a_t for every hidden state h_t, as shown in equation (3.12). The authors of [PCJ17] use tanh as activation.

$$\hat{a}_t = a(h_t) = \tanh(W_a h_t) \tag{3.12}$$

In order to obtain a distribution for the attention weights a that add up to one, a $softmax$ function is applied to the results of the FNN.

$$a = softmax(\hat{a}) \tag{3.13}$$

Afterwards, the weights are multiplied with their corresponding hidden states. These products are summed up to get an adjusted hidden state called context state c.

$$c = \sum_{t}^{T} a_t h_t \tag{3.14}$$

State c serves for further calculations and has the same dimensions as any hidden state from the RNN. It is an averaged transformation of the whole RNN output, which includes the weighted influence of each hidden state in the sequence. Usually attention is applied for machine translation tasks that have an underlying sequence-to-sequence structure in which a sequential input is mapped to a sequential output. As shown in [RE15] or [PCJ17], attention is also applied to sequence-to-one or many-to-one tasks, which predict a single-valued output for a sequence or a set of instances. Therefore, the attention mechanism is implemented and analyzed in the practical part of this thesis.

3.4. Hierarchical Attention Networks

Hierarchical Attention Networks (HANs) were developed by Yang et al. [Yan+16] and use the attention principle. They are applied on text classification and have two attention mechanisms. One puts the attention to the sentence-level and the other one focuses on the word-level. The concentration is only set to important sentences with meaningful words. This draws the attention to the way in which documents are usually created. Words build up sentences and sentences construct documents.
A document consists of L sentences s_i and each sentence s_i contains a list of words T_i. The t-th word in sentence s_i is denoted with w_{ti}. The architecture of a HAN is illustrated in Figure 3.5. It has two bidirectional GRU layers with separate attention. The first layer operates on the word-level and receives word embeddings for every word w_{ti} in sentence s_i. One GRU processes the words in forward direction and the other in reverse direction. The hidden states from the forward $\overrightarrow{h}_{it}$ and backward $\overleftarrow{h}_{it}$ GRU are concatenated to encode the word w_{ti} with h_{it}. Afterwards, an attention layer calculates the attention weights u_w in the sentence s_i for every word w_{ti}. Formulas (3.12), (3.13) and (3.14) are used to compute an overall representation for sentence s_i.
The second bidirectional GRU-layer performs similar calculations, but uses the previously generated sentence representations as input. A separate attention-layer determines the sentence-specific weights u_s that set the importance for each sentence in the document. Lastly, a softmax-layer is used to predict the document's class. Yang et al. show that the Hierarchical Attention Network outperforms other widely used approaches, such as LSTMs and Convolutional Neural Networks (CNNs), in six different document classification tasks. For instance, they have improved the accuracy of classifying Internet Movie Database (IMDb) reviews, which give a movie a rating between 1 and 10, by 4.1% compared to previous state-of-the-art results.

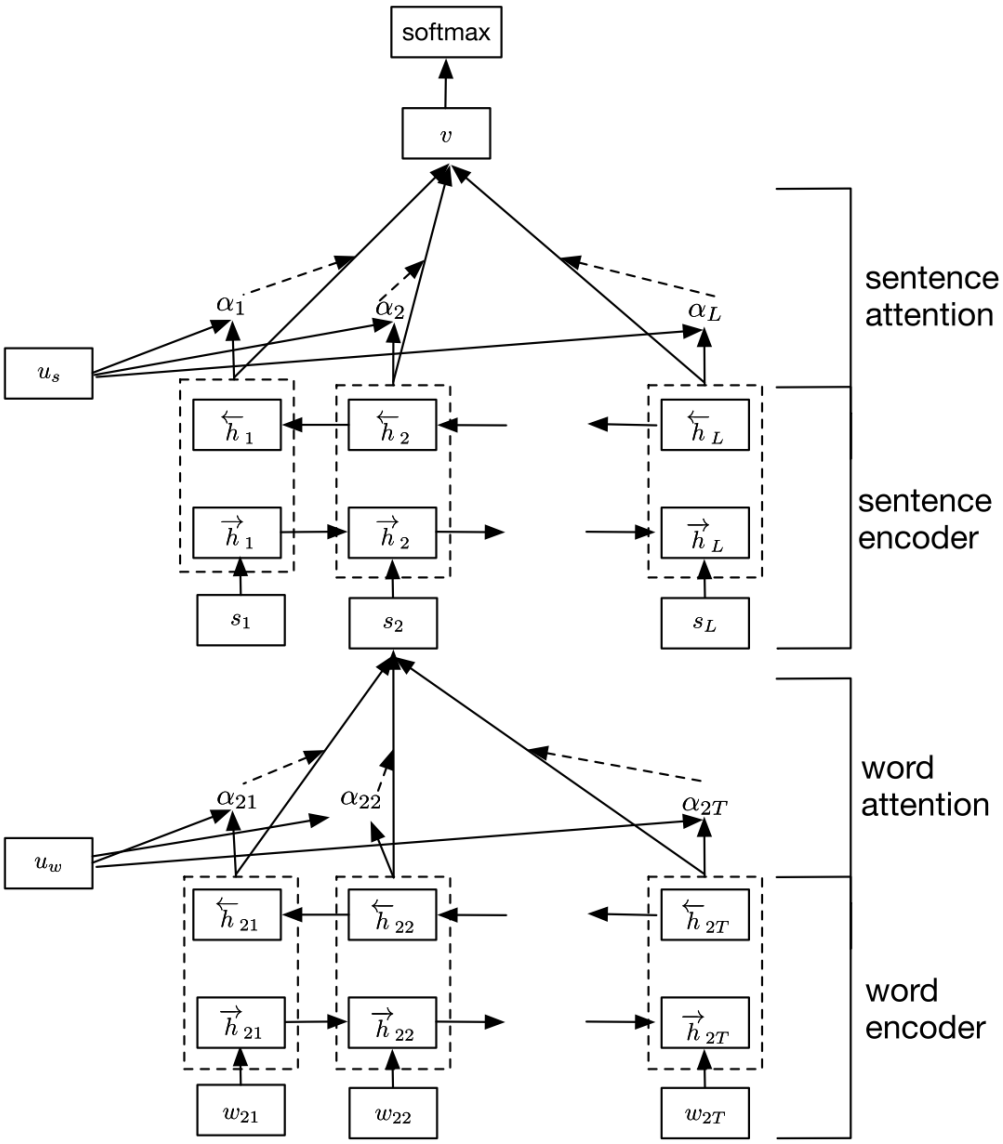


Figure 3.5.: Architecture of a Hierarchical Attention Network [Yan+16]. One bidirectional RNN calculates the attention for words and another computes the attention for a sentence representation on top. A FNN with a sofmtax layer on top is used to classify the weighted sentence vector.

3.5. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were originally developed for computer vision, e.g. for image analysis. Since 2011 they have become more and more popular in NLP [Col+11]. In 2014 Kim [Kim14] and Kalchbrenner et al. [KGB14] successfully used CNNs in NLP tasks such as sentence classification and sentence modeling. The work of Kim's sentence classification will be discussed further in this section, because it is a suitable approach to analyze texts with respect to price prediction. At first, a short explanation of the basic idea behind a CNN will be given.
A CNN consists of convolutional and pooling layers. The input data is passed into a stringing of one or more convolutional layers, which are followed by a pooling layer. This convolutional-pooling layer structure can be repeated to create deeper configurations that form a deep CNNs. A convolutional layer applies a filter pattern to the input data. This helps to map the input data to an underlying feature map and to find local connections between them. The filter pattern reduces the number of weights in the network by mapping parts of the input to one feature. Furthermore, the pooling layer shrinks the feature map and merges semantically similar features together [LBH15]. A convolutional layer can consist of more than one filter pattern, which is shown by the stacked structure in Figure 3.6.

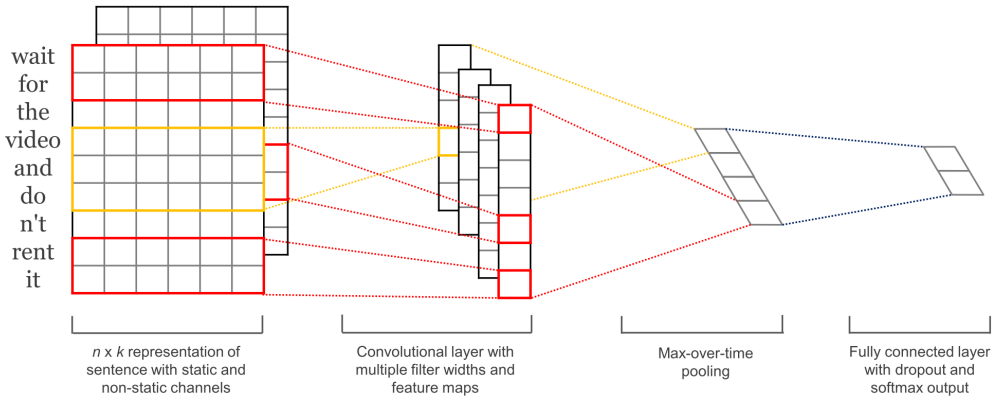


Figure 3.6.: Model of a Convolutional Neural Network that is used in sentence classification [Kim14]. The sentences are passed into the network and get filtered by convolutional layers. After pooling, a FNN with a softmax layer on top predicts the class for the sentence.

The authors of [Kim14] apply different types of CNNs to various sentence classification and question classification tasks, such as detecting positive or negative movie reviews. Their approach outperforms the state-of-the-art procedure in four tasks. They take a pretrained word2vec word embedding for every word in the sentence and construct a matrix with the dimension of $n \times k$. Hereby, $k = ED$ represents the dimension of the word embedding and n denotes the sentence's length. This matrix serves as input for their CNN. They use the hyperbolic tangent as filter in the convolutional layer. A max pooling layer is then applied to the filtered feature map. Their model incorporates a FNN with dropout

used as last layer to classify the input sentence. Modifications of Kim's approach were successfully applied to predict classes in biological [RK15] or medical [Hug+17] science.

3.6. Conclusion on State-of-the-Art

This chapter provided an answer to the first research question: *Which state-of-the-art NLP-approaches exist to analyze user-generated product description texts with respect to their influence on the product price?* by focusing on word representations and text classification methods. I showed that state-of-the-art approaches in NLP are based on meaningful word vectors that are passed into a neural network architecture to predict the target variable. The variability of the output layer in a neural network makes it possible to change the prediction of a class into a prediction of a price or price residual (regression).
Four different approaches were presented in Section 3.1 and 3.2 to show the state-of-the-art in building word representations. They are build upon FNNs or deep bidirectional architectures that provide promising results in current NLP research. Section 3.1.1 illustrated the advantage of word embeddings over the vector space model or tf-idf, which were presented in the previous chapter. Contextual word representations make it possible to display mathematical relations between words based on their semantic and syntax.
Advanced Artificial Neural Networks for improving the performance in text classification were presented in the second half of the chapter. The attention mechanism enables RNNs to put their focus on essential parts of sequential inputs, such as a set of words or sentences. Besides the recurrent architectures, CNNs were also presented shortly to extend the review of state-of-the-art methods in text classification tasks.

4. Approach

„Prediction is very difficult, especially if it's about the future."

Niels Bohr

The following chapter presents my approach for the analysis of vehicle description texts from an online car market in which sellers offer new and used cars. I improve an existing pricing model by incorporating the user-generated description texts and applying NLP-based approaches explained in the previous chapters. This and the next chapter focus on the second research question: *How do such approaches perform on real description text data for vehicles in an online car market?*
My approach splits into the steps Business Understanding, Data Understanding, Data Preparation and Modeling, which refer to the CRISP-DM that was explained in Section 2.1. In the phase of Business Understanding, I will define the objective variable, which will be predicted to see whether it is possible to improve the price forecast for cars by using descriptive texts. Then, an analysis of the dataset, which is used in my research, is performed in Section 4.2. The next step is Data Preparation that includes techniques, such as tokenization, lemmatization and stop word removal. Finally, I will give an overview of the predictive models used to forecast the target variable. The evaluation of the whole approach will be discussed in Chapter 5.

4.1. Business Understanding

As mentioned in the introduction, online car markets, such as *mobile.de* [Mob17], *pkw.de* [PKW], *cars.com* [Car] and *autotrader.com* [Aut], predict market prices based on vehicles' structured attributes. For example, the price forecast is based on the vehicle's year of first registration, mileage, color, make of car, model or performance. This helps every prospective buyer to retrieve a neutral information about the value of a car. In addition to the technical facts a car has an individual condition, which does also have an influence on its value. I generally assume that more details about a vehicle's individual condition and therefore also about a car's price are contained in the description texts of the automobiles provided. This provides the opportunity to analyze the texts with respect to their monetary value.
The following examples support this expectation: The price models from online car markets are based on structured attributes, but the additional information that the sellers provide in their user generated description texts is not analyzed with respect to their influence on the car prices. In these texts sellers can tell more detailed information about bumps, scratches

or rust. Moreover, facts about the replacement of car parts such as engine, clutch or axles can also be covered in the text format. Thus, analyzing the vehicle description texts should improve price prediction. Since this hypothesis is very important for further analysis, I will make a human-based evaluation of the vehicle description texts in Section 4.2.1. This analysis provides a baseline for my approach and checks whether price information is contained in the texts.
However, if the assumption above is correct, it is possible to improve the price forecast for cars through their descriptive texts. Figure 4.1 illustrates a schematic example of my idea to predict a price deviation by a user-generated description text. My approach improves the price prediction by forecasting a price difference between a predicted market price and a selling price.

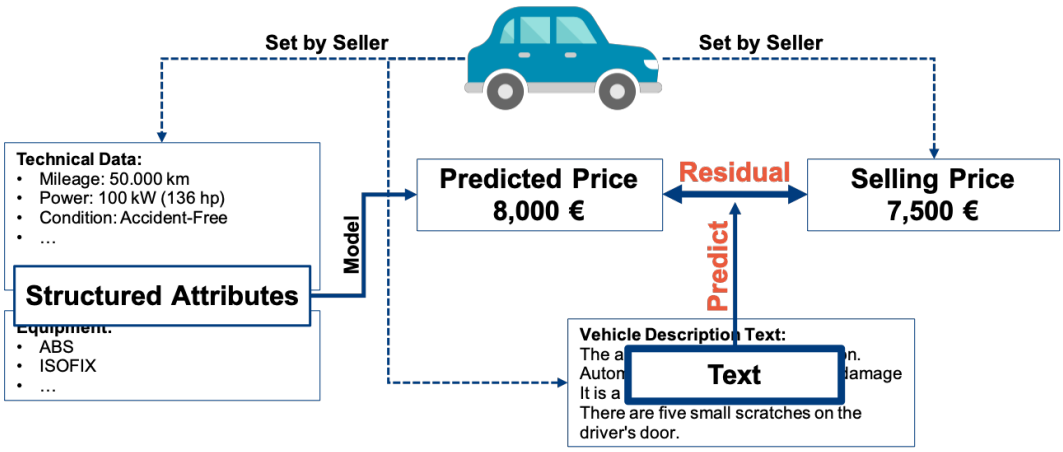


Figure 4.1.: A car has structured attributes, which are used to predict a market price. The seller of the car sets a price at which the car should be sold. The basic idea of my approach is the prediction of the price deviation between the predicted market price and the selling price by using the individual description text written by the seller. In this example, the price residual to be explained by a text is 500 €. (Car icon was retrieved from [Fla].)

Therefore, to test whether an improvement of the existing model can be achieved, the target variable is set to the price residual between the predicted price $price_{pred}$ according to the structured attributes and the actual sales price $price_{sold}$. Moreover, the whole difference is relativized by $price_{sold}$ in order to get a better comparison between high and low priced vehicles. Formula (4.1) shows the calculation of the objective criterion called the **relative price residual** (PR_{rel}):

$$PR_{rel} = \frac{price_{pred} - price_{sold}}{price_{sold}}. \tag{4.1}$$

For example, a negative PR_{rel} indicates an underprediction of $price_{pred}$. The price model trained on the structured attributes therefore suggests a lower price, compared to the price at which the car was sold. I assume that the text justifies this higher price by presenting the automobile in a positive way than its structured attributes do. The opposite holds for

a positive PR_{rel}. I set the **business objective** to the prediction of relative price residuals for vehicles by using their textual product descriptions as input. The whole pipeline for my approach is illustrated in Figure 4.2.

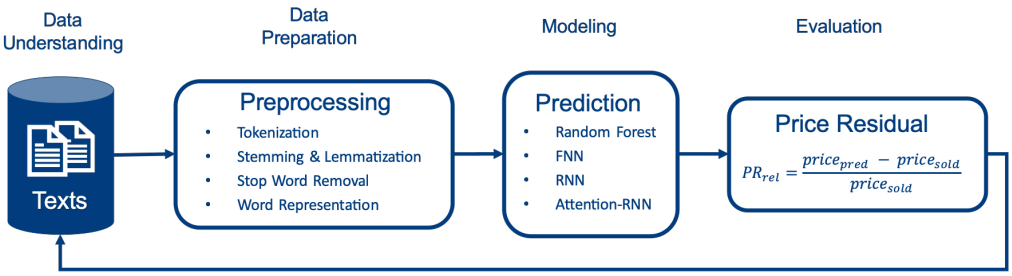


Figure 4.2.: The figure shows my processing pipeline for predicting price residuals, which is adapted to the CRISP-DM. It splits into Data Understanding, Data Preparation, Modeling and Evaluation. The cyclic structure indicates that the whole process has been executed iteratively in this thesis.

My approach starts with the extraction and preparation of the vehicle description texts from an online car market. The extraction of these texts is constrained by a couple of factors. Only cars with a vehicle description text are taken from the marketplace. Furthermore, it is important that the offers were already removed from the website, as it is assumed that the car was sold at the price specified by the seller. I do not know the accurate selling price, but the last price offered on the website is the closest approximation. In addition, I focus on listings from private car sales only, presuming that private sellers write texts that are more natural and personalized. Texts from professional dealers mostly repeat the structured attributes in the texts, which does not lead to a real improvement in price prediction.
After extracting the texts, they are put into a preprocessing phase that finally creates word representations, which are further passed into a prediction model. This model forecasts the corresponding PR_{rel} for the texts. In this thesis, different approaches and models are evaluated against each other, leading to multiple iterations of the processing pipeline. Chapter 5 focuses on the evaluation step that discusses results and performance of my proposed predictive models.

To facilitate the task and get a better judgment on whether the objective criterion PR_{rel} can be predicted, I turn the task into a classification problem. The result of a classification task can easily be evaluated by prediction accuracy, which allows simple interpretation. It would be difficult to judge a regression task for which there are no reference results. Therefore, the entire input text set is divided into three classes according to the relative price residual PR_{rel}, resulting in an underpredicted, neutral and overpredicted class. In order to achieve a balanced outcome, the number of vehicles in the classes are evenly distributed, i.e. each class holds 33.3% of the data, as illustrated by Figure 4.3. The business objective hereby is the prediction of the correct class for a vehicle description text. With this transformation it is possible to calculate the prediction accuracy, which makes the

whole results more comparable. As baseline for later comparison, we notice that a random classifier would achieve an accuracy of 33.3%.

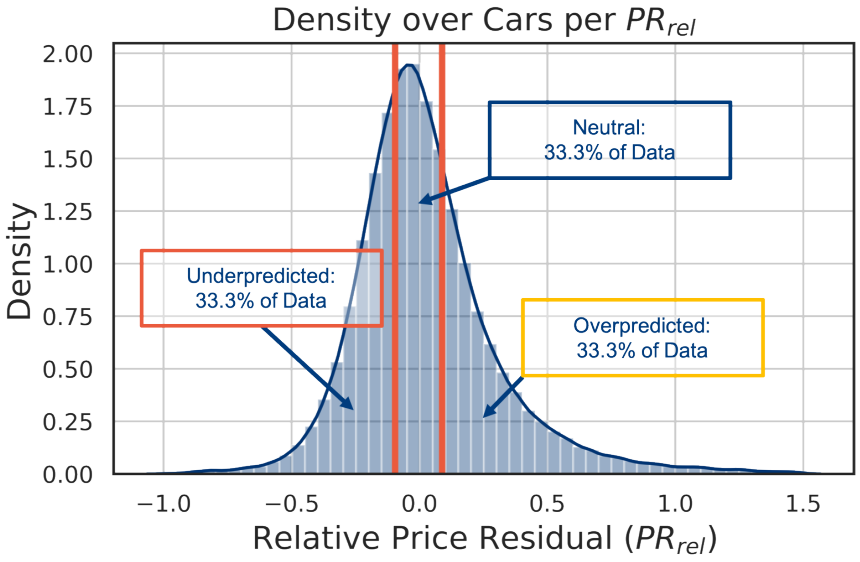


Figure 4.3.: Every class underpredicted, neutral and overpredicted includes one third of the cars texts. The plot shows a density distribution of the cars over PR_{rel}. The red vertical lines illustrate the class boundaries.

4.2. Data Understanding

The following section provides information about the data that is used in my research. This data stems from an online vehicle market that provides product description texts in German. Two different datasets are extracted to generalize the results obtained. The datasets will allow a comparison between newer and older vehicles, as I assume that the prices for older cars depend more on their individual condition that is covered by the vehicle description texts. Thus, my approach should especially improve price prediction for older automobiles. I extract three features for every car: **predicted price (**$price_{pred}$**)**, **the selling (**$price_{sold}$**)** and the **vehicle description text**. For further processing and analysis, I transform all gathered car instances into a *pandas* DataFrame, which is a data structure that was explained in Subsection 2.5.3.3. Table 4.1 shows more details about the datasets „new cars“ and „old cars“. Using Formula (4.1), I calculate the relative price residual for each vehicle based on $price_{pred}$ and $price_{sold}$. Every PR_{rel} has got a value between -1 and ∞. The difference between $price_{pred}$ and $price_{sold}$ has got a minimum value of -$price_{sold}$, because $price_{pred}$ is at least zero. $price_{sold}$ can be set to a very low value, which would lead to low denominator and to a high PR_{rel}. After computing PR_{rel}, I assign all instances of a dataset to one of the classes: underpredicted, neutral and overpredicted.

Metadata:	**1: New Cars**	**2: Old Cars**
Age of Cars:		
Oldest Car registered in:	07-2009	01-2000
Youngest Car registered in:	09-2018	06-2009
Number of Cars:	61,943	112,348
Vocabulary (words):	38,965	56,347
Average Text Length (words):	18.2	20.3
Median of Text Length (words):	10	11
$price_{pred}$ **predicted on:**		
Full Feature Model:	✓	✓
$RMSE(price_{pred}, price_{sold})$:	3,516.03	4,792.91
$MAD(PR_{rel})$:	0.182	0.355
Reduced Feature Model:	✓	✗
$RMSE(price_{pred}, price_{sold})$:	4,233.39	-
$MAD(PR_{rel})$:	0.243	-

Table 4.1.: This table provides details about two datasets that were used for my research. The first dataset contains newer cars and the second focuses on older cars. For the newer vehicles, there is a Full Feature Model that predicts the price of $price_{pred}$ on all structured attributes and a Reduced Feature Model that was trained on a smaller number of structured attributes. The Mean Absolute Deviation (MAD) and the RMSE indicate that price prediction based on a all structured attributes is less accurate for older cars.

The first dataset contains newer cars that were first registered in 2009 or later. Histogram A.1 can be found in the appendix and gives an insight into the number of vehicles per year of first registration. Most of the cars in this dataset were licensed between 2010 and 2012.
In addition, two predicted prices for the first dataset are computed by two different price prediction models. A Full Feature Model is trained on all structured attributes, while a Reduced Feature Model contains only a few structured attributes for price prediction. This makes it possible to compute a relative price residual on the Reduced and on the Full Feature Model for the new cars dataset. The RMSE between the $price_{pred}$ and $price_{sold}$ for both models is shown in Table 4.1. As expected, the Full Feature Model has a lower RMSE as the Reduced Feature Model. Besides, the Mean Absolute Deviation (MAD) computed on the relative price residual PR_{rel} is lower for the Full Feature Model. This makes it possible to apply my approach on a less and a more accurate target variable. I assume that the prediction accuracy on the price residuals calculated with the Reduced Feature Model is higher, because the vehicle texts also contain technical information that is usually covered in the structured attributes. A pricing model trained on fewer features can therefore be more easily improved by information from the texts. In this way it can be tested whether my approach to predict price residual classes works and whether the vehicle description texts can explain a price difference between prediction and sales price.

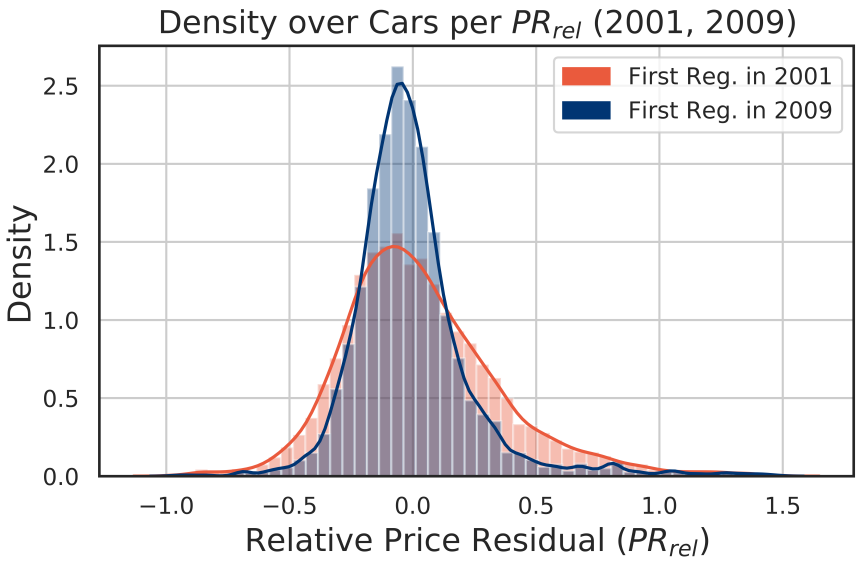


Figure 4.4.: The plot incorporates instances from the older cars dataset. The density over cars per Relative Price Residual PR_{rel} that had their first registration in 2001 (n = 7,572; μ = 0.033; σ = 0.322) is displayed by the red curve. The blue curve shows the density for cars that were first registered in 2009 (n = 7,566; μ = −0.006; σ = 0.241). Newer cars tend to have a smaller PR_{rel}, so the Full Featured Model based on structured attributes predicts the price more accurately for newer cars than for older cars.

The second dataset contains older vehicles that were licensed before June 2009. Figure A.2 illustrated in the appendix shows all cars per year of first registration. The oldest cars were registered in 2000. As the RMSE and the MAD of the Full Feature Model for the second dataset indicate, the price prediction model for older cars has even a higher error than the Reduced Model for the first dataset. This leads to the assumption that structured attributes have less influence on the price for older cars. I therefore conclude that individual conditions are more decisive for price prognosis and are presented in the description texts of the automobiles. Older cars in particular do have a longer text length on average, which is indicated by Histogram A.3 shown in the appendix.
Moreover, Figure 4.4 displays the density of cars per PR_{rel} for vehicles that had their first registration in 2001 (red) and 2009 (blue). The standard deviation $\sigma = 0.241$ for newer cars is lower than $\sigma = 0.322$ for older ones. Thus, I will especially test if it is possible to explain the relative price residual PR_{rel} with the descriptive texts for older vehicles. I expect to achieve the greatest impact by analyzing their texts.
Another important insight that can be obtained from the data is that the text length varies over PR_{rel}. Underpredicted cars have longer text lengths than overpredicted cars, as shown in Figure A.4 that can be found in the appendix. For example, a car that has a PR_{rel} of -0.9 has an average text length of 24 words, while a car with $PR_{rel} = 0.7$ is described by a text of 17 words on average. This makes sense because people want to improve the price of a car by writing longer description texts and outlining its individual features.

4.2.1. Human-Based Evaluation of Vehicle Description Texts

In this subsection, I will test the assumption whether vehicle description texts in online car markets contain information about the price of a car. Besides, I will provide a human-based evaluation for labeling vehicle description texts according to the three classes: underpredicted, neutral and overpredicted. This analysis is based on 360 description texts that are taken from the older cars dataset. At first, the whole dataset is split into three parts of equal size according to their predicted market price $price_{pred}$, leading to a low, a mid and a high price segment. The price intervals can be obtained from Table A.1 that is illustrated in the appendix. This is done to test whether higher or lower prices also have an impact on the prediction performance of relative price residuals in general.
For example, a minimal change on the absolute price may greatly increase or decrease the relative price residual for a low price, while the relative price residual for a high price is more stable. The exchange of a low priced car component, which could be described in the description text, leads to a different impact on the relative residual.

For the manual analysis, 360 texts are extracted from the three price segments, leading to 120 instances per price set. I only extracted texts that are longer than the median of $\bar{x}_{0.5} = 11$ words, presuming that longer texts contain more meaningful information than shorter ones. Moreover, every set of 120 texts contains 40 texts of each price residual class in order to obtain an equal class distribution of 33.3% . After the construction of these sets, the texts are read and tagged according to the classes: underpredicted, neutral and overpredicted. I will present the accuracy values of this human-based analysis now. The low priced cars were predicted with an accuracy of 45,00%. The mid price range was most

accurately tagged with an accuracy of 51.67%, while the high priced car set was classified by a human with an accuracy of 40.00% only. This suggests that the vehicle description texts of mid priced cars can be most precisely predicted to explain a price deviation.

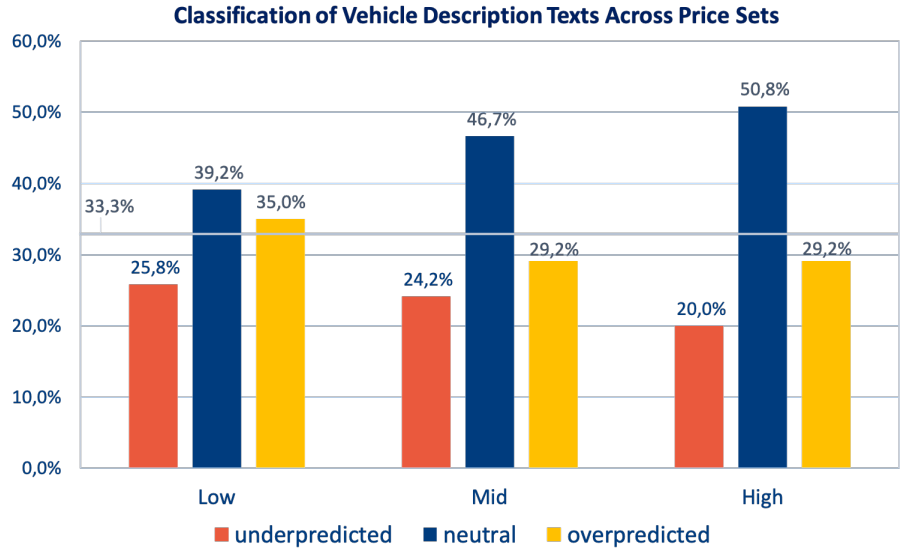


Figure 4.5.: The columns illustrates how many classes a human found in the three price sets. Low, mid and high priced cars were evaluated separately. Every price set contained 120 car texts that were classified according to three classes: underpredicted, neutral and overpredicted. The gray line at 33.3% indicates the real distribution in every set.

Besides evaluating the accuracy of the low, mid and high price set, it is also crucial to assess which residual price class has been classified the most. Figure 4.5 illustrates the distribution of the price residual classes in the three price sets generated by the human-based evaluation. It illustrates that texts often do not indicate an underpridction. The human tagged approximately 23% of the texts with the underpredicted class (red). Therefore, user-generated vehicle description texts do net present the cars in a better way to justify a higher selling price. In addition, the neutral price residual class is tagged the most in all three price segments (blue). This indicates that the vehicle description texts often have no influence on the price. To test this hypothesis I will read the texts a second time, but the outcome is known this time. I will discuss this evaluation in the next paragraph.

In the second human-based analysis, the texts are classified with regard to the question whether the text explains the difference between the selling price $price_{sold}$ and the predicted price $price_{pred}$. For this evaluation, it was necessary to re-read the texts while knowing the actual price residual class for the texts. The results of the task are illustrated in Figure 4.6. In the lower price segment, 50.83% of the vehicle description texts explain the price difference, and in the upper price segment only 40% are explained by the texts. Both tasks indicate an upper value for prediction accuracy at approximately 50%.
The first analysis provided the result that a human predictor achieves an accuracy of

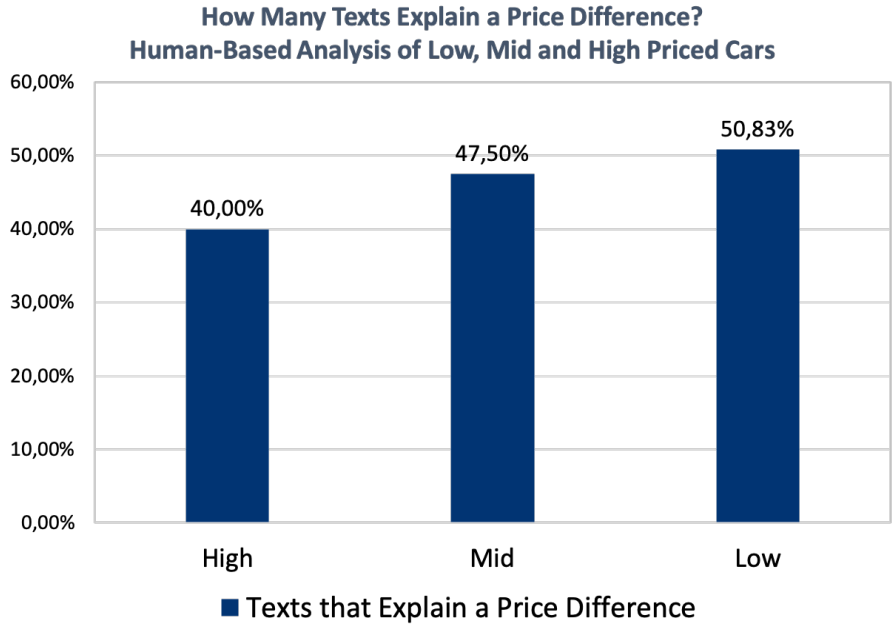


Figure 4.6.: The texts were read a second time, but this time the outcome was known. The blue columns show how many texts explained a difference between the selling price $price_{sold}$ and the predicted price $price_{pred}$.

51.67% to predict the three classes: underpredicted, neutral and overpredicted. The second task indicates that only 40%-50% of the texts explain a price difference. This shows that the human predictor did some good guesses for the mid priced car segment in the classification task, because only 47.5% of the texts explained the price difference here.

4.2.2. Results Obtained from Data Understanding

The phase of Data Understanding provided a detailed analysis of vehicle description texts from an online car marketplace. In general, price prediction for older cars on structured attributes is less accurate than for newer cars. I showed that the texts have an average text length of 20 words and a median of 11. Thus, the description texts are short, which is an indicator that the they do not contain very much information. Furthermore, manually labeling longer texts illustrate that the descriptions often do not explain the price difference between the predicted price and the selling price. Many texts repeat the structured attributes and frequently state an opposite interpretation, such as indicating an overprediction when an underprediction happened. For example, it was written in some texts that a car's body got scratches and bumps, but the selling price was higher than the predicted market price. The human-based analysis showed that even a human was not able to surpass an prediction accuracy of 51.67% for classifying the texts into underpredicted, neutral or overpredicted.

4.3. Data Preparation

Preparation of texts is crucial for the success of their analysis. The thesis' theoretical part discussed various text preprocessing techniques. The phase of preparing data is focused on text transformation and splitting the data into a train and test set for modeling and evaluation. I will outline shortly, which configurations of data preprocessing are used to answer my second research question: *How do such approaches perform on real description text data for vehicles?* I start with the description of my word preprocessing approach, in which the individual words in the vehicle description texts are prepared.

4.3.1. Word Preprocessing

The first step in my implementation is the transformation of texts into tokens. The theory of tokenization was explained in Paragraph 2.2.2.1. I apply a modified version of the *spaCy* Tokenizer for the German language to all texts of the datasets. The *spaCy*-Tokenizer splits a whole text based on the occurrence of space symbols „ “. It then checks whether an exception rule can be applied from the specified language, such as splitting the English word „who's“ into „who“ and „'s“. The two characters „'s“ abbreviate the word „is“. Afterwards, the Tokenizer looks at prefixes, suffixes and infixes of words and checks if they could divide a word into two parts. Thereby, punctuations, hyphens and quotes serve as additional split criteria. Because the provided German vehicle description texts contain many symbols, such as parentheses and bullet points, I adjust the Tokenizer by adding regular expressions that catch more prefixes, suffixes and infixes. All rules of the *spaCy*-Tokenizer are hard-coded for every language. Furthermore, the Tokenizer is working in a loop. If a word has been split by an exception, prefix, suffix or infix, the Tokenizer runs through the newly generated words again. This makes it possible to detect nested tokens that were build up by many words [Exp17]. The average word length for the data that was split by the *spaCy*-Tokenizer is shown in Table 4.1.

The second phase is lemmatization or stemming. Both approaches are tested to check which one works better. It makes sense to apply lemmatization before stop word removal, due to the fact that lemmas are contained in stop word lists. In contrast, stemming should be applied afterwards, unless a stemmed stop word list is available. Lemmatization is done by *spaCy*'s Lemmatizer and *NLTK* provides the Snowball-Stemmer for German.

The next step is stop word removal, which is applied to reduce the overall vocabulary in the corpus. I will test whether my approach performs better with or without stop word removal, because stop words could be essential for receiving correct information about a car's condition. For example, if somebody writes that the car has „no“ scratches or „no“ rust, the stop word „no“ changes the whole statement of a vehicle description text and is therefore important. The stop word list is provided by *spaCy*'s German language model and is implemented by simply checking whether a word from the text is included in the stop word list. For better word recognition and a second reduction in vocabulary, I have converted all words to lowercase before stop word removal.

4.3.2. Document and Word Representations

Transforming words and documents to machine readable representations is the last part in my preprocessing pipeline, as they work as input for the modeling phase. Five different approaches are tested separately: Vector Space Model with term frequency, Vector Space Model with tf-idf, word2vec, GloVe and fastText. They represent words and documents in different forms. This subsection describes how the word representations are constructed and in what kind of format I will use them.

4.3.2.1. Vector Space Model

The Vector Space Model represents documents as vectors. I construct these document vectors on the new and old car dataset. Therefore, all features (words) from a vehicle description text are included in a single document vector. The vocabulary of the newer car dataset contains 38,965 unique words, while the old car set has a vocabulary of 56,347 words. The vector size for representing a document in the Vector Space Model equals the number of words in a dataset. The theory for this was presented in Subsection 2.2.2.5.
The first representation build by the Vector Space Model is based on term frequency only. I count the occurrences of all words in a vehicle description text and set their corresponding entries in the document vector to their quantity. For example, a vehicle description text from the newer cars dataset will be represented by a vector with the length of 38,965. The vector for the document d is created by

$$\Psi_{tf}(d) = (tf(w_1, d), tf(w_2, d), \ldots, tf(w_{\mathrm{n}}, d)) \in \mathbb{R}^n, \tag{4.2}$$

where $n = 38965$. $tf(w_i, d)$ indicates the term frequency in the document for every word in the vocabulary.
The second representation that is build by the Vector Space Model is based on tf-idf. I use the Python library *scikit-learn* and the relative normalized term frequency ($\mathrm{ntf}_{\mathrm{rel}}$) to calculate the tf-idf for every word in a document. tf-idf weights the words according to their inverse occurrence in the whole corpus. Frequent words do have a lower weight than rare words.

4.3.2.2. Word Embeddings

In Chapter 3 three state-of-the-art approaches for the creation of word embeddings were presented. I will use word2vec, GloVe and fastText to represent a vehicle description text by a sequence of word vectors. For example, a document $d = w_1, w_2, \ldots, w_{|d|}$ will be presented by

$$\Psi_{embed}(w_1, w_2, \ldots, w_{|d|}) = v_1, v_2, \ldots, v_{|d|} = \underbrace{\begin{pmatrix} 0.41 \\ \vdots \\ 0.78 \end{pmatrix}, \begin{pmatrix} 0.20 \\ \vdots \\ 0.97 \end{pmatrix}, \ldots, \begin{pmatrix} 0.32 \\ \vdots \\ 0.30 \end{pmatrix}}_{|d| \text{ word vectors}}, \tag{4.3}$$

whereby v_i has the length of the embedding dimension (ED). The word vectors $v_1, v_2, \ldots, v_{|d|}$ and their size depend on the word embedding method used. I will continue to explain how the word vectors are created with the word2vec approach.
The word2vec representations are trained with the CBOW approach and a context window of 5 words. 3,728,006 vehicle description texts that include 9,787,091 sentences are used as input for training. This set of vehicle description texts was extracted separately from the online car market. Furthermore, ED is set to 300. The word embeddings are trained with the implementation from the Python package *gensim*. word2vec enables words used in a similar local context to have similar word vectors. For example, the most similar words for the word „beule“, which is German for „bump“, is „delle“, „kratzer“ and „schramme“. They all indicate a small damage in a car's bodywork and are used in the same contexts.

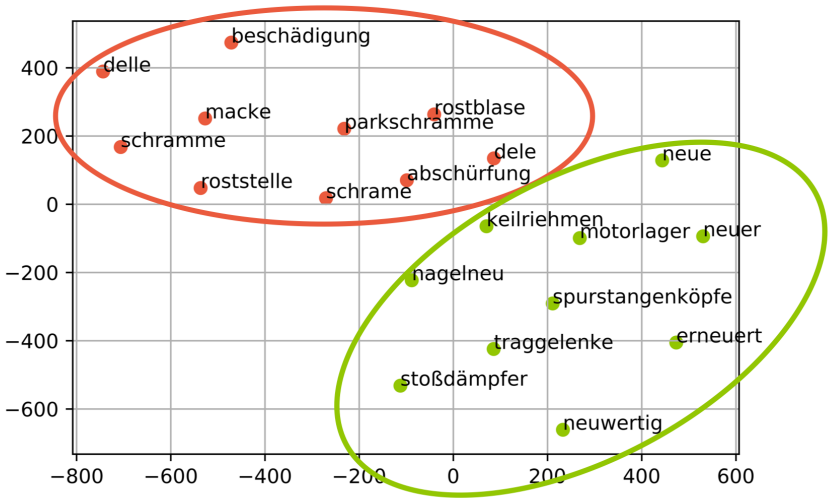


Figure 4.7.: The graph shows a t-Distributed Stochastic Neighbor Embedding (t-SNE) of word2vec embeddings trained on 3,728,006 vehicle description texts. t-SNE maps the embeddings with a space of 300 into a two-dimensional representation. The red dots are the 10 most similar word embeddings for „beule“ and the green points show the 10 most similar word embeddings for „neu“. This graph illustrates that both clusters are used in different context and can clearly be separated.

Figure 4.7 shows the positions of the most similar words of „beule“ in a two-dimensional space, which was created by transforming the high-dimensional embeddings with t-Distributed Stochastic Neighbor Embedding (t-SNE).
The green word embeddings show the most related words for „neu“, which translates to the English word „new“. Both word clusters are used in different context and can clearly be separated. In addition, the figure provides an insight into the orthography of the vehicle description texts. Three out of 20 words that are illustrated here are misspelled. However, since word embeddings generate word vectors based on context, the spelling should be a minor problem when the embeddings are trained locally on the vehicle description texts.

Besides word2vec, I will also test the performance GloVe as input representation. GloVe will be trained on the same dataset as word2vec and the embedding size is also set to 300. The word vectors are trained with the implementation provided by the authors of [PSM14][1].

fastText is also tested as word embedding, but the embeddings are pretrained on a large corpus containing German News and Wikipedia articles. I want to check whether pretrained embeddings can outperform embeddings trained on task specific texts. I use fastText for this task, due to the fact that fastText trains on subword information, which makes it suitable to build up word vectors for made up and out of vocabulary words.

4.3.3. Train- and Testsplit

This subsection will shortly illustrate the train- and testsplit for my implementation. Table 4.2 indicates that 20% of the data are used to test my models, while 80% are used to train them. I pick the test and train instances at random. With this split approximately 50,000 instances can be used for training a model for the newer cars dataset, which is enough to train complex ANN-models.

Train- and Testsplit:	**1: New Cars**	**2: Old Cars**
Number of Train Instances:	49,554	89,831
Number of Test Instances:	12,389	22,458
Total Vocabulary:	38,965	56,347
Vocabulary of Train Set:	34,455	49,805
Vocabulary of Test Set:	15,897	22,616
Shared Vocabulary:	11,387	16,074

Table 4.2.: Both datasets are split using a 80/20 split. 80% of data are used to train the model and 20% provide a hold out set for evaluation. In addition, the instances are selected at random. The train- and test set share in both datasets approximately 30% of the vocabulary.

The vocabulary in the train- and test set are not evenly distributed. Figure 4.8 displays a Venn diagram for the new cars dataset, which illustrates the distribution over the vocabulary in its train and test set. 29.22% of the vocabulary appear in both sets. Therefore, approximately 30% of the words in the vocabulary are features that a predictive model already has seen during training and come up in test set again. However, the vehicle description texts of the new car dataset contain 1,127,233 words and 1,086,279 of this words are in the train and test set. Therefore, 96.37% of the overall words are covered by 29.22%

[1] `https://github.com/stanfordnlp/GloVe`

of the vocabulary. The Venn diagram for the older car dataset is shown in the appendix (Figure A.5).

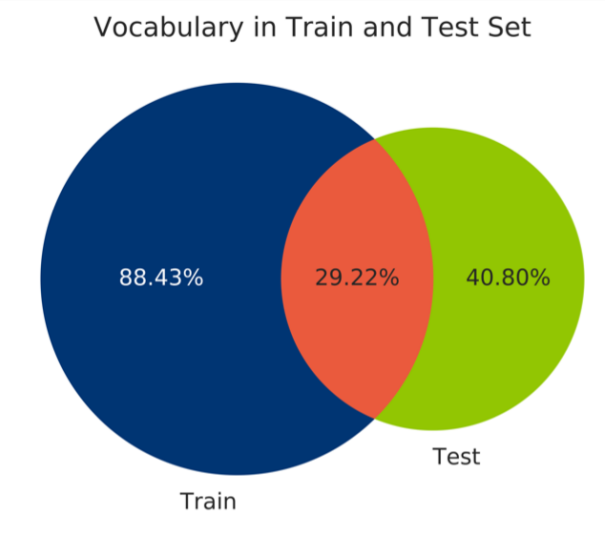


Figure 4.8.: This Venn diagram shows the vocabulary in the train- and test set of the new cars dataset. 88.43% of the words in the vocabulary are contained in the training set, while the test dataset incorporates 40.80%. 29.22% of the vocabulary in the newer car dataset occur in the train- and test set, which is indicated by the red area.

4.4. Modeling

The last section in my approach focuses on details of the predictive models. As illustrated by Figure 4.2, I will use Artificial Neural Networks to predict price residual classes, because they are recently used in NLP, as shown in Chapter 3. A FNN, a LSTM and a LSTM with an attention mechanism will serve as prediction algorithms. Nevertheless, I will also compare them to a classical machine learning approach. Random Forest will serve as a baseline here. Table 4.3 shows my different input-model combinations. The objective of all models is the classification of a vehicle description text. They are all trained on the train set and evaluated on the test set. The results are discussed in the next chapter. Random Forest and the FNN process document vectors. Because Random Forest serves only as baseline, I will test term frequency and tf-idf only. The RNNs take sequential word vectors as inputs that will be provided by the word embedding approaches. In addition, it is possible to use averaged word embeddings for a document as input for the FNN. The word vector for each word in a document is used. All these vectors are then summed up to compute their average. I will examine this input for the FNN by using word2vec. In the training phase, I set the split criterion to the information gain and construct ten trees. Furthermore, I used *scikit-learn* to implement the classifier.

Model:	Random Forest	FNN	RNNs
Vector Space Model tf:	✓	✓	-
Vector Space Model tf-idf:	✓	✓	-
word2vec	-	✓	✓
GloVe	-	-	✓
fastText	-	-	✓

Table 4.3.: This table shows different input-model configurations, which are tested to predict the PR_{rel}-classes. Random Forest and FNN take a single document vector as input, while the recurrent algorithms deal with a sequence of word vectors. For incorporating word embeddings into the FNN, I use averaged word vectors to create one document vector.

4.4.1. Random Forest

Besides the human-based labeling of the texts, which was discussed in the Section 4.2.1, I will test Random Forest to provide another baseline. The decision trees of the Random Forest are trained to forecast a price residual class for every description text. In Subsection 2.2.3.1 I have explained the theory of Random Forest. The input is a entire document vector that has been converted into a machine readable representation by the Vector Space Model. I will use term frequency and tf-idf as input and test which one performs better. Figure 4.9 illustrates the implementation for the term frequency input.

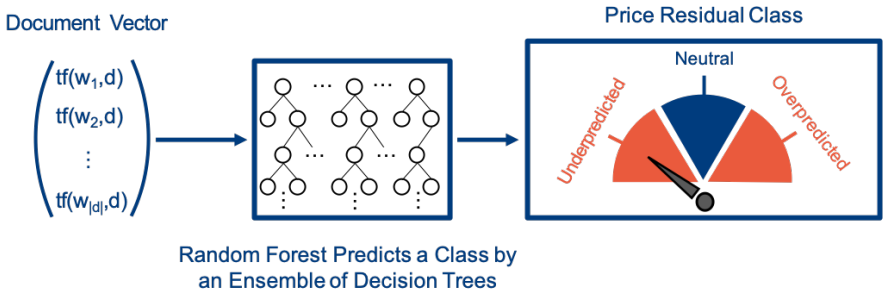


Figure 4.9.: The Figure illustrates the schema for the Random Forest classifier. An ensemble of decision trees predicts the class for the vehicle description text. The input is a document vector that was created by the Vector Space Model and the term frequency.

4.4.2. Feed Forward Neural Network

After the implementation of Random Forest that will provide a baseline, I integrate a FNN into my approach. *PyTorch* is used as supporting framework to implement the FNN. The

input of the network are document vectors with the length of the vocabulary generated by the Vector Space Model and averaged word2vec. When testing the averaged word embeddings, the FNN processes vectors with the length of the embedding dimension of word2vec. The FNNs output is determined by a softmax function on top of the output layer that computes a probability for each class. Figure 4.10 shows the schema for my implementation of the FNN model.

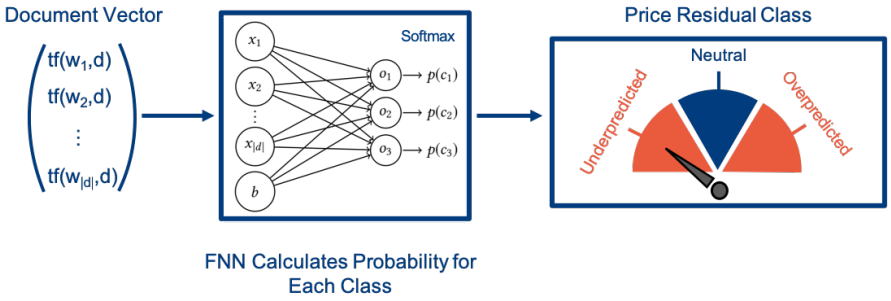


Figure 4.10.: The Figure illustrates the architecture for the FNN used in my approach. It calculates the probability for each class with a softmax function on top of the output layer. The size of the input layer equals the size of the vocabulary in a dataset. The input vector is created by the Vector Space Model based on term frequency.

In addition, I train the network with mini-batching and the Adaptive Moment Estimation (Adam), which is a stochastic optimization algorithm that calculates „adaptive learning rates for different parameters from estimates of first and second moments of the gradients“ [KB15]. Therefore, Adam adapts its learning rate η during training and optimizes the learning rate for every parameter. Due to the fact that Adam works well in practice and outperforms SGD [KB15], I will choose it as the optimization algorithm to train the FNN. By using this approach, I need to specify two hyperparameters. The first one is the batch size that is used in the training step and the second hyperparameter is the initial learning rate for updating the network's weights. I will do a grid search on these parameters, which will be discussed in the next chapter. Moreover, I also add a hidden layer to the FNN to improve the model complexity and test whether this could improve the prediction accuracy on the test set. I use ReLU as activation function between the hidden and output layer, because it usually outperform other functions like tanh, as discussed in Section 2.3.2.

4.4.3. Recurrent Neural Networks

The recurrent neural networks that I implement in this thesis are inspired by the work of [PCJ17] and [RE15]. A LSTM and a LSTM with an attention mechanism are used to predict the price residual classes for vehicle description texts. Just like implementing the FNN, I also use *PyTorch* for the RNNs. I test them with different kinds of word embeddings, such as word2vec, GloVe or fastText. Because, RNNs support sequential inputs, every word is converted to its corresponding word vector. These vectors are then passed into the network in the order in which they appear in the vehicle description text. By using this

sequential input, it is possible to get the meaning of statements built with more than one word. Especially, negative statements that are introduced by negating word, such as „no“ or „none“ are found. This should increase the prediction accuracy for the price residual classes. The training of both networks is also done by Adam and mini-batching. *PyTorch* processes batches with the same length only. Therefore, I take the length of the longest document in a dataset and fill the difference in shorter texts with vectors of zeros. Both RNNs use this sequential inputs. Furthermore, I need to specify three hyperparameters: learning rate, hidden size and batch size, which will be optimized in a grid search.

4.4.3.1. LSTM

The LSTM analyzes the sequence step by step and generates a hidden state h_i after processing each input vector. The last hidden state $h_{|d|}$ is forwarded to an FNN, which finally predicts the class for the vehicle description text. A softmax function on top of the output layer calculates the class probabilities. Figure 4.11 illustrates the schema of this procedure.

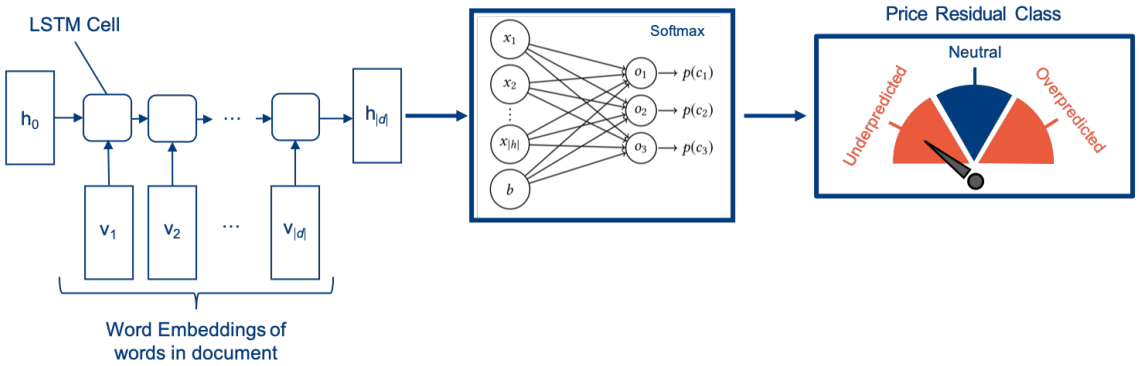


Figure 4.11.: The word embeddings of a vehicle description text are passed into the LSTM network. The last hidden state of the sequence is put into a FNN that predicts the class for the text. Thus, the FNN takes vectors with the length of a hidden LSTM state h. To predict the price residual class, a softmax function is used on the output layer.

4.4.3.2. Attention-LSTM

The second LSTM, which is tested in my research has got an attention mechanism that is similar to the ones proposed in [PCJ17] and [RE15]. The whole structure, which is illustrated in Figure 4.12, is more complex than the one for the LSTM introduced above. A LSTM processes the word vectors step by step, but forwards them to an attention network. This network computes attention weights for every hidden state h_i. Since a softmax function on top of the attention network's output layer is used, the weights add up to 1. Every hidden state is then multiplied with its corresponding attention weight. Afterwards, all weighted states are summed up to create a context state c, which is forwarded to a FNN that calculates the probability for each price residual class. This approach shifts the focus

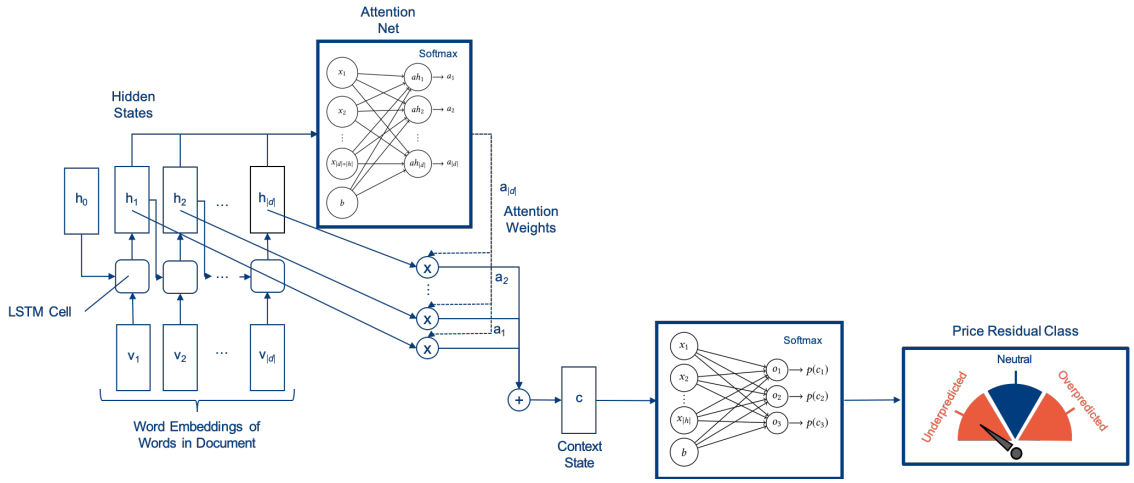


Figure 4.12.: The word embeddings of a document are forwarded to a LSTM. An attention net calculates the attention weights for every hidden state h_i. The hidden states are multiplied with their attention weight and summed up to generate a context state c. This state is passed into an FNN, which predicts the price residual class.

to the words in the input sequence that are important for predicting the target variable. The result of both recurrent networks are presented in the next chapter. I will examine whether adding an attention mechanism to the LSTM can improve its performance.

5. Evaluation and Discussion

„Torture the data, and it will confess to anything.“

Ronald Coase

The fifth chapter of this thesis is devoted to the evaluation of my approach. Chapter 4 provided details about the applied research part of this thesis and includes preliminary work to answer the second research question: *How do such approaches perform on real description text data for vehicles in an online car market?* The following chapter will outline the performance of my proposed approaches on the new and old cars dataset that were presented in Section 4.2. This is done to investigate whether the prediction of price residuals works better when using the vehicle description texts of newer or older automobiles. I will provide the results of Random Forest as baseline, the FNN, the LSTM and the Attention-LSTM for both datasets. Training and evaluation of the ANNs is done on a NVIDIA Tesla K80 GPU that provides 24 GB of memory. I will compare the models in terms of their test accuracy. A random classifier would classify 33.3% of the vehicle description texts correctly, because all three classes (undepredicted, neutral and overpredicted) are equally distributed.
I will start with the results on the new cars dataset. In this section, I will evaluate my approach on PR_{rel}-classes computed by the Full Feature and Reduced Feature Model that were introduced in Table 4.1. At the end of this chapter, the performance of my approach is evaluated on the old cars dataset. This also includes a separate assessment whether my approach predicts low, mid or high priced cars more precisely. I will compare these results to the human-based evaluation in Section 4.2.1.

5.1. Evaluation of the Approach on New Cars

In the following section, I present the results for the new cars dataset. I will focus on the relative price residual as target variable that is computed by Full Feature Model at first. To give an overview of the results, Table 5.1 presents the best performance of every model. It displays the test accuracy for each algorithm. I compute the accuracy on all 12,389 test instances from the new cars dataset.
Random Forest is only applied to provide a baseline for comparing the ANNs with a classical machine learning approach. Therefore, I present its results shortly here. As mentioned in Section 4.4.1, 10 decision trees are used and the entropy is set as split criterion. With this setting, Random Forest predicts the price residual classes with a maximum accuracy of 43.22%, which is 9.89 percentage points better than randomly classifying the texts. The best input representation for Random Forest is tf-idf.

Model:	**Accuracy:**
Random Forest:	43.22%
FNN:	**45.78%**
LSTM:	44.98%
Attention-LSTM:	45.47%

Table 5.1.: The table shows the best accuracy values for all predictive models. The best results with the FNN are achieved with a lemmatized word input and no hidden layers.

The results of the ANNs will be presented in the next sections in more detail. Nevertheless, I will briefly summarize the outcomes here. The FNN predicts the vehicle description texts with an accuracy of 45.78%. The optimal hyperparameters for the FNN are no hidden layers, a training batch size of 512 and a learning rate of 0.003.
With word vectors created by word2vec, the LSTM accomplishes an accuracy of 44.98% with a batch size of 512, a hidden state size of 64 and a learning rate of 0.0003. The Attention-LSTM classifies 45.47% correctly with the same hidden size and batch size as the LSTM, but a higher η of 0.003. Overall, the FNN is the most accurate model on the new cars dataset.

5.1.1. Feed Forward Neural Network

This section provides a detailed evaluation of the FNN whose architecture was illustrated in Section 4.4.2. I will start with a grid search on the hyperparameters batch size and

Hyperparameters:	**Values:**
Learning Rate (η):	{0.0003, 0.001, 0.003, 0.01}
Batch Size:	{128, 256, 512}
Number of Hidden Layers:	{0, 1}
Hidden Size:	{16, 32, 128, 256, 512}
Word Preprocessing:	stop word removal, stemming, lemmatization
Input Representation:	term frequency, tf-idf, averaged word2vec

Table 5.2.: The tested hyperparameters for the FNN are shown in this table. Four different learning rates and three batch size are tested. The performance of a FNN with one hidden layer and four distinct hidden sizes is evaluated separately. In addition, I compare three word preprocessing techniques and three document transformation methods.

learning rate. Afterwards, I continue to test whether adding a hidden layer will improve the performance of the FNN. At the end of this subsection an evaluation of lemmatization, stemming and stop word removal will take place. Moreover, I will compare the performance of averaged word embeddings, the Vector Space Model with term frequency and tf-idf. I vary six different hyperparameters for the FNN, as shown in Table 5.2. These represent the initial parameters of the grid search only. If a parameter on the edge of the grid will perform best, I will extend the grid.

5.1.1.1. Grid Search on Batch Size and Learning Rate

For the realization of the hyperparameter grid search on batch size and learning rate, I use the Vector Space Model with term frequency as input for every vehicle description text and passes it into the network. This provides a simple setting to test which batch size and learning rate work best. I will test the performance of averaged word2vec and tf-idf in Subsection 5.1.1.4.
The initial learning rate for Adam is chosen from 0.0003, 0.001, 0.003 and 0.01. *PyTorch*'s default learning rate for Adam is 0.001, which conforms to the information given in literature [LH17]. Thus, I set the learning rate to values below and above this default parameter to see whether a higher or lower learning rate performs better. Moreover, I will start with a batch size of 128, 256 and 512, since a batch size between 32 and 512 training instances performs good in practice [Kes+16]. The training is done with early stopping and an evaluation on the whole test set in each training epoch. If the accuracy on the test set does not increase after 5 training iterations, the FNN stops training. I will compare the outcomes of the grid search with the test accuracy. Table 5.3 shows the results for

Maximum Accuracy on the Test Set - Early Stopping				
Batch Size	**Learning Rate η**			
	0.0003	0.001	**0.003**	0.01
128	45.42%	45.42%	45.32%	44.87%
256	45.18%	45.44%	45.37%	45.11%
512	45.31%	45.43%	**45.52%**	45.38%
1024	43.60%	45.41%	45.47%	45.24%

Table 5.3.: The results are generated by a grid search for optimizing batch size and learning rate of the FNN. The highest value is achieved with a training batch size of 512 and $\eta = 0.003$. This setting reaches the highest accuracy after 12 iterations.

all hyperparameters. The FNN performs best with a batch size of 512. Since this value is on the edge of the grid, I also test a batch size of 1024. However, increasing the batch size does not improve performance. The best result that the grid search found is gained after 12 training iterations. Hereby, an accuracy of 45.52% is achieved with $\eta = 0.003$ and

a training batch size of 512. Therefore, I will set the hyperparameters to these values for further improving the FNN.

5.1.1.2. Adding a Hidden Layer

To see whether increasing the complexity of the FNN can improve test accuracy, I add a hidden layer. This is the first step to crate a Deep FNN. For finding the best dimension of the hidden layer, another grid search is done. Learning rate is kept at 0.003 and the training batch size stay at 512. I set the hidden size to 16, 32, 128, 256 and 512 to see if a lower or higher dimension can increase the performance. Furthermore, ReLU is used as activation function between the hidden and output layer, as discussed in Section 4.4.2.

Hidden Size	**Maximum Accuracy on the Test Set - Early Stopping**
16	45.33%
32	**45.48%**
128	45.12%
256	45.09%
512	44.78%

Table 5.4.: This table shows the results of the grid search for a FNN with one hidden layer. Five different dimensions of the hidden layer are tested: 16, 32, 128, 256 and 512. For training, I use a batch size of 512 and a learning rate of 0.003. The highest accuracy on the test set of 45.48% is achieved after 3 training iterations with a hidden size of 32.

Table 5.4 illustrates that the best accuracy of 45.48% is reached with a hidden size of 32. The results indicate that adding a hidden layer to the network does not improve the test accuracy. The network with no hidden layers achieves a higher accuracy. Besides, the network with a hidden layer suffers from overfitting, which is shown by Figure 5.1.
The figure demonstrates the accuracy per training iteration for the network with one hidden layer and a hidden size of 32. The accuracy of the train set is shown by the red curve, whereas the blue curve indicates the accuracy on the test set. After 3 training iterations the test accuracy drops slowly, while the network learns the training data by heart. The hidden layer improves the networks capacity, which leads to low bias but high variance. To address this problem, I add L2-regularization to prevent the network from overfitting. As explained in Subsection 2.3.4.6, L2-regularization adds a penalty term when the weights of the network are updated. This restricts overfitting on the training data and usually increases generalization. Different penalty weights (λ) are tested to check whether L2-regularization can improve the accuracy on the test set. The results of this parameter optimization for λ are presented in Table A.2 in the appendix. $\lambda = 0.0001$ performs best with an accuracy of 45.25%. Thus, even L2-regularization does not prevent

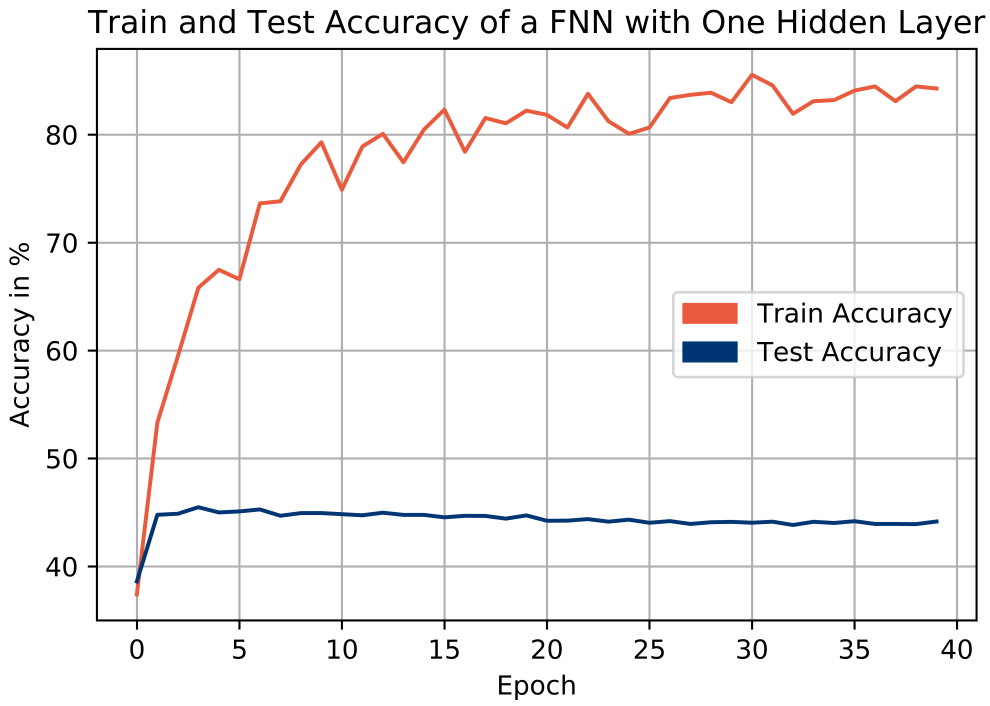


Figure 5.1.: Figure 5.1 shows the accuracy of the train- (red) and test set (blue) per training epoch for a FNN with one hidden layer and a hidden size of 32. To illustrate the overfitting, the FNN is trained with 40 training epochs. The red curve is growing very fast, while the blue curve is slightly decreasing after 3 epochs.

the FNN with one hidden layer from overfitting. On the contrary, the network trained with L2-regularization is worse than without it. In conclusion, adding a hidden layer does not improve the test accuracy. Therefore, I will not add more hidden layers, as this would lead to an even higher model capacity and an overfitting on the training data.

5.1.1.3. Optimization of Word Preprocessing

Besides optimizing the number of hidden layers, batch size and learning rate, it is necessary to find the best suited word inputs. Thus, I compare different methods for word preprocessing that were discussed in Section 4.3.1. Combinations of stop word removal (swr), stemming and lemmatization are tested. Just like in Subsection 5.1.1.1, I use the Vector Space Model with term frequency as representation for the vehicle description texts, a learning rate of 0.003 and a batch size of 512.

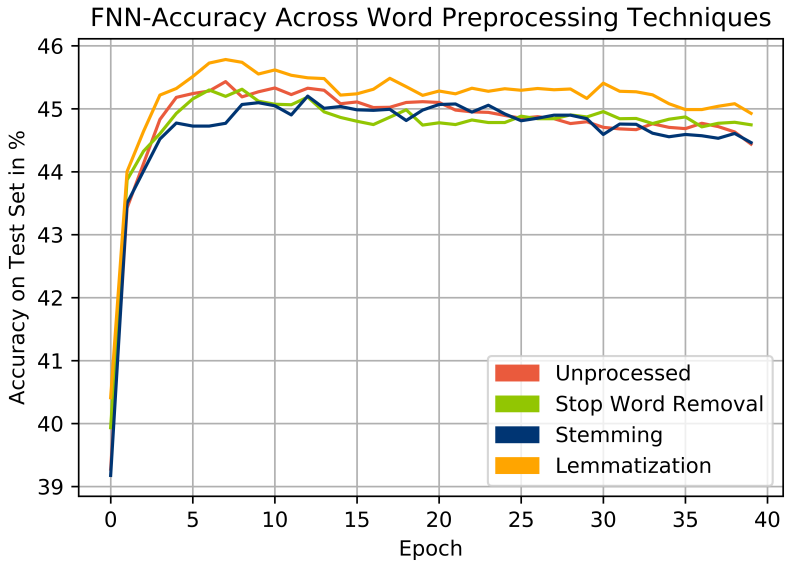


Figure 5.2.: The plot shows the test accuracy per training iteration for different word preprocessing methods. The performances of stop word removal (green), stemming (blue) and lemmatization (orange) are displayed. To see if word preprocessing can make a difference, the red curve illustrates the performance of an unprocessed word input. A lemmatized input without stop word removal performs best.

Figure 5.2 shows how the different word preprocessing methods perform. For the sake of clarity, I only illustrate the performance of an unprocessed, stop word removed, stemmed and lemmatized word input. I also test all other useful combinations, such as lemmatization with stop word removal, but will not show their results here, since the best suited word preprocessing technique for the FNN is already included in the Figure.
In general, stop word removal does not improve prediction accuracy on the test set. On the contrary, it lowers it. This is a result that was expected, because stop words can change the whole meaning of a statement, which makes them important for price prediction. Thus,

stop word removal will be left out. A lemmatized input is better than any other word input for the FNN. Lemmatizing reduces the vocabulary in the new cars dataset from 38,965 to 36,297 words and therefore decreases the number of input features by 6.8 percentage points. The words are joined with their lemmas, which enables the FNN to recognize similar words. In order to get price information from vehicle description texts, it helps when words that have a similar meaning are associated with each other. This makes it easier to deduce the actual message of a text. With lemmatization a maximum accuracy of 45.78% is achieved. Every other combination of word preprocessing, such as applying lemmatization, stemming and stop word removal at the same time, perform worse than mere lemmatization. Therefore, I assume that a tokenized word input with lemmatization and no stop word removal works best for predicting relative price residual classes.

5.1.1.4. Choosing the Best Document Representation

Apart from word preprocessing, the transformation of the vehicle description texts into a document vector is also important for optimizing the FNN. Averaged word2vec, the Vector Space Model with term frequency and tf-idf are compared to choose the best document vector as input. This will be the last optimization I will do for the FNN.
Since lemmatization performs best for term frequency, I train the FNN with tf-idf and averaged word2vec embeddings on lemmatized words with no stop word removal. In addition, I also evaluate an unprocessed input with no lemmatization, no stop word removal and no stemming for tf-idf and averaged word2vec. The training is again done with a batch size of 512 and $\eta = 0.003$. The results generated by tf-idf are not better than those created with term frequency. Lemmatization with tf-idf reaches a maximum accuracy of 45.31% after 64 training iterations, while the unprocessed input achieves an accuracy of 45.29%.
The averaged word2vec vectors as input do not improve test accuracy either. A maximum accuracy of 44.26% is achieved with unlemmatized word embeddings. They are probably suffering from an averaging effect here. To test lemmatization, I train word vectors based on lemmatized words and calculate their average for every document. However, by using lemmatized word vectors as input, a maximum accuracy of 44.19% is reached.
Since RNNs are able to process word embeddings sequentially, I assume that they perform better in RNNs. This will be evaluated in the next section. Nevertheless, the best outcome that the FNN creates is an accuracy of 45.78% based on term frequency and lemmatization.

5.1.2. Long Short-Term Memory Networks

This Section discusses the results of RNNs. The structure of my LSTM and attention-based LSTM were illustrated in Section 4.4.3. I will investigate if sequence-based approaches outperform FNN and Random Forest. This section mainly focuses on the LSTM, because I first optimize it and use the insights for deploying the more complex attention-based LSTM whose performance will be evaluated in Subsection 5.1.2.3.
At first, I will start again with a grid search on the hyperparameters for the LSTM. This time not only the batch size and learning rate must be optimized, but also the size of the hidden state. Afterwards, I will compare the performance of word2vec, GloVe and

Hyperparameters:	Values:
Learning Rate:	{0.0003, 0.001, 0.003}
Batch Size:	{128, 256, 512}
Size of Hidden State:	{64, 128, 256, 512}
Input Representation:	word2vec, GloVe, fastText

Table 5.5.: Four different hyperparameters for the LSTM are tested, as this table illustrates. I use three different learning rates and three batch sizes to find the optimal setting. The dimension of the LSTM's hidden state is set to 64, 128, 256 and 512.

fastText as input. I will not compare stop word removal or a stemming, because they do not improve the result of the FNN. Furthermore, I will not test whether preprocessing words with lemmatization will provide an improvement in accuracy. Since RNNs use word embeddings as input, lemmatization will not lead to an enhancement, because a lemma of a word has a word vector similar to the word itself. In addition, unlemmatized word embeddings provided a better input for the FNN than lemmatized ones, as discussed in Section 5.1.1.4. Table 5.5 shows hyperparameters and their values for optimizing the LSTM.

5.1.2.1. Grid Search on Hidden Size, Batch Size and Learning Rate

The grid search for the hyperparameters of the LSTM is similar to that for the FNN. However, LSTMs have a hidden state size that also needs to be optimized. I chose different dimensions for the hidden state vectors and test a size of 64, 128, 256 and 512. Furthermore, the batch size is set to 128, 256 and 512. The grid search implements learning rates of 0.0003, 0.001, and 0.003. This leads to 27 different training-settings for LSTM at least. I will again increase the grid, if a edge value performs best. The input for the network in this grid search is created by word2vec trained on user-generated vehicle description texts that were extracted from the online marketplace separately, as explained in Subsection 4.3.2.2. I use locally trained embeddings, because they usually outperform global embeddings trained on a general word corpus [Wan+18]. Table A.3 in the appendix provides the results for the LSTM's hyperparameter optimization. The test accuracy is again computed on all 12,389 test instances in every training epoch. The best result is achieved with a batch size of 512, a learning rate of 0.0003 and a hidden size of 64. These parameters are further used for finding the best word embeddings.

5.1.2.2. Finding the Best Word Embedding

This subsection is dedicated to the evaluation of different word embedding approaches, which were proposed in Subsection 4.3.2.2. I will compare word2vec and GloVe that are trained on vehicle description texts from the online car market. The performance of fastText trained on German news and Wikipedia texts will also be analyzed.

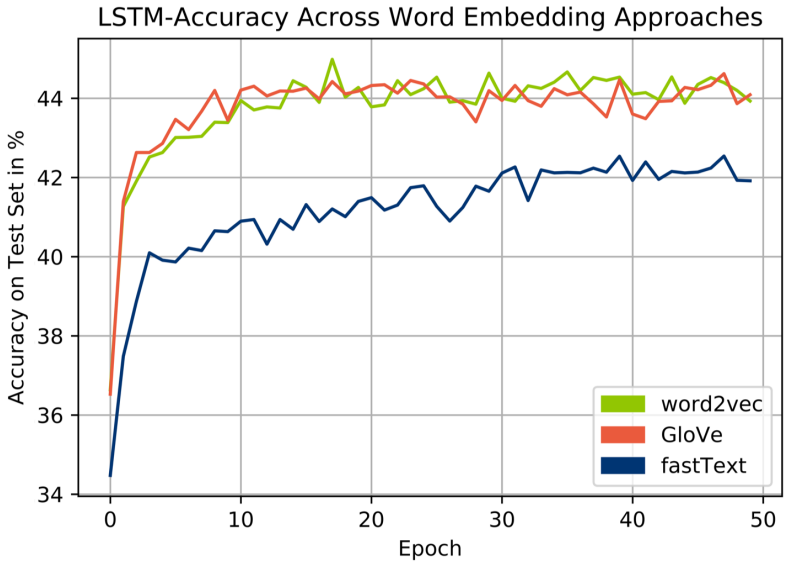


Figure 5.3.: The test accuracy per training epoch of the LSTM is displayed here. Three different word embeddings that provide the input for LSTM are tested; word2vec (green) and GloVe (red) are almost identical in performance, while fastText (blue) achieves the lowest results.

Figure 5.3 illustrates the test accuracy of the LSTM per training epoch across these word embedding approaches. The green curve shows how well the LSTM performs when word2vec is used as word embedding. The performance of GloVe (red) is comparable to that of word2vec. fastText achieves the lowest accuracy, which is illustrated by the blue curve. These results display that my approach works better on locally word embeddings that are trained on the domain-specific description texts than general word embeddings computed on German news and Wikipedia articles. This fact is also confirmed in literature [Wan+18]. The vehicle description texts do contain many domain-specific vocabulary and abbreviations. In addition, words are used in other contexts than in news or Wikipedia articles. The orthography of the user-generated texts is also on a different level, which was shown by Figure 4.7 in Subsection 4.3.2.2. Because word2vec achieves a marginally higher accuracy, I will use this method to compute input word vectors for the attention-based LSTM. The LSTM achieves a best accuracy of 44.98% that surpasses the performance of Random Forest, but does not exceed the results of the FNN.

5.1.2.3. Attention-LSTM

Optimizing the hyperparameters of the LSTM illustrate that a batch size of 512 and a small hidden size of 64 perform best. In addition, word embeddings trained on the user-generated description texts generate the highest accuracy. To fine-tune the hyperparameters of the attention-based LSTM, I do another grid search based on a batch sizes of 256, 512 and 1024. The hidden size is set to 32, 64 and 128. The word vectors that provide the input are generated by word2vec. Table 5.6 illustrates the results of the grid search.

Maximum Accuracy on the Test Set - Early Stopping					
Batch Size	**Dimension of Hidden State**	**Learning Rate η**			
		0.0003	0.001	**0.003**	0.01
256	32	42.09%	43.98%	45.26%	44.65%
	64	44.72%	44.85%	45.07%	44.40%
	128	44.58%	45.04%	45.31%	45.04%
512	32	43.01%	43.92%	43.64%	44.74%
	64	43.96%	44.81%	**45.47%**	44.73%
	128	44.33%	45.05%	45.26%	44.90%
1024	32	41.98%	43.42%	44.92%	45.04%
	64	43.22%	44.32%	44.75%	44.97%
	128	43.88%	44.16%	45.33%	44.58%

Table 5.6.: The table shows the results of the grid search done for the Attention-LSTM. The highest accuracy on the test set of 45.47% is achieved with a training batch size of 512, a hidden state dimension of 64 and $\eta = 0.003$.

The attention-based LSTM performs best with a batch size of 512 and a hidden size of 64, such as LSTM. The highest accuracy is 45.47%. The attention mechanism increases the accuracy of the LSTM by marginal 0.49 percentage points. On the new cars dataset, the LSTM with attention delivers the second best performance. It is 2.25% more accurate than Random Forest but can not surpass the FNN. Therefore, the FNN based on term frequency is best suited for the prediction of price residual classes in an online car market.

5.1.3. Difference between Reduced and Full Feature Model

This subsection will examine whether my approach explains the price residuals more precisely for a less or a more accurate pricing model. Moreover, it will investigate if it is possible to predict price residual classes from vehicle description texts in general. As shown by Table 4.1, two different price prediction models exist for the new cars dataset. I evaluate FNN on both models and compare their results. Test and training instances are the same, but the predicted price in the target objective PR_{rel} is computed one time with the Full Feature Model and another time with the Reduced Feature Model, which was trained on less structured attributes. I assume that the class prediction accuracy for the vehicle description texts is higher if the target variable is computed with the Reduced Feature Model. The vehicle description texts often contain the same information that is already included in structured attributes. Therefore, a price difference between selling price and predicted price based on fewer attributes can be more easily improved by the

texts. Attributes that have been left out for predicting the market price are described in the texts. The FNN extracts the information about these attributes to predict the price residual class. Thus, it should be more precise in explaining the price deviation for the feature-poor pricing model. In addition, if this assumption holds, the comparison shows that my approach can improve an existing pricing model.
Plot 5.4 illustrates the results of the FNN on the Reduced and Full Feature Model. The accuracy of predicting price residual classes is higher when the predicted sales prices are forecasted by the Reduced Feature Model. Therefore, it is possible to predict relative price residuals with user-generated vehicle description texts. The texts do explain the price residual classes more precisely when a less accurate model is used to predict the market price of a vehicle.

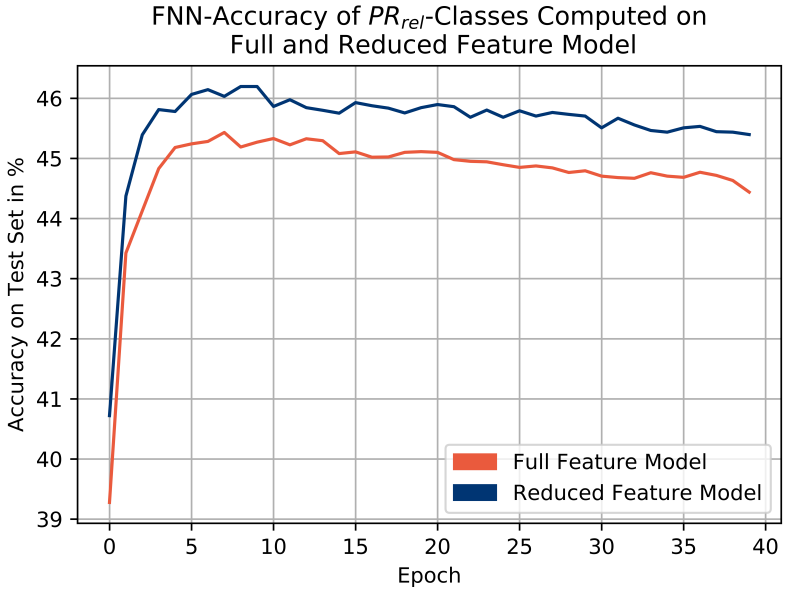


Figure 5.4.: The red and blue curves illustrate the accuracy of a FNN based on the Vector Space Model with term frequency. The red curve displays the classification accuracy of the price residuals classes computed with the Reduced Feature Model. In contrast, the blue line shows the accuracy when the classes are computed by the Full Feature Model. The evaluation is performed on all 12,389 test instances in each epoch.

5.2. Evaluation of the Approach on Old Cars

In the following sections, I will present the results of my approach on the old cars dataset. I want to improve the price prediction of old cars in particular, because their structured attributes are less decisive for their price, which was illustrated in Section 4.2. Table 5.7 shows the results for the old cars dataset. All models are trained with the optimized hyperparameters that were found in the previous sections. Random Forest achieves a maximum accuracy of 40.40%, which is lower compared to the results for the new cars

dataset. Besides Random Forest, the FNN performs also worse on the old cars dataset. An accuracy of 42.62% is reached by the FNN after 4 training epochs. The LSTM and Attention-LSTM also have a lower accuracy on the old cars dataset. After 15 training iterations, the LSTM reaches an accuracy of 42.31%. The attention-based LSTM is only marginally better than the mere LSTM. The best performance is again achieved by the FNN.

Model:	**Accuracy:**
Random Forest:	40.40%
FNN:	**42.62%**
LSTM:	42.31%
Attention-LSTM	42.52%

Table 5.7.: The table shows the accuracy for all predictive models on the old cars dataset. FNN reaches an accuracy of 42.62% and outperforms all other algorithms.

The results of the old cars dataset illustrate that my approach can improve the price prediction for older cars. However, all predictive models are more accurate on the new cars dataset. As illustrated in Section 4.2 it is more difficult to predict car prices based on structured attributes for older cars. My analysis shows that this is also holds for their description texts. Therefore, the descriptive texts for older vehicles do not explain the price residuals more precisely than those for newer vehicles. Compared to the random classification of vehicle description texts, the accuracy is increased by 9.29 percentage points. The human-based evaluation on texts from the old cars dataset in Section 4.2.1 showed that a human could reach a maximum accuracy of 51.67%. My approach, which produces an accuracy of 42.62%, is only 9.05 percentage points less accurate than a human. The human-based evaluation was the most accurate on mid-priced cars. To see whether my approach provides the same results, I will investigate its performance on low, mid and high priced cars in the next section.

5.2.1. Results on Low, Mid and High Priced Cars

The human-based evaluation that was discussed in Section 4.2.1 divides the old cars dataset into three price classes (low, mid, high). This was done to test whether vehicle description texts of higher or lower priced cars have a different impact on their relative price residuals. For example, in the case of a cheap car, the replacement of a high-priced vehicle component that could be described in the description text leads to a different effect on PR_{rel} than in the case of an expensive car. I will evaluate if my approach produces the same results as the human-based evaluation. To test this, I train the FNN three times on each price range to examine if they also have a different accuracy when predicting price residual classes for lower or higher priced cars. In each price range I use 80% of the texts for training. The evaluation is done on the remaining 20%. Table A.1 in the appendix illustrates the price ranges in more detail.

Figure 5.5 displays the results of the FNN across the three price ranges. It clearly illustrates that cars with mid-ranged car prices can be predicted more precisely, which is consistent with the results of the human-based evaluation in Section 4.2.1. Therefore, vehicle description texts of mid priced cars have the greatest influence on the explanation of their price residuals. For example, let us assume we have two cars that are both described by a similar user-generated text. Furthermore, one car is in the mid price range and the other one is from the low price range. Both description texts include a price reducing fact, such as a bump or scratch in a car's bodywork, which should be reflected in the selling price. Mid priced cars do usually have a better condition than low priced cars. Since, it is not expected that a mid priced car has scratches, the description text has a greater impact on its price. On the contrary, the relative price residuals of a high priced car does not change much if a scratch exist, because repainting a small car part does have small impact on a high price.

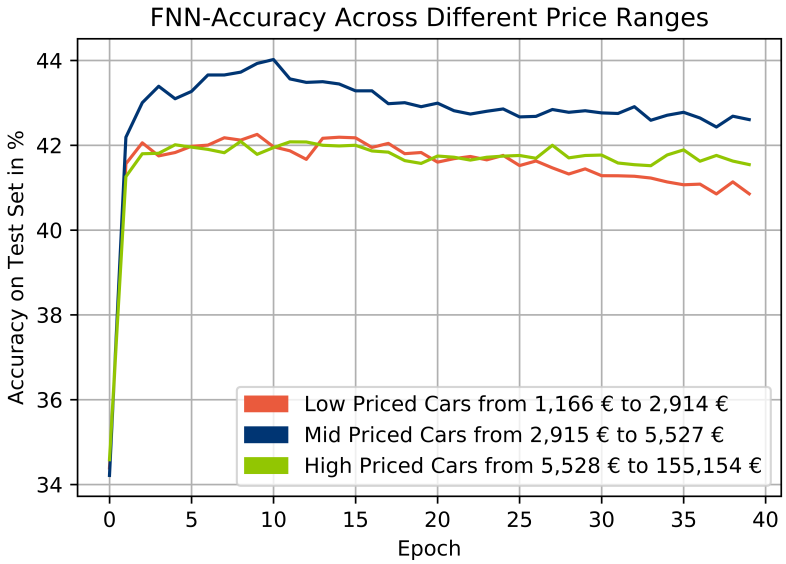


Figure 5.5.: The Figure displays the accuracy of a FNN that is based on the Vector Space Model with term frequency. The red curve illustrates the performance on low priced cars, while the green curve displays the accuracy on high priced cars. The most precise prediction is done on the mid priced car range (blue). These results are consistent with the human based evaluation from Section 4.2.1.

5.3. Summary of the Observed Results in Evaluation and Discussion

The practical part of this thesis shows that it is possible to increase price prediction by incorporating vehicle description texts. The evaluation of my approach on a target variable that was computed by a Reduced and Full Feature Model in Section 5.1.3 provides evidence

for this. Moreover, it is possible to increase the classification accuracy for predicting price residual classes up to 12.45 percentage points compared to random classification. The best predictive model from my approach is a FNN with no hidden layer. All models are more precise on the new cars dataset than on the older cars dataset. Discussing the results from the previous subsection showed that the relative price residual classes can be predicted the most accurate on vehicle description texts from mid priced cars.

6. Conclusion and Outlook

„In literature and in life we ultimately pursue, not conclusions, but beginnings."

Sam Tanenhaus

This master thesis presented current state-of-the-art techniques in NLP and showed an approach to apply these NLP methods to the prediction of price residual classes for user-generated vehicle description texts in an online car market.

Current research in NLP-based text analysis focuses on advanced Artificial Neural Networks that use meaningful word representations as input. This thesis covered FNNs, RNNs, attention based RNNs, HANs and CNNs that are contemporarily used to analyze texts with respect to a given target variable. Furthermore, I presented different types of word embeddings and used them in the approach to show their performance on user-generated vehicle description texts. The theory behind deep contextualized word representations like ELMo was also treated to give an outlook on the development of recent trends in NLP. The transformation of words into meaningful and machine-readable formats is an emerging field of research in NLP, driven by complex deep learning architectures like deep bidirectional RNNs. By providing theoretical foundations and a literature review in Chapters 2 and 3, the thesis answered the first research question: *Which state-of-the-art NLP-approaches exist to analyze user-generated product description texts with respect to their influence on the product price?* The focus was mainly set on text classification here, because this provided the best solutions to analyze texts with respect to their impact on product prices. Moreover, research indicated that the analysis of product description texts alone to predict a business outcome, such as a price, is sparsely covered in literature. The most similar work that is comparable to my approach was done by Pryzant et al. [PCJ17]. They use an attention-based LSTM to predict sales volumes based on product description texts.

The practical part of this thesis verified that user-generated texts in an online car market do have small influence on vehicle prices. To prove this, I tested whether vehicle description texts can explain the residuals between cars' predicted market prices based on structured attributes and actual selling prices. Chapters 4 and 5 focus on this and answered my second research question from Section 1.2: *How do such approaches perform on real description text data for vehicles in an online car market?* The theory from Chapters 2 and 3 was applied to a real world scenario.
I showed that state-of-the-art NLP approaches can achieve an accuracy of 45.78 % in predicting price residuals for vehicle description texts using three equiprobable classes. The best performance was achieved by a FNN with no hidden layers. Compared to a

classical machine learning model, the FNN improved accuracy by 2.2 to 2.5 percentage points. Since these user-generated texts contain on average 20 words, their analysis with respect to their price influence was not easy, be it for a human or a machine. The human-based evaluation in Section 4.2.1 showed that only 50% of user-generated description texts explain a price deviation between a car's predicted and observed selling price. This leads to the assumption that these description texts only contain few additional price information that is not already covered by the structured attributes. Further assumption can be that my models or data representation are not good enough. The evaluation of my approach on price residuals computed by a Full and Reduced Feature Model in Section 5.1.3 provided evidence against this. The performance of my approach was better when less structured attributes were used to compute the market prices. This showed that a price influence can be recognized by analyzing vehicle description texts with my approach, but a lot of price-relevant data is already contained in the structured attributes.

In conclusion, the difference between a car's market price based on its structured attributes and its selling price cannot be fully explained by user-generated vehicle description texts. More information to justify a price divergence could lie in the associated car images. Furthermore, other market factors, such as word of mouth or customer-specific automotive experience, that can hardly be summarized in data can have another influence on a car's price. However, text analysis can contribute a little to improve price prediction for vehicles in the online car market that provided data for my research. My results showed that the price residual classes for newer cars that had their first registration in 2009 or later could be predicted more accurately. For vehicles licensed before 2009, it was possible to predict mid priced cars more precisely, as discussed in Section 5.2.1.

Further work to improve the results of my approach could be done by using ELMo or other contextualized word representations as explained in Section 3.2. I did not apply these methods, because this would have exceeded the scope of my thesis. The next step for including my approach into the pricing model of the online car market would be the prediction of the relative price residual and not a price residual class. However, It was necessary to perform a fundamental analysis in this thesis at first to see whether it is possible to improve price prediction for vehicles by analyzing their user-generated description texts. The classification of these texts simplified this analysis and made it more comparable. Nevertheless, it is easy to adapt the ANNs in my approach to regression analysis by changing the activation function on top of the output layer. Future work that deals with the NLP-based analysis of vehicle description texts, should incorporate the facts that they are very short and have a domain-specific vocabulary. In addition, approximately 50% of the texts contain useful price information, which was shown by the human based-analysis. Compared to this upper limit, my approach, which predicts a correct price residual class for 45.78% of the texts, is only 4.22 percentage points less accurate than a human.

Bibliography

[AGI11] Nikolay Archak, Anindya Ghose, and Panagiotis G Ipeirotis. "Deriving the Pricing Power of Product Features by Mining Consumer Reviews". In: (2011), pp. 1–30.

[All+17] Mehdi Allahyari et al. "A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques". In: (2017). URL: http://arxiv.org/abs/1707.02919.

[Aut] Autotrader.com. *Sell Your Car - Resource Center - Autotrader*. URL: https://www.autotrader.com/sell-car/adviser/prepare-to-sell/pricing-your-vehicle/car-pricing-sources-explained.xhtml (visited on 01/03/2019).

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: (Sept. 2014). URL: http://arxiv.org/abs/1409.0473.

[Ben+01] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of Machine Learning Research* 3 (2001), pp. 1137–1155. URL: https://dl.acm.org/citation.cfm?id=944919.944966.

[Ber04] Michael W. Berry, ed. *Survey of text mining*. New York: Springer, 2004.

[Bia+17] Filippo Maria Bianchi et al. "An overview and comparative analysis of Recurrent Neural Networks for Short Term Load Forecasting". In: *Journal of Official Statistics* (2017), pp. 1–41. DOI: 10.1007/978-3-319-70338-1. URL: http://arxiv.org/abs/1705.04378%0Ahttp://dx.doi.org/10.1007/978-3-319-70338-1.

[BMZ11] Johan Bollen, Huina Mao, and Xiao-jun Zeng. "Twitter mood predicts the stock market ." In: (2011), pp. 1–8.

[Boj+15] Piotr Bojanowski et al. "Bag of Tricks for Efficient Text Classification". In: (2015).

[Boj+16] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: (July 2016). URL: http://arxiv.org/abs/1607.04606.

[Bre01] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 08856125. DOI: 10.1023/A:1010933404324. URL: http://link.springer.com/10.1023/A:1010933404324.

[Bri13] Ted Briscoe. *Introduction to Linguistics for Natural Language Processing*. Tech. rep. 2013, pp. 1–37.

[Car] Cars.com. *Price Comparison Tool*. URL: https://www.cars.com/price/ (visited on 01/03/2019).

[Cha+00] Pete Chapman et al. “Crisp-Dm 1.0”. In: *CRISP-DM Consortium* (2000), p. 76. ISSN: 0957-4174. DOI: 10.1109/ICETET.2008.239.

[Cho+14] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: (2014). ISSN: 09205691. DOI: 10.3115/v1/D14-1179. URL: http://arxiv.org/abs/1406.1078.

[Cho03] Gobinda G. Chowdhury. “Natural Language Processing”. In: *Annual Review of Information Science and Technology,* 37 (2003), pp. 51–89. URL: http://dx.doi.org/10.1002/aris.1440370103.

[Chu+14] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: (2014).

[Chu07] Kenneth Church. “A Pendulum Swung Too Far”. In: 2.4 (2007).

[Col+11] Ronan Collobert et al. “Natural Language Processing (Almost) from Scratch”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2493–2537.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: (Nov. 2015). URL: http://arxiv.org/abs/1511.07289.

[Dev+18] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: (2018).

[DH18] Changshun Du and Lei Huang. “Text classification research with attention-based recurrent neural networks”. In: *International Journal of Computers, Communications and Control* 13.1 (2018), pp. 50–61. ISSN: 18419844. DOI: 10.15837/ijccc.2018.1.3142.

[Dom18] Domo Inc. *Data Never Sleeps 6 | Domo*. 2018. URL: https://www.domo.com/learn/data-never-sleeps-6 (visited on 10/09/2018).

[DS17] Matthew James Denny and Arthur Spirling. “Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, And What To Do About It”. In: *SSRN Electronic Journal* 26.02 (Apr. 2017), pp. 168–189. ISSN: 1556-5068. DOI: 10.2139/ssrn.2849145. URL: https://www.cambridge.org/core/product/identifier/S1047198717000444/type/journal_article%20https://www.ssrn.com/abstract=2849145.

[Exp17] Explosion AI. *spaCy 101: Everything you need to know · spaCy Usage Documentation*. 2017. URL: https://spacy.io/usage/spacy-101#annotations-token.

[Fel98] Christiane [Hrsg.] Fellbaum, ed. *WordNet : an electronic lexical database*. Language, speech and communication. : No price. Cambridge, Mass. [u.a.]: MIT Press, 1998. ISBN: 0-262-06197-X.

[Fla] Flaticon. *Car - Free transport icons*. URL: https://www.flaticon.com/free-icon/car_171239#term=car&page=1&position=13 (visited on 01/16/2019).

[FPS97] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. “From Data Mining to Knowledge Discovery in Databases”. In: *Association For The Advancement Of Artificial Intelligence* (1997), p. 18. ISSN: 0738-4602. DOI: 10.1145/240455.240463.

[FS07] Ronen Feldman and James Sanger. *The text mining handbook : advanced approaches in analyzing unstructured data*. Cambridge: Cambridge Univ. Press, 2007. ISBN: 0-521-83657-3.

[Gar18] Gartner Inc. *5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018 - Smarter With Gartner*. 2018. URL: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/ (visited on 12/30/2018).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech Recognition with Deep Recurrent Neural Networks". In: *IEEE International Conference* 3 (2013), pp. 6645–6649.

[Gre+16] Zac Greene et al. "The Nuts and Bolts of Automated Text Analysis. Comparing Different Document Pre-Processing Techniques in Four Countries". In: (2016). DOI: 10.17605/OSF.IO/GHXJ8. URL: https://osf.io/ghxj8.

[HKW08] Judith Hurwitz, Daniel Kirsch, and John Wiley. *Machine Learning Machine Learning For Dummies ® , IBM Limited Edition*. 2008, p. 69. ISBN: 9781119454953. URL: http://www.wiley.com/go/permissions..

[Hoc+01] Sepp Hochreiter et al. "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies". In: *A Field Guide to Dynamical Recurrent Networks* (2001). ISSN: 1098-6596. DOI: 10.1109/9780470544037.ch14. URL: http://ieeexplore.ieee.org/search/srchabstract.jsp?arnumber=5264952.

[HQW06] Gerhard Heyer, Uwe Quasthoff, and Thomas Wittig. *Text Mining: Wissensrohstoff Text : Konzepte, Algorithmen, Ergebnisse*. IT lernen. Herdecke: W3L-Verl., 2006. ISBN: 3-937137-30-0.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Springer Series in Statistics*. 2nd ed. New York: Springer, New York, NY, 2009. ISBN: 0-387-94725-6 Springer-Verlag. DOI: 10.1007/978-0-387-98135-2. URL: http://www.springerlink.com/index/D7X7KX6772HQ2135.pdf.

[Hug+17] Mark Hughes et al. "Medical Text Classification using Convolutional Neural Networks". In: (2017), p. 5.

[ID10] Nitin INDURKHYA and Fred J Damerau. *HANDBOOK OF NATURAL LANGUAGE PROCESSING*. Chapman & Hall/CRC, 2010. ISBN: 978-1-4200-8593-8. URL: http://portal.acm.org/citation.cfm?doid=1220835.1220877.

[Jon94] Karen Sparck Jones. "Natural Language Processing : a Historical Review". In: *Natural Language Processing : a Historical Review* 9.10 (1994), pp. 53–59. DOI: 10.1007/978-0-585-35958-8{_}1. URL: https://pdfs.semanticscholar.org/7445/69e1ff6377ba0b7e3a8e2bcd88ac94d9a02d.pdf.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2015). URL: http://arxiv.org/abs/1412.6980.

[Kes+16] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: (2016), pp. 1–16. ISSN: 0148396X. DOI: 10.1227/01.NEU.0000255452.20602.C9. arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[KGB14] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: (2014). ISSN: 1365-2486. DOI: 10.3115/v1/P14-1062. URL: http://arxiv.org/abs/1404.2188.

[Kim14] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: (2014).

[Kro92] Anders Krogh. *A Simple Weight Decay Can Improve*. 1992.

[LBH15] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 14764687. DOI: 10.1038/nature14539.

[Lee+14] Heeyoung Lee et al. "On the Importance of Text Analysis for Stock Price Prediction". In: *Lrec* 2009 (2014), pp. 1170–1175. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/1065_Paper.pdf.

[LH17] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: 100 (2017). ISSN: 10495258. DOI: 10.1111/cbdd.12322. arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[Liu+12] Haibin Liu et al. "BioLemmatizer: a lemmatization tool for morphological processing of biomedical text." In: *Journal of biomedical semantics* 3 (Apr. 2012), p. 3. ISSN: 2041-1480. DOI: 10.1186/2041-1480-3-3. URL: http://www.ncbi.nlm.nih.gov/pubmed/22464129%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3359276.

[Lov68] Julie Beth Lovins. "Development of a stemming algorithm". In: *Mechanical Translation and Computational Linguistics* 11.June (1968), pp. 22–31. URL: http://journal.mercubuana.ac.id/data/MT-1968-Lovins.pdf.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: (2015). ISSN: 10495258. DOI: 10.18653/v1/D15-1166. URL: http://arxiv.org/abs/1508.04025.

[LW02] Liaw A and Wiener M. "Classification and Regression by Random Forest". In: *R news* 2/3.December (2002), pp. 18–22. ISSN: 16093631. DOI: 10.1177/154405910408300516. arXiv: 1609-3631. URL: http://www.bios.unc.edu/%7B~%7Ddzeng/BIOS740/randomforest.pdf.

[Mar15] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. 2015. ISBN: 9781466583337. DOI: 10.1145/242224.242229.

[Mik+13] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: (Jan. 2013). URL: http://arxiv.org/abs/1301.3781.

[Mit97] Tom M. Mitchell. *Machine learning*. 1997. ISBN: 0070428077.

[MLS13] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. "Exploiting Similarities among Languages for Machine Translation". In: (2013). ISSN: 10495258. DOI: 10.1162/153244303322533223. URL: http://arxiv.org/abs/1309.4168.

[Mob17] Mobile.de. *Preistransparenz bei mobile.de*. 2017. URL: https://www.mobile.de/magazin/news/2017/Preistransparenz.html (visited on 01/03/2019).

[MP43] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. ISSN: 00074985. DOI: 10.1007/BF02478259.

[MP69] Marvin L. Minsky and Seymour A. Papert. "Perceptrons (1988 ed)". In: 1969. ISBN: 0-262-63111-3. URL: https://ia801601.us.archive.org/1/items/Perceptrons/Perceptrons.pdf.

[MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations". In: *Proceedings of NAACL-HLT* June (2013), pp. 746–751. ISSN: 9781937284473. DOI: 10.3109/10826089109058901. URL: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Linguistic+Regularities+in+Continuous+Space+Word+Representations#0%5Cnhttps://www.aclweb.org/anthology/N/N13/N13-1090.pdf.

[Ola15] Christopher Olah. *Understanding LSTM Networks – colah's blog*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[OW17] David L. Olson and Desheng Wu. *Predictive Data Mining Models*. Computational Risk Management. Singapore: Springer Singapore, 2017. ISBN: 978-981-10-2542-6. DOI: 10.1007/978-981-10-2543-3. URL: http://link.springer.com/10.1007/978-981-10-2543-3.

[PCJ17] Reid Pryzant, Young-Joo Chung, and Dan Jurafsky. "Predicting Sales from the Language of Product Descriptions". In: *Proceedings of SIGIR, Tokyo, Japan, August 2017 (SIGIR 2017 eCom)* (2017). URL: https://nlp.stanford.edu/pubs/pryzant2017sigir.pdf.

[Pet+18] Matthew E. Peters et al. "Deep contextualized word representations". In: (2018). ISSN: 9781941643327. DOI: 10.18653/v1/N18-1202. URL: http://arxiv.org/abs/1802.05365.

[PKW] PKW.de. *Gebrauchtwagen mit Preis-Check - PKW.de*. URL: https://www.pkw.de/ (visited on 01/08/2019).

[Por80] M. F. Porter. *An algorithm for suffix stripping*. Mar. 1980. DOI: 10.1108/eb046814. URL: http://www.emeraldinsight.com/doi/10.1108/eb046814.

[PR01] Martin Porter and Boulton Richard. *Snowball*. 2001. URL: http://snowballstem.org/.

[Pre97] Lutz Prechelt. *Early Stopping - but when?* 1997.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "GloVe : Global Vectors for Word Representation". In: (2014).

[Rad+18] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: (2018), pp. 1–12.

[RE15] Colin Raffel and Daniel P W Ellis. "FEED-FORWARD NETWORKS WITH ATTENTION CAN SOLVE SOME LONG-TERM MEMORY PROBLEMS". In: (2015), pp. 1–6. URL: https://arxiv.org/abs/1512.08756.

[RGR17] David Reinsel, John Gantz, and John Rydning. "Data Age 2025 : The Evolution of Data to Life-Critica - Don 't Focus on Big Data; Focus on the Data That's Big Data Age 2025 :" in: April (2017), pp. 1–25.

[RK15] Anthony Rios and Ramakanth Kavuluru. "Convolutional Neural Networks for Biomedical Text Classification: Application in Indexing Biomedical Articles Anthony". In: (2015), pp. 258–267. DOI: 10.1145/2808719.2808746.Convolutional.

[RMD96] Pieter Th. van. Reenen, Margot van. Mulken, and J. W. Dyk. *Studies in stemmatology*. John Benjamins Pub. Co, 1996, p. 311. ISBN: 9027221537. URL: https://books.google.de/books/about/Studies_in_Stemmatology.html?id=eRk_XdVl_xMC&redir_esc=y.

[ŘS10] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (Sept. 2016). URL: http://arxiv.org/abs/1609.04747.

[Rug02] Salvatore Ruggieri. "Efficient C4 . 5". In: 14.2 (2002), pp. 438–444.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for Activation Functions". In: (2017), pp. 1–13. URL: https://arxiv.org/abs/1710.05941.

[Sal+17] Hojjat Salehinejad et al. "Recent Advances in Recurrent Neural Networks". In: (Dec. 2017). URL: http://arxiv.org/abs/1801.01078.

[SC09] Robert P Schumaker and Hsinchun Chen. "Textual Analysis of Stock Market Prediction Using Breaking Financial News : The AZFinText System". In: 2006 (2009), pp. 1–29.

[Sch15] Jürgen Schmidhuber. "Deep learning in neural networks : An overview". In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[SP97] Mike Schuster and Kuldip Paliwal. “Bidirectional Recurrent Neural Networks”. In: *IEEE* 45 (1997), pp. 2673–2681.

[Sri+14] Nitish Srivastava et al. “Dropout : A Simple Way to Prevent Neural Networks from Overfitting”. In: 15 (2014), pp. 1929–1958.

[SS09] Ashok Srivastava and Mehran Sahami, eds. *Text mining : classification, clustering, and applications*. Chapman and Hall/CRC data mining and knowledge discovery series. Boca Raton.: CRC Press, 2009.

[SSG08] Hemlata Sahu, Shalini Shrma, and Seema Gondhalakar. “A Brief Overview on Data Mining Survey”. In: *Ijctee* 1.3 (2008), pp. 114–121.

[Wan+18] Yanshan Wang et al. “A comparison of word embeddings for the biomedical natural language processing”. In: *Journal of Biomedical Informatics* 87 (2018), pp. 12–20. ISSN: 15320464. DOI: 10.1016/j.jbi.2018.09.008. arXiv: 1802.00400.

[WK92] Jonathan J. Webster and Chunyu Kit. “Tokenization as the initial phase in NLP”. In: *Proceedings of the 14th conference on Computational linguistics -* -. Vol. 4. Morristown, NJ, USA: Association for Computational Linguistics, 1992, p. 1106. DOI: 10.3115/992424.992434. URL: http://portal.acm.org/citation.cfm?doid=992424.992434.

[Wol95] David H. Wolpert. “No free lunch theorems for search”. In: *Technical Report SFI-TR-95-02-010* (1995), pp. 1–38. ISSN: 1089778X. DOI: 10.1145/1389095.1389254. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.7505&rep=rep1&type=pdf.

[Xu+15] Kelvin Xu et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: (2015). ISSN: 19410093. DOI: 10.1109/72.279181. URL: http://arxiv.org/abs/1502.03044.

[XWC18] Frank Z Xing, Roy E Welsch, and Erik Cambria. “Natural language based financial forecasting : a survey”. In: *Artificial Intelligence Review* 50.1 (2018), pp. 49–73. ISSN: 1573-7462. DOI: 10.1007/s10462-017-9588-9.

[Yan+16] Zichao Yang et al. “Hierarchical Attention Networks for Document Classification”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2016), pp. 1480–1489. ISSN: 1522-9653. DOI: 10.18653/v1/N16-1174. URL: http://aclweb.org/anthology/N16-1174.

A. Appendix

The following pages display figures and tables that have been left out in Chapters 1 to 6 in order to increase readability. All illustrations were referenced in the text to support a statement or provide additional information.

A.1. Figures and Tables Supporting the Approach

The figures and tables in this section support the statements in Chapter 4. Many of them provide more insights into the datasets new cars and old cars that were extracted from an online car market to test my approach.

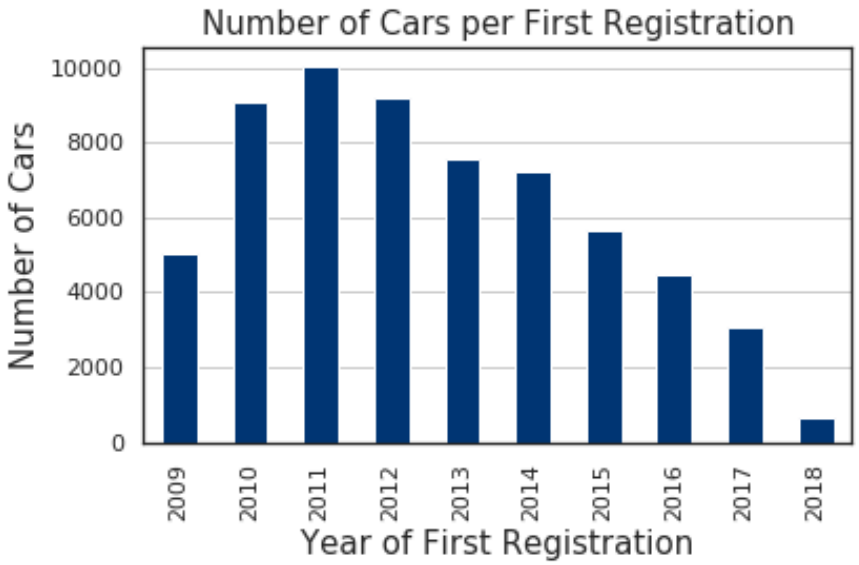


Figure A.1.: The plot shows the amount of cars per year of first registration for the new cars dataset. Most cars in this set had their first registration between 2010 and 2012, so they were 8 to 6 years old. The dataset contains 61,943 vehicle listings overall.

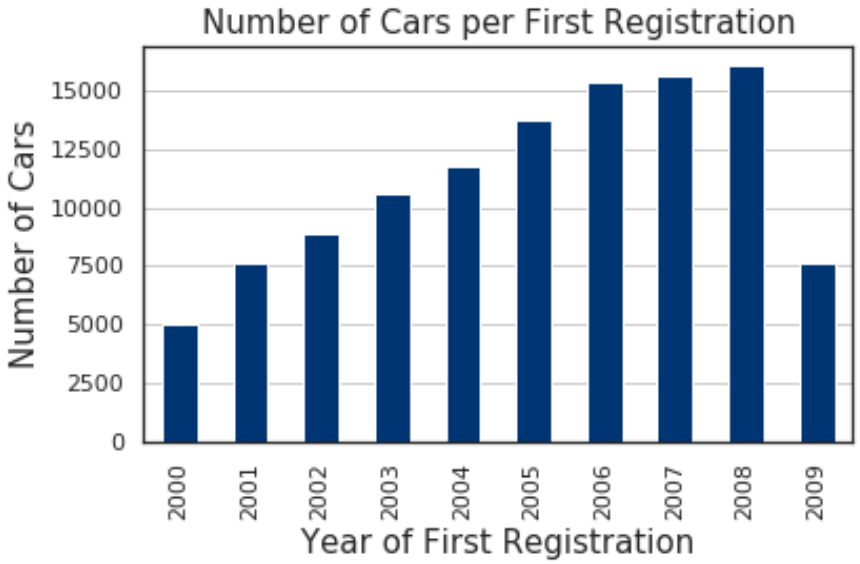


Figure A.2.: The plot shows the amount of cars per year of first registration for the old car dataset. The dataset contains 112,289 vehicle listings overall. The dataset only included cars that had their first registration before June 2009.

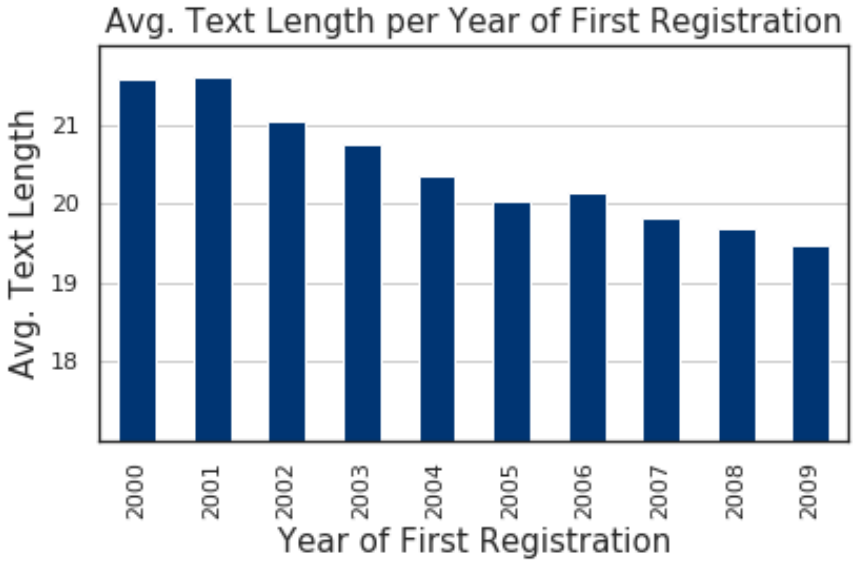


Figure A.3.: The plot shows the average text length of vehicle description texts per year of first registration. It illustrates that older cars do have longer description texts on average. The instances are taken from the second dataset.

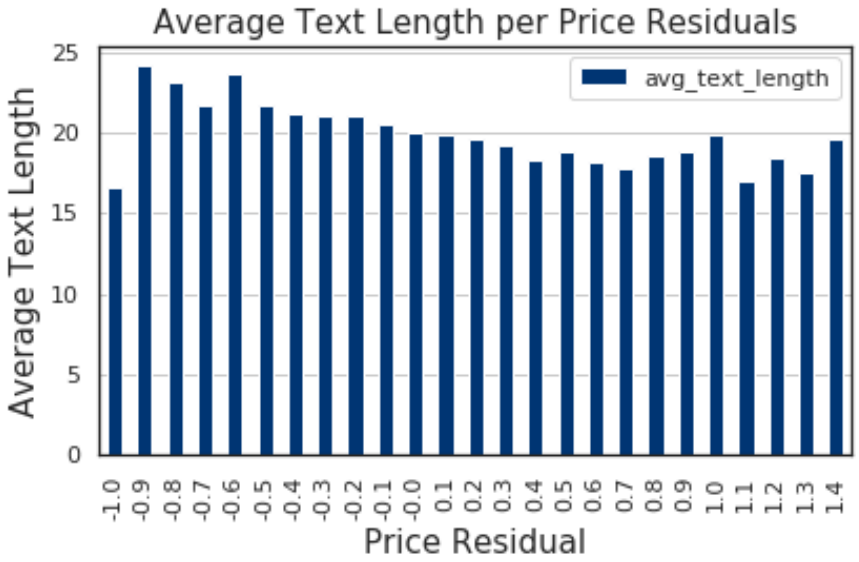


Figure A.4.: The histogram shows the average text length of vehicle description texts per relative price residual PR_{rel}. It illustrates that underpredicted cars do have longer description texts on average than overpredicted vehicles. This implies that longer text should enhance a cars price. The leftest bar only contains 35 cars and I assume that they are outlier. The instances are taken from the second dataset.

Price Range:	$min(price_{pred})$	$max(price_{pred})$	$\mu(price_{pred})$	$\sigma(price_{pred})$
low	1,166.00 €	2,914.00 €	2,087.88 €	444.93 €
mid	2,915.00 €	5,527.00 €	4,072.38 €	748.10 €
high	5,528.00 €	155,154.00 €	10,231.42 €	7,191.00 €

Table A.1.: The table shows price ranges, mean and standard deviation of three different price sets. The older car dataset was split into a low, mid and high price range. 33.3% of the cars are in each price set. This split was done to investigate whether vehicle description texts do have different impacts on a low, mid or high priced car.

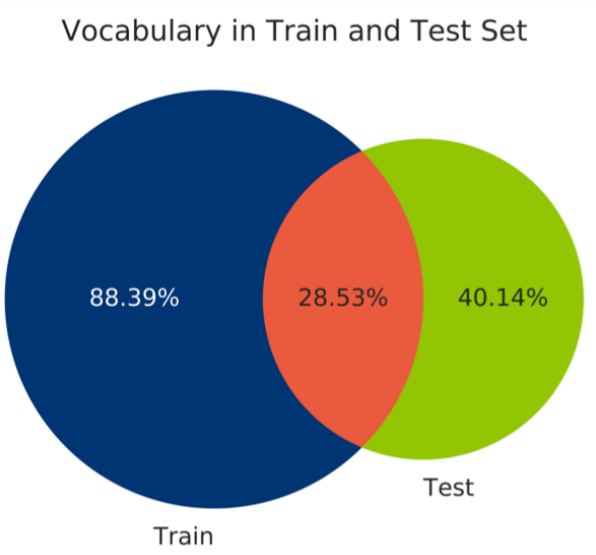


Figure A.5.: The Venn diagram illustrates the vocabulary in the test and train set of the old car dataset. The red area is the common vocabulary. 88.39% are contained in the training set. Only 28.53% of the vocabulary occur in the test and the train set. Overall 2,276,870 words appear in the vehicle description texts and 2,217,651 words are shared in the text and train set. This means that 97.40% of the words are covered by 28.53% of the vocabulary.

A.2. Tables Supporting Evaluation

The following tables show additional results omitted from Chapter 5. Table A.2 illustrates the performance of a FNN with one hidden layer and L2-regularization. In addition, the outcome of the grid search for the hyperparameters of the LSTM is shown in Table 5.6.

Weight Decay Grid Search - Maximum Accuracy on the Test Set - Early Stopping	
λ =	**Accuracy:**
0.003	42.99%
0.001	44.99%
0.0003	45.04%
0.0001	**45.25%**
0.00003	45.22%

Table A.2.: To prevent overfitting and to improve generalization, a weight decay penalty term is added when updating weights of the network. I test five different values for λ and used the L2-regularization term. The network is trained using a hidden dimension of 32, a batch size of 512 and a learning rate of 0.003. $\lambda = 0.0001$ performs best with an accuracy of 45.25%.

Maximum Accuracy on the Test Set - Early Stopping					
Batch Size	**Dimension of Hidden State**	**Learning Rate η**			
		0.0001	**0.0003**	0.001	0.003
128	64	43.90%	44.29%	44.21%	43.50%
	128	44.12%	44.43%	43.91%	43.67%
	256	44.41%	44.47%	44.45%	43.39%
	512	44.28%	44.02%	44.26%	44.13%
256	64	44.06%	44.41%	44.28%	44.17%
	128	44.41%	44.63%	44.39%	44.18%
	256	44.77%	44.34%	44.57%	44.07%
	512	44.47%	44.01%	44.46%	43.97%
512	32	42.23%	43.33%	44.08%	43.33%
	64	44.71%	**44.98%**	44.54%	44.34%
	128	43.86%	44.92%	44.23%	44.47%
	256	44.77%	44.39%	44.41%	44.29%
	512	44.67%	44.45%	44.44%	44.49%
1024	32	41.09%	43.32%	43.25%	43.13%
	64	44.29%	43.52%	44.23%	44.23%
	128	44.88%	44.48%	44.97%	44.06%

Table A.3.: The table shows the results of the grid search done for the LSTM. The highest accuracy on the test set of 44.98% is achieved with a training batch size of 512, a hidden state dimension of 64 and $\eta = 0.0003$. The LSTM reaches this accuracy after 17 training iterations.